MPC7410/MPC7400 RISC Microprocessor Reference Manual

Supports
MPC7410
MPC7400

MPC7410UM/D
10/2008
Rev. 2

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

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

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

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

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

Contents

Paragraph Number | Title | Page Number

About This Book

Chapter 1 Overview

Contents

Paragraph Number | Title | Page Number

Chapter 2
Programming Model

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number | Title | Page Number

Chapter 3
L1 and L2 Cache Operation

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number	Title	Page Number

Contents

Paragraph Number — Title — Page Number

Chapter 4
Exceptions

Contents

Paragraph Number | Title | Page Number

Chapter 5
Memory Management

Chapter 6
Instruction Timing

Contents

Paragraph Number | Title | Page Number

Contents

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

Chapter 7
AltiVec Technology Implementation

Chapter 8
Signal Descriptions

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number | Title | Page Number

Chapter 9
System Interface Operation

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number | Title | Page Number

Contents

Paragraph Number | Title | Page Number

Chapter 10
Power Management

Chapter 11
Performance Monitor

Contents

Paragraph Number | Title | Page Number

Appendix A
MPC7410 Instruction Set Listings

Appendix B
Instructions Not Implemented

Appendix C
Revision History

Contents

Paragraph Number	Title	Page Number

Figures

Figures

Figure Number | Title | Page Number

Figures

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

Figures

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

Tables

Tables

Table Number | Title | Page Number

Tables

Table Number — Title — Page Number

Tables

Tables

Table Number | Title | Page Number

Tables

Tables

Tables

Table Number	Title	Page Number

About This Book

The primary objective of this reference manual is to describe the functionality of the MPC7410 for software and hardware developers. In addition, this manual supports the MPC7400. The MPC7410 is a processor built on Power Architecture™ technology, using the original PowerPC™ instruction set architecture. This book is written from the perspective of the MPC7410, and unless otherwise noted, the information also applies to the MPC7400, which has the same functionality as the MPC7410. Any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are in the hardware specifications.

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

> This manual describes MPC7410 features not defined by the architecture and is to be used with the *Programming Environments Manual*.
>
> Because the 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 architecture's three levels, or programming environments, which are as follows:

- 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.
- Virtual environment architecture (VEA)—The VEA, which is the smallest component of the 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.

- 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 MPC7410 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 MPC7410 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 architecture.

The PowerPC Architecture: A Specification for a New Family of RISC Processors defines the architecture from the perspective of the three programming environments and remains the defining document for the PowerPC architecture. For information on ordering Freescale documentation, see "Related Documentation," on page 38.

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

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

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products for the MPC7410 and the MPC7400. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of RISC processing, and details of the 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 architecture and the MPC7410. This chapter describes the flexible nature of the architecture definition and provides an overview of how the architecture defines the register set, operand conventions, addressing modes, instruction set, cache model, exception model, and memory management model.
- Chapter 2, "Programming Model," is useful for software engineers who need to understand the MPC7410-specific registers, operand conventions, and details regarding how PowerPC instructions are implemented on the MPC7410. Instructions are organized by function.
- Chapter 3, "L1 and L2 Cache Operation," discusses the cache and memory model as implemented on the MPC7410.
- Chapter 4, "Exceptions," describes the exception model defined in the OEA and the specific exception model implemented on the MPC7410.
- Chapter 5, "Memory Management," describes the MPC7410'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 MPC7410.
- Chapter 9, "System Interface Operation," describes signal timings for various operations. It also provides information for interfacing to the MPC7410.
- Chapter 10, "Power Management," provides information about power saving and thermal management modes for the MPC7410.
- Chapter 11, "Performance Monitor," describes the operation of the performance monitor diagnostic tool incorporated in the MPC7410.
- Appendix A, "MPC7410 Instruction Set Listings," lists all PowerPC instructions while indicating those instructions that are not implemented by the MPC7410; it also includes the instructions that are specific to the MPC7410. 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 MPC7410.
- Appendix C, "Revision History," lists corrections to the previous versions of this manual and the MPC7400 User's Manual Rev. 0.
- 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 architecture.

General Information

The following documentation, available through Morgan-Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA, provides useful information about the Power Architecture technology 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
- *Power Architecture™ Technology Primer* (PWRARCPRMRM), available from the Freescale website.

Related Documentation

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

- *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (MPEFPC32B/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 described in this book.
- 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.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale processors.

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

Conventions

This 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
0 0 0 0	Indicates reserved bits or bit fields in a register. Although these bits can be written to as ones or zeros, they are always read as zeros.

Acronyms and Abbreviations

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

Table i. Acronyms and Abbreviated Terms

Term	Meaning
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
dRLDB	Data reload buffer
dRLT	Data reload table
DSISR	Register used for determining the source of a DSI exception
DTLB	Data translation lookaside buffer
EA	Effective address
EAR	External access register
ECC	Error checking and correction
FIFO	First-in-first-out

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
FPR	Floating-point register
FPSCR	Floating-point status and control register
FPU	Floating-point unit
GPR	General-purpose register
HID*n*	Hardware implementation-dependent register
IABR	Instruction address breakpoint register
IBAT	Instruction BAT
ICTC	Instruction cache throttling control register
IEEE	Institute for Electrical and Electronics Engineers
IMMU	Instruction MMU
IQ	Instruction queue
iRLDB	Instruction reload buffer
iRLT	Instruction reload table
ITLB	Instruction translation lookaside buffer
IU	Integer unit
JTAG	Joint Test Action Group
L1OQ	Level 1 operation queue
L2	Secondary cache (level 2 cache)
L2CR	Level 2 cache control register
L2PMCR	Level 2 private memory control register
LFQ	Load fold queue
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
MMCR*n*	Monitor mode control registers
MMU	Memory management unit
MSB	Most-significant byte

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
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
SRU	System register unit
TAU	Thermal assist unit
TB	Time base facility

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
TBL	Time base lower register
TBU	Time base upper register
THRM*n*	Thermal management registers
TLB	Translation lookaside buffer
TTL	Transistor-to-transistor logic
UIMM	Unsigned immediate value
UISA	User instruction set architecture
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.

Table ii. Terminology Conventions

Architecture Specification	This Manual
Data storage interrupt (DSI)	DSI exception
Extended mnemonics	Simplified mnemonics
Fixed-point unit (FXU)	Integer unit (IU)

Table ii. Terminology Conventions (continued)

Architecture Specification	This Manual
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

Architecture Specifications	Equivalencies
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 MPC7410 microprocessor features, including a block diagram showing the major functional components. It also provides information about how the MPC7410 implementation complies with the Power Architecture™ and AltiVec™ definitions. This manual also supports the MPC7400 microprocessor which for the most part has the same functionality as the MPC7410. Any differences between the two microprocessors are specifically noted in this user's manual. Note that the bus timing, AC, DC, mechanical, and thermal characteristics for each microprocessor are detailed in their respective hardware specifications.

1.1 General Operation

This section describes the features and general operation of the MPC7410 and provides a block diagram showing major functional units. The MPC7410 is a reduced instruction set computer (RISC) microprocessor that implements the PowerPC™ instruction set architecture (PowerPC ISA), built on Power Architecture technology. The MPC7410 implements the 32-bit portion of 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 MPC7410 also implements the AltiVec instruction set architectural extension.

The MPC7410 is a superscalar processor that can dispatch and complete two instructions simultaneously. It incorporates the following execution units:

- Floating-point unit (FPU)
- Branch processing unit (BPU)
- System register unit (SRU)
- Load/store unit (LSU)
- Two integer units (IUs):
  - IU1 executes all integer instructions.
  - IU2 executes all integer instructions except multiply and divide instructions.
- Two vector units that support AltiVec instructions:
  - Vector permute unit (VPU)
  - Vector arithmetic logic unit (VALU), which consists of the following independent subunits:
    - Vector simple integer unit (VSIU)
    - Vector complex integer unit (VCIU)
    - 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 MPC7410-based systems. Most integer instructions (including VSIU instructions) have a one-clock cycle execution latency.

The FPU and VFPU are pipelined; that is, the tasks they perform are broken into subtasks executed in successive stages. Typically, a floating-point instruction occupies only one of the three FPU stages at a time, freeing the previous stage to operate on the next floating-point instruction. Thus, three floating-point instructions can be in the FPU execute stage at a time and one floating-point instruction can finish executing per processor clock cycle.

The VFPU has four pipeline stages when executing in non-Java mode and five when executing in Java mode.

Note that for the MPC7410, double- and single-precision versions of floating-point instructions have the same latency. For example, a floating-point multiply-add instruction takes three cycles to execute, regardless of whether it is single- (**fmadds**) or double-precision (**fmadd**).

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

The MPC7410 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 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 and L2 Cache Operation.”

The L2 cache is implemented with an on-chip, two-way, set-associative tag memory, and with external, synchronous SRAMs for data storage. The external SRAMs are accessed through a dedicated L2 cache port that supports a single bank of 256 Kbytes, 512 Kbytes, 1 Mbyte, or 2 Mbytes of synchronous SRAMs. On the MPC7410, the L2 interface can be configured to use half (256 Kbytes minimum) or all of the SRAM area as a direct-mapped, private memory space. Note that the MPC7400 does not support this private memory functionality on its L2 interface. For information about the L2 cache implementation, see Chapter 3, “L1 and L2 Cache Operation.”

The MPC7410 has four software-controllable power-saving modes. Three static modes, doze, nap, and sleep, progressively reduce power dissipation. When functional units are idle, a dynamic power management mode causes those units to enter a low-power mode automatically without affecting operational performance, software execution, or external hardware. The MPC7400 provides an additional thermal assist unit (TAU) and a way to reduce the instruction fetch rate for limiting power dissipation. Note that the MPC7410 does not provide a thermal assist unit. Power management is described in Chapter 10, “Power Management.”

The MPC7410 uses an advanced CMOS process technology and is fully compatible with TTL devices.

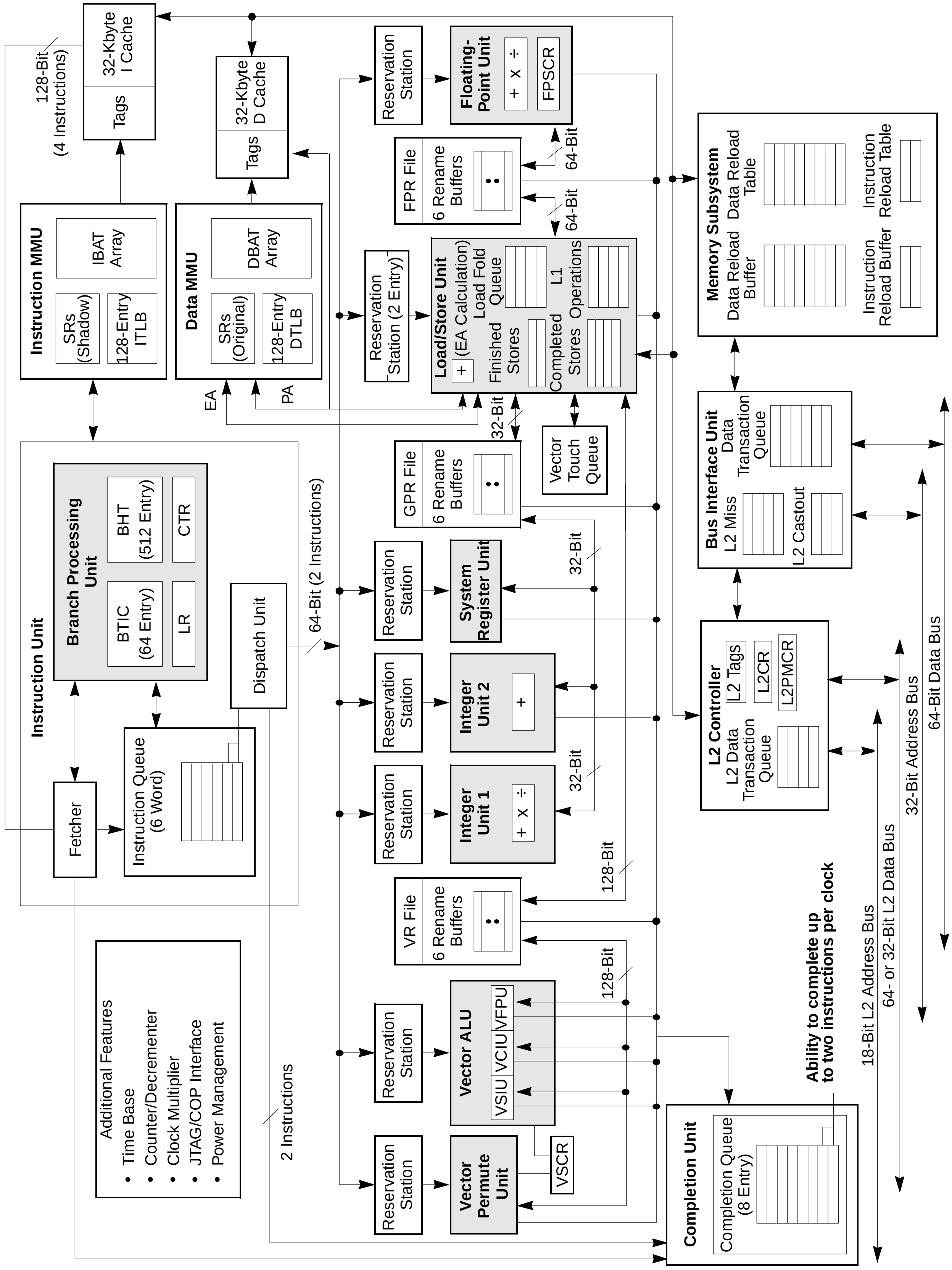


Figure 1-1. MPC7410 Microprocessor Block Diagram

1.2 General Features

This section describes the features of the MPC7410. The interrelationships of these features are shown in Figure 1-1.

1.2.1 Overview of Features

Major features of the MPC7410 are as follows:

- High-performance, superscalar microprocessor
  - As many as four instructions can be fetched from the instruction cache per clock cycle
  - As many as two instructions can be dispatched per clock
  - As many as eight instructions can execute per clock (including two integer instructions and four AltiVec instructions)
  - Single-clock-cycle execution for most instructions
  - One instruction per clock throughput for most instructions
- Eight independent execution units and three register files
  - Branch processing unit (BPU) features static and dynamic branch prediction
    - 64-entry (16-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, if a fetch access hits the BTIC, it provides the first two instructions in the target stream.
    - 512-entry branch history table (BHT) with two bits per entry for four levels of prediction—not-taken, strongly not-taken, taken, strongly taken
    - Branch instructions that do not update the count register (CTR) or link register (LR) are removed from the instruction stream.
  - Two integer units (IUs) that share 32 GPRs for integer operands
    - IU1 can execute any integer instruction.
    - IU2 can execute all integer instructions except multiply and divide instructions (shift, rotate, arithmetic, and logical instructions). Most instructions that execute in IU2 take one cycle to execute. The IU2 has a single-entry reservation station.
  - Three-stage FPU and a 32-entry FPR file
    - Fully compliant with IEEE Std. 754™-1985 FPU for both single- and double-precision operations
    - Supports non-IEEE mode for time-critical operations
    - Hardware support for denormalized numbers
    - Single-entry reservation station
    - Thirty-two 64-bit FPRs for single- or double-precision operands
  - Two vector units and 32-entry vector register file (VRs)
    - Vector permute unit (VPU)

    - Vector arithmetic logic unit (VALU), which consists of the three independent subunits: vector simple integer unit (VSIU), vector complex integer unit (VCIU), and vector floating-point unit (VFPU)
  - Two-stage 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
    - Two-entry reservation station
    - Single-cycle, pipelined load or store cache accesses (byte, half word, double word, quad word) including misaligned accesses within a 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
    - Hits under misses (multiple outstanding misses) supported
    - Six-entry store queue
    - Sequencing for load/store multiples and string operations
    - Supports both big- and little-endian modes, including misaligned little-endian accesses
  - SRU handles miscellaneous instructions
    - Executes CR logical and move to/move from SPR instructions (**mtspr** and **mfspr**)
    - Single-entry reservation station
- Rename buffers
  - Six GPR rename buffers
  - Six FPR rename buffers
  - Six VR rename buffers
  - Condition register buffering supports two CR writes per clock
- Completion unit
  - The completion unit retires an instruction from the eight-entry reorder buffer (completion queue) when all instructions ahead of it have been completed, the instruction has finished execution, and no exceptions are pending.
  - Guarantees sequential programming model (precise exception model)
  - Monitors all dispatched instructions and retires them in order
  - Tracks unresolved branches and flushes instructions from the mispredicted branch
  - Retires as many as two instructions per clock
- Separate on-chip L1 instruction and data caches (Harvard architecture)
  - 32-Kbyte, eight-way set-associative instruction and data caches
  - Pseudo least-recently-used (PLRU) replacement algorithm

- 32-byte (eight-word) L1 cache block
  - Physically indexed/physical tags
  - Cache write-back or write-through operation programmable on a per-page or per-block basis
  - Instruction cache can provide four instructions per clock; data cache can provide four words per clock
  - Caches can be disabled in software
  - Caches can be locked in software
  - Data cache coherency (MEI, MESI, and MERSI) maintained in hardware
  - Separate copy of data cache tags for efficient snooping
  - No snooping of instruction cache except for **icbi** instruction
  - Data cache supports AltiVec LRU and transient instructions, as described in Section 1.3.2.2, “AltiVec Instruction Set.”
  - The critical double word is made available to the requesting unit when it is burst into the reload data queue. The caches are nonblocking, so they can be accessed during this operation.
- Level 2 (L2) cache interface
  - On-chip two-way set-associative L2 cache controller and tags
  - External data SRAMs
  - Support for 256-Kbyte, 512-Kbyte, 1-Mbyte, and 2-Mbyte L2 caches
  - Copyback or write-through data cache (on a per page basis, or for all L2)
  - 32-byte (256 K and 512 K), 64-byte (1 M), or 128-byte (2 M) sectored line size
  - Direct-mapped, private memory capability for half (256 Kbytes minimum) or all of the L2 SRAM space (MPC7410-only; not supported by the MPC7400)
  - Supports pipelined (register-register) synchronous burst SRAMs, PB3 pipelined (register-register) synchronous burst SRAMs, and pipelined (register-register) late-write synchronous burst SRAMs
  - Configurable core-to-L2 frequency divisors
  - Configurable for 64- or 32-bit L2 data bus (MPC7410-only; 32-bit L2 data bus not supported by the MPC7400)
- Separate memory management units (MMUs) for instructions and data
  - 52-bit virtual address; 32-bit physical address
  - Address translation for 4-Kbyte pages, variable-sized blocks, and 256-Mbyte segments
  - Memory programmable as write-back/write-through, cacheable/noncacheable, and coherency enforced/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-reloadable (that is, the page table search is performed in hardware)
- Efficient data flow

- All data buses between VRs, LSU, L1 caches, L2 cache controller, and the bus interface unit are 128 bits wide
    - The L1 data cache is fully pipelined to provide 128 bits/cycle to/from the VRs
    - L2 is fully pipelined to provide 64 (MPC7410 and MPC7400) or 32 (MPC7410-only) bits per L2 clock cycle to the L1 caches
    - Up to 8 outstanding, out-of-order, cache misses allowed between the L1 data cache and L2/bus
    - Up to seven out-of-order transactions on the bus, one in progress and six pending
    - Load folding to fold new L1 data cache misses into older, outstanding load and store misses to the same line
    - Store miss 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).
    - Two-entry finished store queue and 4-entry completed store queue between the LSU and the L1 data cache
    - Separate additional queues for efficient buffering of outbound data (such as cast outs and write throughs) from the L1 data cache and L2
- Multiprocessing support features include the following:
    - Hardware-enforced, cache coherency protocols for data cache
        - 3-state (MEI) similar to the MPC750
        - 4-state (MESI) similar to the MPC604
        - 5-state (MERSI), where the new Recent (R) state allows shared intervention
    - Load/store with reservation instruction pair for atomic memory references, semaphores, and other multiprocessor operations
- Power and thermal management
    - Three static modes, doze, nap, and sleep, progressively reduce power dissipation:
        - Doze—All the functional units are disabled except for the time base/decrementer registers and the bus snooping logic.
        - Nap—The nap mode further reduces power consumption by disabling bus snooping, leaving only the time base register and the PLL in a powered state.
        - Sleep—All internal functional units are disabled, after which external system logic may disable the PLL and SYSCLK.
    - On the MPC7400, a thermal management facility provides software-controllable thermal management. Thermal management is performed through the use of three supervisor-level registers and an MPC7400-specific thermal management exception. The MPC7410 does not support the thermal management facility.
    - Instruction cache throttling provides control of instruction fetching to limit power consumption.
- 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

1.2.2 Instruction Flow

As shown in Figure 1-1, the MPC7410 instruction unit provides centralized control of instruction flow to the execution units. The instruction unit contains a sequential fetcher, six-entry instruction queue (IQ), dispatch unit, and BPU. It determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

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 MPC7410-specific dynamic branch prediction or the architecture-defined static branch prediction.

Branch instructions that do not affect the LR or CTR are removed from the instruction stream. The BPU folds branch instructions when a branch is taken (or predicted as taken); branch instructions that are not taken, or predicted as not taken, are removed from the instruction stream through the dispatch mechanism.

Instructions issued 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.

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

1.2.2.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 1-1, holds as many as six instructions and loads up to four instructions from the instruction cache during a single processor clock cycle. The instruction fetcher continuously attempts to load as many instructions as there were vacancies in the IQ in the previous clock cycle. All instructions except branch, Return from Exception (**rfi**), System Call (**sc**), and Instruction Synchronize (**isync**) instructions are dispatched to their respective execution units from the bottom two positions in the instruction queue (IQ0 and IQ1) at a maximum rate of two instructions per cycle. Reservation stations are provided for the IU1, IU2, FPU, LSU, SRU, VPU, and VALU. The dispatch unit checks for source and destination register dependencies, determines whether a position is available in the completion queue, and inhibits subsequent instruction dispatching as required.

Branch instructions can be detected, decoded, and predicted from anywhere in the instruction queue. For a more detailed discussion of instruction dispatch, see Section 6.3.4, “Instruction Dispatch and Completion Considerations.”

1.2.2.2 Branch Processing Unit (BPU)

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

Unconditional branch instructions and conditional branch instructions in which the condition is known can be resolved immediately. For unresolved conditional branch instructions, the branch path is predicted using either the architecture-defined static branch prediction or the MPC7410-specific dynamic branch prediction. Dynamic branch prediction is enabled if HID0[BHT] = 1.

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. The MPC7410 allows a second branch instruction to be predicted; instructions from the second predicted instruction stream can be fetched but cannot be dispatched.

Dynamic prediction is implemented using a 512-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 MPC7410 executes instructions from the predicted target stream although the results are not committed to architected registers until the conditional branch is resolved. This execution can continue until a second unresolved branch instruction is encountered.

When a branch is taken (or predicted as taken), the instructions from the untaken path must be flushed and the target instruction stream must be fetched into the IQ. The BTIC is a 64-entry, four-way set associative cache that contains the most recently used branch target instructions, typically in pairs. When an instruction fetch hits in the BTIC, the instructions arrive in the instruction queue in the next clock cycle, a clock cycle sooner than they would arrive from the instruction cache. Additional instructions arrive from the instruction cache in the next clock cycle. The BTIC reduces the number of missed opportunities to dispatch instructions and gives the processor a 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 into the LR for certain types of branch instructions. The LR also contains the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction. The CTR contains the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction. Because the LR and CTR are 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 eight-entry completion queue. The completion unit tracks instructions from dispatch through execution and retires them in program order from the two bottom entries in the completion queue (CQ0 and CQ1).

Instructions cannot be dispatched to an execution unit unless there is a vacancy in the completion queue. Branch instructions that do not update the CTR or LR are removed from the instruction stream and do not take an entry in the completion queue. Instructions that update the CTR and LR follow the same dispatch and completion procedures as non-branch instructions, except that they are not issued to an execution unit.

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 MPC7410 must recover from a mispredicted branch or any exception. An instruction is retired as it is removed from the completion queue.

For a more detailed discussion of instruction completion, see Section 6.3.4, “Instruction Dispatch and Completion Considerations.”

1.2.2.4 Independent Execution Units

In addition to the BPU, the MPC7410 provides the seven execution units described in the following sections.

1.2.2.4.1 AltiVec Vector Permute Unit (VPU)

The VPU performs the following permutations on vector operands:

- Pack—Vector pack instructions truncate the contents of two concatenated source operands (grouped as eight words or sixteen half words) into a single result of eight half words or sixteen bytes, respectively.
- Unpack—Vector unpack instructions unpack the eight low or high bytes (or four low or high half words) of one source operand into eight half words (or four words) using sign extension to fill the most significant bytes (MSBs).
- Merge—Byte vector merge instructions interleave the eight low bytes (or eight high bytes) from two source operands producing a result of 16 bytes. Similarly, half-word vector merge instructions interleave the four low half words (or four high half words) of two source operands producing a result of eight half words, and word vector merge instructions interleave the two low words (or two high words) from two source operands producing a result of four words. The vector merge instruction has many uses, and it can be used to efficiently transpose SIMD vectors.
- Splat—Vector splat instructions prepare vector data for operations in which one source vector is to consist of elements that all have the same value (for example, multiplying all elements of a vector register by a constant). Vector splat instructions also can move data. For example, to multiply all elements of a vector register by a constant, the vector splat instructions can be used to splat the scalar into a vector register. Likewise, when storing a scalar into an arbitrary memory location, it must be splatted into a vector register, and that register 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.
- Permute—Permute instructions allow any byte in any two source vector registers 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 to be taken. The Vector Permute (**vperm**) instruction provides many useful functions. For example, it can be used efficiently to perform table lookups and data alignment. For an example of how to align data, see Section 3.1.6, “Quad-Word Data Alignment,” in the *AltiVec Technology Programming Environments Manual*.
- Select—Data flow in the vector unit can be controlled without branching by using a vector compare instruction and the vector select (**vsel**) instruction. In this case, the compare result vector is used directly as a mask operand of a vector select instruction. 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.

These instructions are described in detail in Chapter 2, “Addressing Modes and Instruction Set Summary,” in the *AltiVec Technology Programming Environments Manual*.

1.2.2.4.2 AltiVec Vector Arithmetic Logic Unit (VALU)

As shown in Figure 1-1, the VALU consists of the following three independent subunits:

- Vector simple integer unit (VSIU)—executes simple vector integer computational instructions, such as addition, subtraction, maximum and minimum comparisons, averaging, rotation, shifting, comparisons, and Boolean operations
- Vector complex integer unit (VCIU)—executes longer-latency vector integer instructions, such as multiplication, multiplication/addition, and sum-across with saturation
- Vector floating-point unit (VFPU)—executes all vector floating-point instructions

Although only one instruction can be dispatched to the VALU per processor clock cycle, all three subunits can execute simultaneously. For example, if instructions are dispatched one at a time to the VFPU, VCIU, and VSIU, all three subunits can be executing separate instructions, and, if enough VR rename resources are available, two of them can write back their results in the same clock cycle.

1.2.2.4.3 Integer Units (IUs)

The integer units IU1 and IU2 are shown in Figure 1-1. The IU1 can execute any integer instruction; IU2 can execute any integer instruction except multiplication and division instructions. Each IU has a single-entry reservation station that can receive instructions from the dispatch unit and operands from the GPRs or the rename buffers.

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

The IU1 has a 32-bit integer multiplier/divider as well as the adder, shift, and logical units of the IU2. The multiplier supports early exit for operations that do not require full 32- x 32-bit multiplication.

Each IU has a dedicated result bus (not shown in Figure 1-1) that connects to rename buffers.

1.2.2.4.4 Floating-Point Unit (FPU)

The FPU, shown in Figure 1-1, is designed such that single-precision operations require only a single pass, with a latency of three cycles. As instructions are dispatched to the FPU’s reservation station, source operand data can be accessed from the FPRs or from the FPR rename buffers. Results in turn are written to the rename buffers and are made available to subsequent instructions. Instructions pass through the reservation station in dispatch 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 MPC7410 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 may occur after three consecutive, independent floating-point arithmetic instructions 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 six floating-point rename registers. The MPC7410 writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The MPC7410 supports all IEEE Std. 754 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software exception routines.

1.2.2.4.5 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 calculates effective addresses, performs data alignment, and provides sequencing for load/store string and multiple instructions.

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 cycle from the perspective of the LSU, with a two-cycle total latency on a cache hit. 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 MPC7410 executes store instructions with a maximum throughput of one per cycle and a three-cycle total latency to the data cache. The time required to perform the actual load or store operation depends on the processor/bus clock ratio and whether the operation involves the on-chip cache, the L2 cache, system memory, or an I/O device.

1.2.2.4.6 System Register Unit (SRU)

The SRU executes various system-level instructions, as well as condition register logical operations and move to/from special-purpose register instructions. To maintain system state, most instructions executed by the SRU are execution-serialized; that is, the instruction is held for execution in the SRU until all previously issued instructions have executed. Results from execution-serialized instructions executed by the SRU are not available or forwarded for subsequent instructions until the instruction completes.

1.2.3 Memory Management Units (MMUs)

The MPC7410's MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 4 Gigabytes (2^{32}) of physical memory for instructions and data. The MMUs control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to support demand-paged virtual memory systems.

The LSU 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 MPC7410 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.
- 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)

If translation is enabled, the appropriate MMU translates the higher-order bits of the effective address into physical address bits. The lower-order address bits (that are untranslated and therefore, considered both logical and physical) are directed to the on-chip caches where they form the index into the 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-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 searches for one in the page table following the model defined by the 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 MPC7410’s instruction and data TLBs are 128-entry, two-way set-associative caches that contain address translations. The MPC7410 automatically generates a search of the page tables in memory on a TLB miss.

1.2.4 On-Chip Instruction and Data Caches

The MPC7410 implements separate L1 instruction and data caches. Each cache is 32-Kbyte and eight-way set associative. As defined by the architecture, they are physically indexed. Each cache block contains eight contiguous words from memory that are loaded from an 8-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 simultaneously written to the cache and forwarded to the requesting unit, minimizing stalls due to load delays. The cache being loaded is not blocked to internal accesses while the load completes.

The MPC7410 cache organization is shown in Figure 1-2.

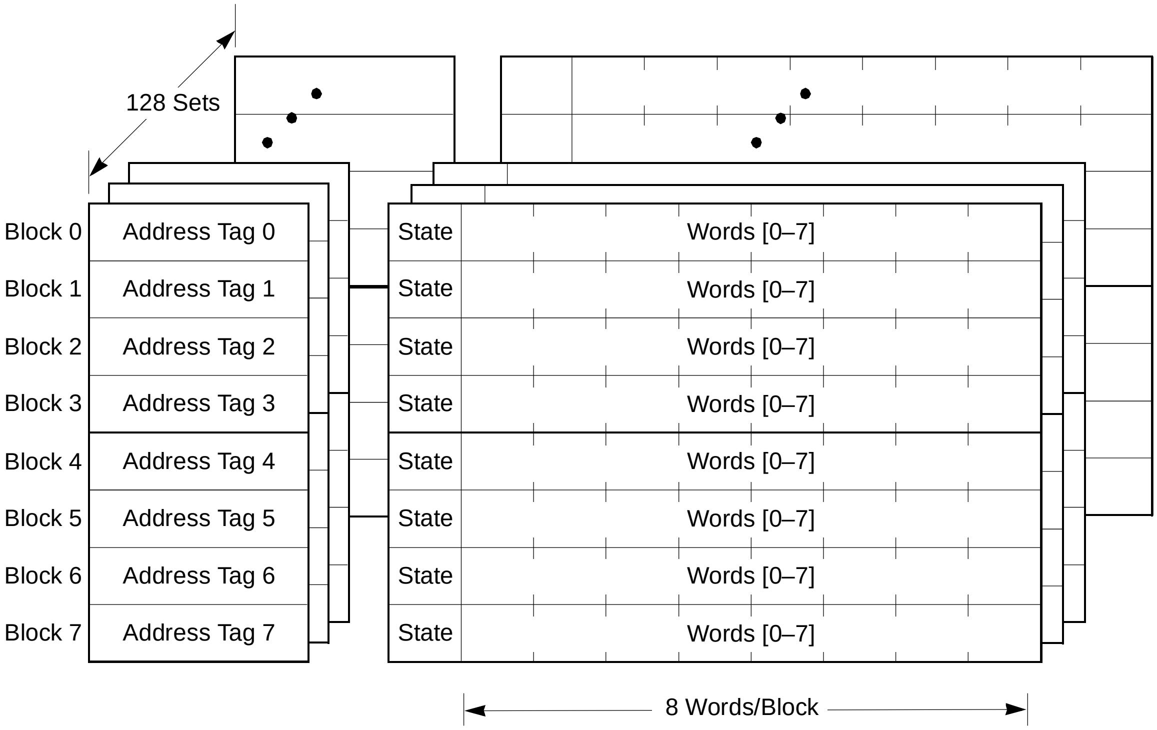


Figure 1-2. L1 Cache Organization

The instruction cache provides up to four instructions per 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 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 MPC7410 also implements a 64-entry (16-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 two instructions in the target stream. The BTIC can be disabled and invalidated through software.

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 L2 cache is implemented with an on-chip, two-way, set-associative tag memory, and with external, synchronous SRAMs for data storage. The external SRAMs are accessed through a dedicated L2 cache port that supports a single bank of 256-Kbyte, 512-Kbyte, 1-Mbyte, or 2-Mbyte synchronous SRAMs. The L2 cache normally operates in write-back mode and supports system cache coherency through snooping.

Depending on its size, the L2 cache is organized into 32-, 64-, or 128-byte lines. Lines are subdivided into 32-byte sectors (blocks), the unit at which cache coherency is maintained.

The L2 cache controller contains the L2 cache control register (L2CR), which includes bits for enabling parity checking, setting the L2-to-processor clock ratio, and identifying the type of RAM used for the L2 cache implementation. The L2 cache controller also manages the L2 cache tag array, which is two-way set-associative with 8K tags per way. Each sector (32-byte cache block) has its own valid, shared, and modified status bits. The L2 implements the MERSI 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; they are forwarded to the bus interface if they miss.

The L2 cache can accept multiple, simultaneous accesses. The L1 instruction cache can request an instruction at the same time that the L1 data cache is requesting data. The L1 data cache requests are handled through the data reload table (shown in Figure 1-1), which can have up to eight outstanding data cache misses. The L2 cache also services snoop requests from the bus. If there are multiple pending requests to the L2 cache, snoop requests have highest priority. The next priority are load and store requests from the L1 data cache. The next priority are instruction fetch requests from the L1 instruction cache.

On the MPC7410, the L2 interface can be configured to use half (256 Kbytes minimum) or all of the SRAM area as a direct-mapped, private memory space. Note that the MPC7400 does not support this private memory functionality on its L2 interface. 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 L2 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 and L2 Cache Operation.”

1.2.6 System Interface/Bus Interface Unit (BIU)

The MPC7410 processor bus interface is based on the 60x bus, but it includes several features that allow it to provide significantly higher memory bandwidth. The MPC7410 can be configured to support either an MPC750-compatible 60x mode or an expanded bus mode called MPX bus mode.

The MPC7410 has a separate address and data bus, each with its own set of arbitration and control signals. This allows for the decoupling of the data tenure from the address tenure of a transaction, and provides for a wide range of system bus implementations including:

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

The MPC7410 supports only the normal memory-mapped address segments defined in the architecture.

The 60x bus interface has the following features:

- 32-bit address bus (plus 4 bits of odd parity)
- 64-bit data bus (plus 8 bits of odd parity); a 32-bit data bus mode is not provided
- Supports two cache coherency protocols:
  - Three-state (MEI) similar to the MPC750
  - Four-state (MESI) similar to the MPC604
- On-chip snooping to maintain L1 data cache and L2 cache coherency for multiprocessing applications
- Supports address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Support for limited out-of-order transactions
- Support for up to seven transactions (six pending plus one data tenure in progress)
- TTL-compatible interface

In addition to the 60x bus features, to gain increased performance, the MPX bus mode has the following features:

- Increased address bus bandwidth by eliminating dead cycles under some circumstances
- Full data streaming for burst reads and burst writes
- Increased levels of address pipelining
- Support for full out-of-order transactions
- Support for data intervention in multiprocessing systems
- Support for third cache coherency protocol: Five-state (MERSI), where the new R state allows shared intervention
- Improved electrical timings (for example, programmable option for keeping address bus driven)

1.2.6.1 System Interface Operation

The primary activity of the MPC7410 system interface is transferring data and instructions between the processor and system memory. There are three types of bus transfers:

- Single-beat transfers—These memory accesses allow transfer sizes of 8, 16, 24, 32, or 64 bits in one bus clock cycle. Single-beat transactions are caused by uncacheable read and write operations that access memory directly (that is, when caching is disabled), cache-inhibited accesses, and stores in write-through mode.
- Two-beat burst (16 bytes) data transfers—Generated to support caching-inhibited or write-through AltiVec loads and stores (only generated in MPX bus mode).
- Four-beat burst (32 byte) data transfers—Initiated when an entire cache block is transferred.

Because the first-level caches on the MPC7410 are write-back caches, burst-read memory operations are the most common memory accesses, followed by burst-write memory operations, and single-beat (noncacheable or write-through) memory read and write operations.

The MPC7410 also supports address-only operations, variants of the burst and single-beat operations (for example, atomic memory operations and global memory operations that are snooped), and address retry activity (for example, when a snooped read access hits a modified block in the cache). Because all I/O is memory-mapped, I/O accesses use the same protocol as memory accesses. The MPX bus also supports data-only operations to provide for data intervention.

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 MPC7410 to be integrated into systems that implement various fairness and bus parking procedures to avoid arbitration overhead.

Typically, memory accesses are weakly ordered—sequences of operations, including load/store string and multiple instructions, do not necessarily execute in the order they begin—maximizing the efficiency of the bus without sacrificing data coherency. The MPC7410 allows read operations to be performed ahead of store operations (except when a dependency exists, or in cases where a noncacheable access is performed). The MPC7410 provides support for a write operation to be performed ahead of a previously queued read data tenure (for example, letting a snoop push be enveloped between address and data tenures of a read operation) in 60x bus mode and full data-tenure reordering in MPX bus mode. Because the MPC7410 can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

The system interface supports address pipelining, which allows the address tenure of one transaction to overlap the data tenure of another. The extent of the pipelining depends on external arbitration and control circuitry. Similarly, the MPC7410 supports split-bus transactions for systems with multiple potential bus masters—one device can have mastership of the address bus while another has mastership of the data bus. Allowing multiple bus transactions to occur simultaneously increases the available bus bandwidth for other activity.

The system interface is specific for each microprocessor implementation.

1.2.6.2 Signal Groupings

The MPC7410 signals are grouped as shown in Figure 1-3. Signals are provided for implementing the bus protocol, clocking and control of the L2 caches, as well as separate L2 address and data buses. Test and control signals provide diagnostics for selected internal circuits.

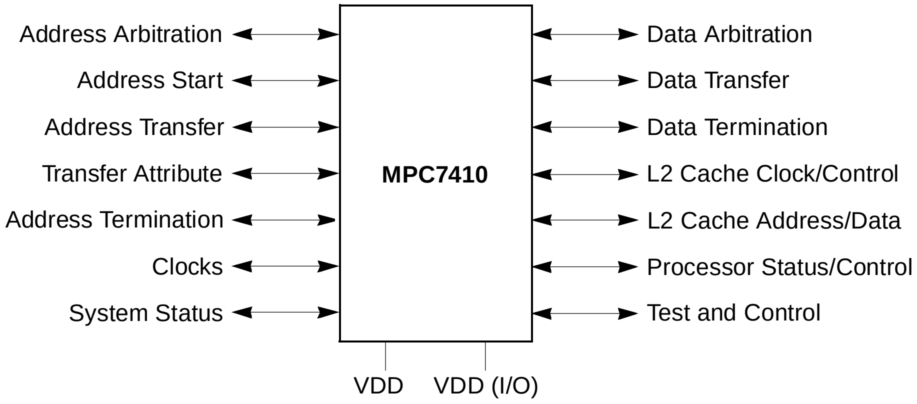


Figure 1-3. System Interface

The signals used for the 60x and the MPX bus protocols are largely identical except that the MPX bus differs in the following ways:

- Does not use the $\overline{\text{ABB}}$ and $\overline{\text{DBB}}$ output signals
- Uses three DTI[0:2] signals instead of a single $\overline{\text{DBWO}}$ signal
- Uses two $\overline{\text{SHD}}$[0:1] signals instead of a single $\overline{\text{SHD}}$ signal

The MPC7410 bus protocol signals are grouped as follows:

- Address arbitration signals—The MPC7410 uses these signals to arbitrate for address bus mastership.
- Address start signals—These signals indicate that a bus master has begun a transaction on the address bus.
- Address transfer signals—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 signals—These signals provide information about the type of transfer, such as the transfer size and whether the transaction is bursted, write-through, or caching-inhibited.
- Address termination signals—These signals are used to acknowledge the end of the address phase of the transaction. They also indicate whether a condition exists that requires the address phase to be repeated.
- Data arbitration signals—The MPC7410 uses these signals to arbitrate for data bus mastership.
- Data transfer signals—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 termination signals—Data termination signals are required after each data beat in a data transfer. In a single-beat transaction, a data termination signal also indicates the end of the tenure; in burst accesses, data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. They also indicate whether a condition exists that requires the data phase to be repeated.

The remaining signals are used for functions other than the bus protocol and they are grouped as follows:

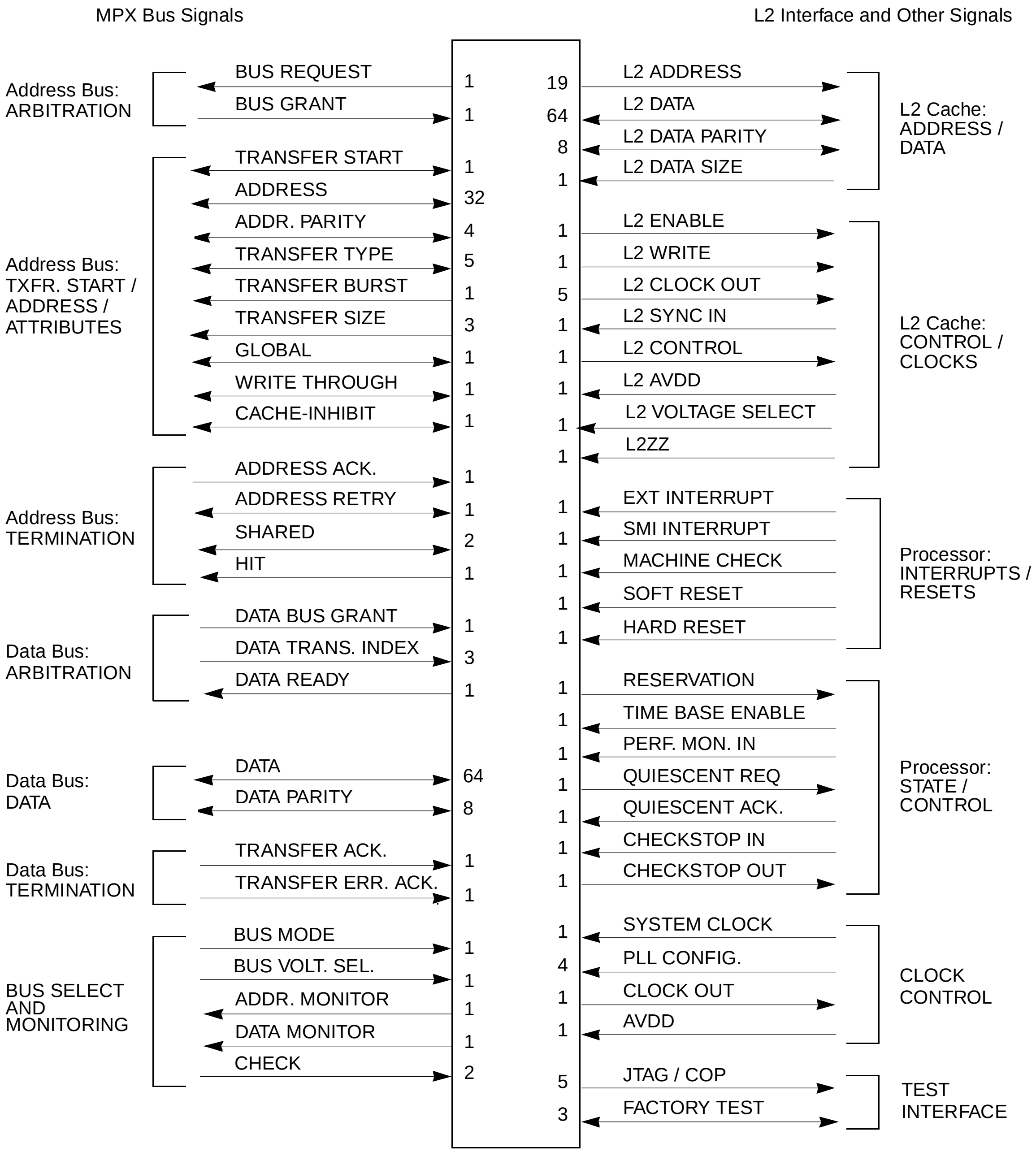
- L2 cache clock/control signals—These signals provide clocking and control for the L2 cache.
- L2 cache address/data—The MPC7410 has separate address and data buses for accessing the L2 cache.
- Interrupt and reset signals—These signals include the interrupt signal, checkstop signals, and both soft reset and hard reset signals. These signals are used to generate interrupt exceptions and, under various conditions, to reset the processor.
- Processor status/control signals—These signals are used to set the reservation coherency bit, enable the time base, and other functions.
- Miscellaneous signals—These signals are used in conjunction with resources such as the time base facility.
- JTAG/COP interface signals—The common on-chip processor (COP) unit provides a serial interface to the system for performing board-level boundary scan interconnect tests.
- Clock signals—These signals determine the system clock frequency. These signals can also be used to synchronize multiprocessor systems.

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

1.2.6.2.1 Signal Configuration

Figure 1-4 shows the MPC7410’s logical pin configuration. The signals are grouped by function.



Note: 266 total signal pins are shown (including analog V_{DD}s)
The data transaction index includes DBWO for 60x compatibility.
The bus monitor signals include ABB and DBB for 60x compatibility.

Figure 1-4. MPC7410 Microprocessor Signal Groups

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

1.2.6.2.2 Clocking

For functional operation, the MPC7410 uses a single clock input signal, SYSCLK, from which clocking is derived for the processor core, the L2 interface, and the MPX bus interface. Additionally, internal clock information is made available at the pins to support debug and development.

The MPC7410'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 are used to program the internal clock rate to a multiple of SYSCLK as defined in the MPC7410 hardware specification. The bus clock is maintained at the same frequency as SYSCLK. SYSCLK does not need to be a 50% duty-cycle signal.

The MPC7410 generates the clock for the external L2 synchronous data RAMs. The clock frequency for the RAMs is divided down from (and phase-locked to) the core clock frequency of the MPC7410. The core-to-L2 frequency divisor for the L2 PLL is selected through L2CR[L2CLK].

1.3 Implementation

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

- 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.
- 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.
- 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 MPC7410 supports all three levels of the architecture described above. For more information about the PowerPC architecture, see *The Programming Environments.* Specific MPC7410 features are listed in Section 1.2, "General Features."

This section describes the PowerPC ISA in general, and specific details about the MPC7410 implementation of this architecture. The structure of this section follows the organization of the user's manual; each subsection provides an overview of each 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 MPC7410. The information in this section is described more fully in Chapter 2, "Programming Model."

  Instruction set and addressing modes—Section 1.3.2, "Instruction Set," describes the instruction set and addressing modes for the PowerPC operating environment architecture, and defines and describes the PowerPC instructions implemented in the MPC7410. 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 MPC7410 cache implementation. The information in this section is described more fully in Chapter 3, "L1 and L2 Cache Operation."
- Exception model—Section 1.3.4, "Exception Model," describes the exception model of the operating environment architecture and the differences in the MPC7410 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 MPC7410's implementation of the 32-bit 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 MPC7410. The information in this section is described more fully in Chapter 6, "Instruction Timing."
- Power management—Section 1.3.8, "Power Management," describes how the power management can be used to reduce power consumption when the processor, or portions of it, are idle. The information in this section is described more fully in Chapter 10, "Power Management."
- Thermal management (MPC7400 only) —Section 1.3.9, "Thermal Management—MPC7400 only," describes the thermal management unit (TAU) of the MPC7400. Note that the MPC7410 does not support the thermal management facility. The TAU and its associated registers (THRM1–THRM3) and exception can be used to manage system activity in a way that prevents exceeding system and junction temperature thresholds. This is particularly useful in high-performance portable systems, which cannot use the same cooling mechanisms (such as fans) that control overheating in desktop systems. The information in this section is described more fully in Chapter 10, "Power Management."

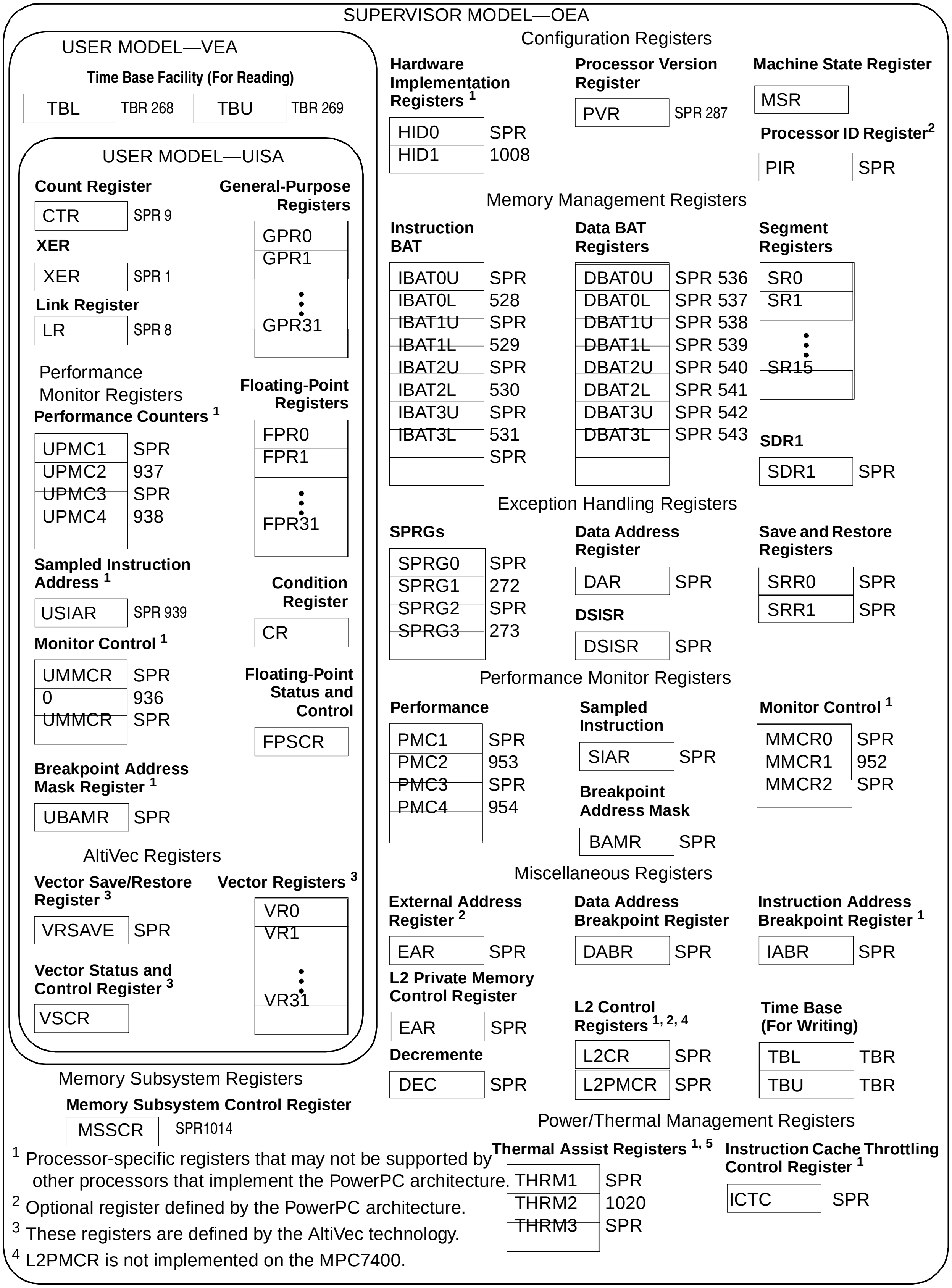
1.3.1 PowerPC Registers and Programming Model

The PowerPC ISA 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 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, special-purpose registers (SPRs), and several miscellaneous registers. The AltiVec extensions augment the programming model with 32 VRs, one status and control register, and one save and restore register. Each processor also has its own unique set of implementation-specific registers to support functionality that may not be defined by the 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 1-5 shows all the MPC7410 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.”



[1] Processor-specific registers that may not be supported by other processors that implement the PowerPC architecture.

[2] Optional register defined by the PowerPC architecture.

[3] These registers are defined by the AltiVec technology.

[4] L2PMCR is not implemented on the MPC7400.

Figure 1-5. MPC7410 Microprocessor Programming Model—Registers

The following tables summarize the PowerPC registers implemented in the MPC7410; Table 1-1 describes registers (excluding SPRs) defined by the architecture.

Table 1-1. Architecture-Defined Registers on the MPC7410 (Excluding SPRs)

Register	Level	Function
CR	User	The condition register (CR) consists of eight four-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.
FPRs	User	The 32 floating-point registers (FPRs) serve as the data source or destination for floating-point instructions. These 64-bit registers can hold either single- or double-precision floating-point values.
FPSCR	User	The floating-point status and control register (FPSCR) contains the floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE Std. 754.
GPRs	User	The 32 GPRs serve as the data source or destination for integer instructions.
MSR	Supervisor	The machine state register (MSR) defines the processor state. Its contents are saved when an exception is taken and restored when exception handling completes. The MPC7410 implements MSR[POW], (defined by the architecture as optional), which is used to enable the power management feature. The MPC7410-specific MSR[PM] bit is used to mark a process for the performance monitor.
SR0–SR15	Supervisor	The sixteen 32-bit segment registers (SRs) define the 4-Gbyte space as sixteen 256-Mbyte segments. The MPC7410 implements segment registers as two arrays—a main array for data accesses and a shadow array for instruction accesses; see Figure 1-1. Loading a segment entry with the Move to Segment Register (**mtsr**) instruction loads both arrays. The **mfsr** instruction reads the master register, shown as part of the data MMU in Figure 1-1.

The OEA defines numerous special-purpose registers that serve a variety of functions, such as providing controls, indicating status, configuring the processor, and performing special operations. During normal execution, a program can access the registers, shown in Figure 1-5, depending on the program's access privilege (supervisor or user, determined by the privilege-level (PR) bit in the MSR). GPRs and FPRs are accessed through operands that are part of the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions) or implicit, as the part of the execution of an instruction. Some registers can be accessed both explicitly and implicitly.

In the MPC7410, all SPRs are 32 bits wide. Table 1-2 describes the architecture-defined SPRs implemented by the MPC7410. *The Programming Environments Manual* describes these registers in detail, including bit descriptions. Section 2.1.1, "Register Set Overview," describes how these registers are implemented in the MPC7410. In particular, this section describes which features the architecture defines as optional are implemented on the MPC7410.

Table 1-2. Architecture-Defined SPRs Implemented by the MPC7410

Register	Level	Function
LR	User	The link register (LR) can be used to provide the branch target address and to hold the return address after branch and link instructions.
BATs	Supervisor	The architecture defines 16 block address translation (BAT) registers, which operate in pairs. There are four pairs of data BATs (DBATs) and four pairs of instruction BATs (IBATs). BATs are used to define and configure blocks of memory.
CTR	User	The count register (CTR) is decremented and tested by branch-and-count instructions.
DABR	Supervisor	The optional data address breakpoint register (DABR) supports the data address breakpoint facility.
DAR	User	The data address register (DAR) holds the address of an access after an alignment or DSI exception.
DEC	Supervisor	The decrementer register (DEC) is a 32-bit decrementing counter that provides a way to schedule decrementer exceptions.
DSISR	User	The DSISR defines the cause of data access and alignment exceptions.
EAR	Supervisor	The external access register (EAR) controls access to the external access facility through the External Control In Word Indexed (**eciwx**) and External Control Out Word Indexed (**ecowx**) instructions.
PIR	Supervisor	The processor ID register (PIR) is used to differentiate between processors in a multiprocessor system.
PVR	Supervisor	The processor version register (PVR) is a read-only register that identifies the processor.
SDR1	Supervisor	SDR1 specifies the page table format used in virtual-to-physical page address translation.
SRR0	Supervisor	The machine status save/restore register 0 (SRR0) saves the address used for restarting an interrupted program when a Return from Interrupt (**rfi**) instruction executes.
SRR1	Supervisor	The machine status save/restore register 1 (SRR1) is used to save machine status on exceptions and to restore machine status when an **rfi** instruction is executed.
SPRG0–SPRG3	Supervisor	SPRG0–SPRG3 are provided for operating system use.
TB	User: read Supervisor: read/write	The time base register (TB) is a 64-bit register that maintains the time of day and operates interval timers. The TB consists of two 32-bit fields—time base upper (TBU) and time base lower (TBL).
XER	User	The XER contains the summary overflow bit, integer carry bit, overflow bit, and a field specifying the number of bytes to be transferred by a Load String Word Indexed (**lswx**) or Store String Word Indexed (**stswx**) instruction.

Table 1-3 describes the registers defined by the AltiVec technology.

Table 1-3. AltiVec-Specific Registers

Register	Level	Function
VRs	User	The 32 vector registers (VRs) serve as the data source or destination for AltiVec instructions.
VSCR	User	The 32-bit vector status and control register (VSCR). A 32-bit vector register that is read and written in a manner similar to the FPSCR.
VRSAVE	User	The 32-bit vector save (VRSAVE) register is defined by the AltiVec technology to assist application and operating system software in saving and restoring the architectural state across process context-switched events.

Table 1-4 describes the supervisor-level SPRs in the MPC7410 that are not defined by the architecture. Section 2.1.2, “Register Set Summary,” gives detailed descriptions of these registers, including bit descriptions.

Table 1-4. MPC7410-Specific Registers

Register	Level	Function
BAMR	Supervisor	Breakpoint address mask register is used in conjunction with the events that monitor IABR and DABR hits.
HID0	Supervisor	The hardware implementation-dependent register 0 (HID0) provides checkstop enables and other functions.
HID1	Supervisor	The hardware implementation-dependent register 1 (HID1) allows software to read the configuration of the PLL configuration signals.
IABR	Supervisor	The instruction address breakpoint register (IABR) supports instruction address breakpoint exceptions. It can hold an address to compare with instruction addresses in the IQ. An address match causes an instruction address breakpoint exception.
ICTC	Supervisor	The instruction cache-throttling control register (ICTC) has bits for controlling the interval at which instructions are fetched into the instruction queue in the instruction unit. This helps control the MPC7410’s overall junction temperature.
L2CR	Supervisor	The L2 cache control register (L2CR) is used to configure and operate the L2 cache. It has bits for enabling parity checking, setting the L2-to-processor clock ratio, and identifying the type of RAM used for the L2 cache implementation.
L2PMCR	Supervisor	MPC7410 only. The L2 private memory control register (L2PMCR) is used to configure the private memory function of the L2 interface. This register is not implemented on the MPC7400.
MMCR0–MMCR2	Supervisor	The monitor mode control registers (MMCR0–MMCR1) are used to enable various performance monitoring interrupt functions. UMMCR0–UMMCR1 provide user-level read access to MMCR0–MMCR1.
MSSCR0	Supervisor	The memory subsystem control register is used to configure and operate the memory subsystem.
PMC1–PMC4	Supervisor	The performance monitor counter registers (PMC1–PMC4) are used to count specified events. UPMC1–UPMC4 provide user-level read access to these registers.
SIAR	Supervisor	The sampled instruction address register (SIAR) holds the EA of an instruction executing at or around the time the processor signals the performance monitor interrupt condition. The USIAR register provides user-level read access to the SIAR.

Table 1-4. MPC7410-Specific Registers (continued)

Register	Level	Function
THRM1, THRM2	Supervisor	MPC7400 only. THRM1 and THRM2 provide a way to compare the junction temperature against two user-provided thresholds. The thermal assist unit (TAU) can be operated so that the thermal sensor output is compared to only one threshold, selected in THRM1 or THRM2.
THRM3	Supervisor	MPC7400 only. THRM3 is used to enable the TAU and to control the output sample time.
UBAMR	User	The user breakpoint address mask register (UBAMR) provides user-level read access to BAMR.
UMMCR0–UMMCR2	User	The user monitor mode control registers (UMMCR0–UMMCR1) provide user-level read access to MMCR0–MMCR2.
UPMC1–UPMC4	User	The user performance monitor counter registers (UPMC1–UPMC4) provide user-level read access to PMC1–PMC4.
USIAR	User	The user sampled instruction address register (USIAR) provides user-level read access to the SIAR register.

1.3.2 Instruction Set

All instructions tdefined by the architecture 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/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 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 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.
- Processor control instructions—These instructions are used to read and write from the vector status and control register (VSCR).
- Memory control instructions—These instructions are used for managing the caches (user level and supervisor level).
- 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 Instruction Set

The MPC7410 instruction set is defined as follows:

- The MPC7410 provides hardware support for all 32-bit PowerPC instructions.
- The MPC7410 implements the following instructions optional to the 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**)

1.3.3 On-Chip Cache Implementation

The following subsections describe the architecture's treatment of cache in general and the MPC7410-specific implementation, respectively. A detailed description of the MPC7410 cache implementation is provided in Chapter 3, "L1 and L2 Cache Operation."

1.3.3.1 Cache Model

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

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

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

The 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 Cache Implementation

The MPC7410 cache implementation is described in Section 1.2.4, "On-Chip Instruction and Data Caches," and Section 1.2.5, "L2 Cache Implementation." The BPU also contains a 64-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 exception model defined by the architecture as well as the implementation specific to the MPC7410. A detailed description of the MPC7410 exception model is provided in Chapter 4, "Exceptions."

1.3.4.1 PowerPC Exception Model

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

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

The architecture requires that exceptions be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are handled in order. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that are undispatched, are required to complete before the exception is taken, and any exceptions those instructions cause must also be handled first. Likewise, asynchronous, precise exceptions are recognized when they occur, but are not handled until the instructions currently in the completion queue successfully retire or generate an exception, and the completion queue is emptied.

Unless a catastrophic condition causes a system reset or machine check exception, only one exception is handled at a time. For example, if one instruction encounters multiple exception conditions, those conditions are handled sequentially. After the exception handler handles an exception, the instruction processing continues until the next exception condition is encountered. Recognizing and handling exception conditions sequentially guarantees that exceptions are recoverable.

When an exception is taken, information about the processor state before the exception was taken is saved in SRR0 and SRR1. Exception handlers should save the information stored in SRR0 and SRR1 early to prevent the program state from being lost due to a system reset or machine check exception, or due to an instruction-caused exception in the exception handler. The contents of SRR0 and SRR1 should also be saved before enabling external interrupts.

The architecture supports four types of exceptions:

- Synchronous, precise—These are caused by instructions. All instruction-caused exceptions are handled precisely; that is, the machine state at the time the exception occurs is known and can be completely restored. This means that (excluding the trap and system call exceptions) the address of the faulting instruction is provided to the exception handler and that neither the faulting instruction nor subsequent instructions in the code stream will complete execution before the exception is taken. Once the exception is processed, execution resumes at the address of the faulting instruction (or at an alternate address provided by the exception handler). When an exception is taken due to a trap or system call instruction, execution resumes at an address provided by the handler.
- Synchronous, imprecise—The architecture defines two imprecise floating-point exception modes: recoverable and nonrecoverable. Even though the MPC7410 provides a means to enable the imprecise modes, it implements these modes identically to the precise mode (that is, enabled floating-point exceptions are always precise).

- Asynchronous, maskable—The architecture defines external and decrementer interrupts as maskable, asynchronous exceptions. When these exceptions occur, their handling is postponed until the next instruction, and any exceptions associated with that instruction, completes execution. If no instructions are in the execution units, the exception is taken immediately upon determination of the correct restart address (for loading SRR0). As shown in Table 1-5, the MPC7410 implements additional asynchronous, maskable exceptions.
- Asynchronous, nonmaskable—There are two nonmaskable asynchronous exceptions: system reset and the machine check exception. These exceptions may not be recoverable, or may provide a limited degree of recoverability. Exceptions report recoverability through the MSR[RI] bit.

1.3.4.2 MPC7410 Exception Implementation

Although exceptions have other characteristics, such as priority and recoverability, Table 1-5 describes categories of exceptions the MPC7410 handles uniquely. Table 1-5 includes no synchronous imprecise exceptions; although the PowerPC architecture supports imprecise handling of floating-point exceptions, the MPC7410 implements these exception modes precisely.

Table 1-5. Exception Classifications

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

Table 1-6 lists MPC7410 exceptions and conditions that cause them. Exceptions specific to the MPC7410 are indicated. Note that only three exceptions may result from execution of an AltiVec instruction:

- AltiVec unavailable exception. Taken if there is an attempt to execute any non-stream vector instruction with MSR[VA] = 0. After this exception is handled, execution resumes at offset 0x00F20. This exception does not occur for stream instructions (**dst**[**t**], **dstst**[**t**], or **dss**). Note that the contents of the VRSAVE register are not protected by this exception, which is consistent with the AltiVec specification.
- A DSI exception. Taken if a vector load or store operation encounters a page fault (does not find a valid PTE) or a protection violation. Also a DSI occurs if a vector load or store attempts to access T = 1 direct store space.
- AltiVec assist exception. Taken in some cases if a vector floating-point instruction detects denormalized data as an input or output in Java mode.

Table 1-6. Exceptions and Conditions

Exception Type	Vector Offset (hex)	Causing Conditions
Reserved	00000	—
System reset	00100	Assertion of either $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$ or at power-on reset

Table 1-6. Exceptions and Conditions (continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Machine check	00200	Assertion of $\overline{\text{TEA}}$ during a data bus transaction, assertion of $\overline{\text{MCP}}$, or an address, data, or L2 bus parity error. MSR[ME] must be set.
DSI	00300	As specified in the architecture. For TLB misses on load, store, or cache operations, a DSI exception occurs if a page fault occurs. The MPC7410 takes a DSI if a **lwarx** or **stwcx.** instruction is executed to an address marked write-through or if the data cache is enabled and locked.
ISI	00400	As defined by the PowerPC architecture.
External interrupt	00500	MSR[EE] = 1 and $\overline{\text{INT}}$ is asserted.
Alignment	00600	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. The operand of **dcbz** is in memory that is write-through-required or caching-inhibited or the cache is disabled
Program	00700	As defined by the architecture.
Floating-point unavailable	00800	As defined by the architecture.
Decrementer	00900	As defined by the architecture, when the most significant bit of the DEC register changes from 0 to 1 and MSR[EE] = 1.
Reserved	00A00–00BFF	—
System call	00C00	Execution of the System Call (**sc**) instruction.
Trace	00D00	MSR[SE] = 1 or a branch instruction completes and MSR[BE] = 1. Unlike the architecture definition, **isync** does not cause a trace exception on MPC7410.
Reserved	00E00	The MPC7410 does not generate an exception to this vector. Other processors may use this vector for floating-point assist exceptions.
Reserved	00E10–00EFF	—
Performance monitor[1]	00F00	The limit specified in a PMC register is reached and MMCR0[ENINT] = 1
AltiVec unavailable[1]	00F20	Occurs due to an attempt to execute any non-stream AltiVec instruction while MSR[VA] = 0. This exception is not taken for stream instructions (**dst**[**t**], **dstst**[**t**] or **dss**).
Instruction address breakpoint[1]	01300	IABR[0–29] matches EA[0–29] of the next instruction to complete, and IABR[BE] = 1.
System management interrupt[1]	01400	MSR[EE] = 1 and $\overline{\text{SMI}}$ is asserted.
Reserved	01500–015FF	—
AltiVec assist[1]	01600	Supports denormalization detection in Java mode as defined by the AltiVec specification.

Table 1-6. Exceptions and Conditions (continued)

Exception Type	Vector Offset (hex)	Causing Conditions
Thermal management exception[2]	01700	MPC7400 only. Thermal management is enabled, the junction temperature exceeds the threshold specified in THRM1 or THRM2, and MSR[EE] = 1.
Reserved	01800–02FFF	—

[1] MPC7410-/MPC7400-specific

[2] MPC7400-specific

1.3.5 Memory Management

The following subsections describe the memory management features of the architecture and the MPC7410 implementation, respectively.

1.3.5.1 PowerPC Memory Management Model

The primary functions of the MMU are to translate logical (effective) addresses to physical addresses for memory accesses and to provide access protection on blocks and pages of memory. There are two types of accesses generated by the MPC7410 that require address translation—instruction accesses, and data accesses to memory generated by load, store, and cache control instructions.

The architecture defines different resources for 32- and 64-bit processors; the MPC7410 implements the 32-bit memory management model. The memory management model provides 4 Gbytes of logical address space accessible to supervisor and user programs with a 4-Kbyte page size and 256-Mbyte segment size. In addition, it defines an interim 52-bit virtual address and hashed page tables for generating 32-bit physical addresses.

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

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

The hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of 2, and its starting address is a multiple of its size. The page table contains a number of page table entry groups (PTEGs). A PTEG contains eight page table entries (PTEs) of eight bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

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

1.3.5.2 Memory Management Implementation

The MPC7410 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 MPC7410 MMU is described in Section 1.2.3, "Memory Management Units (MMUs)."

The R (referenced) bit is updated in the PTE in memory (if necessary) during a table search due to a TLB miss. Updates to the C (changed) bit are treated like TLB misses. A complete table search is performed and the entire TLB entry is rewritten to update the C bit.

1.3.6 Instruction Timing

The MPC7410 is a pipelined, superscalar processor. A pipelined processor is one in which instruction processing is divided into discrete stages, allowing work to be done on different instructions in each stage. For example, after an instruction completes one stage, it can pass on to the next stage leaving the previous stage available to the subsequent instruction. This improves overall instruction throughput.

A superscalar processor is one that issues multiple independent instructions into separate execution units, allowing instructions to execute in parallel. The MPC7410 has eight independent execution units, two for integer instructions, and one each for floating-point, branch, load/store, system register, vector permute, and vector arithmetic logic unit instructions. Having separate GPRs, FPRs, and VRs allows integer, floating-point, and vector calculations, and load and store operations to occur simultaneously without interference. Additionally, rename buffers are provided to allow operations to post execution results for use by subsequent instructions without committing them to the architected FPRs, GPRs, and VRs.

As shown in Figure 1-6, the common pipeline of the MPC7410 has four stages through which all instructions must pass—fetch, decode/dispatch, execute, and complete/write back. Some instructions occupy multiple stages simultaneously and some individual execution units have additional stages. For example, the floating-point pipeline consists of three stages through which all floating-point instructions must pass.

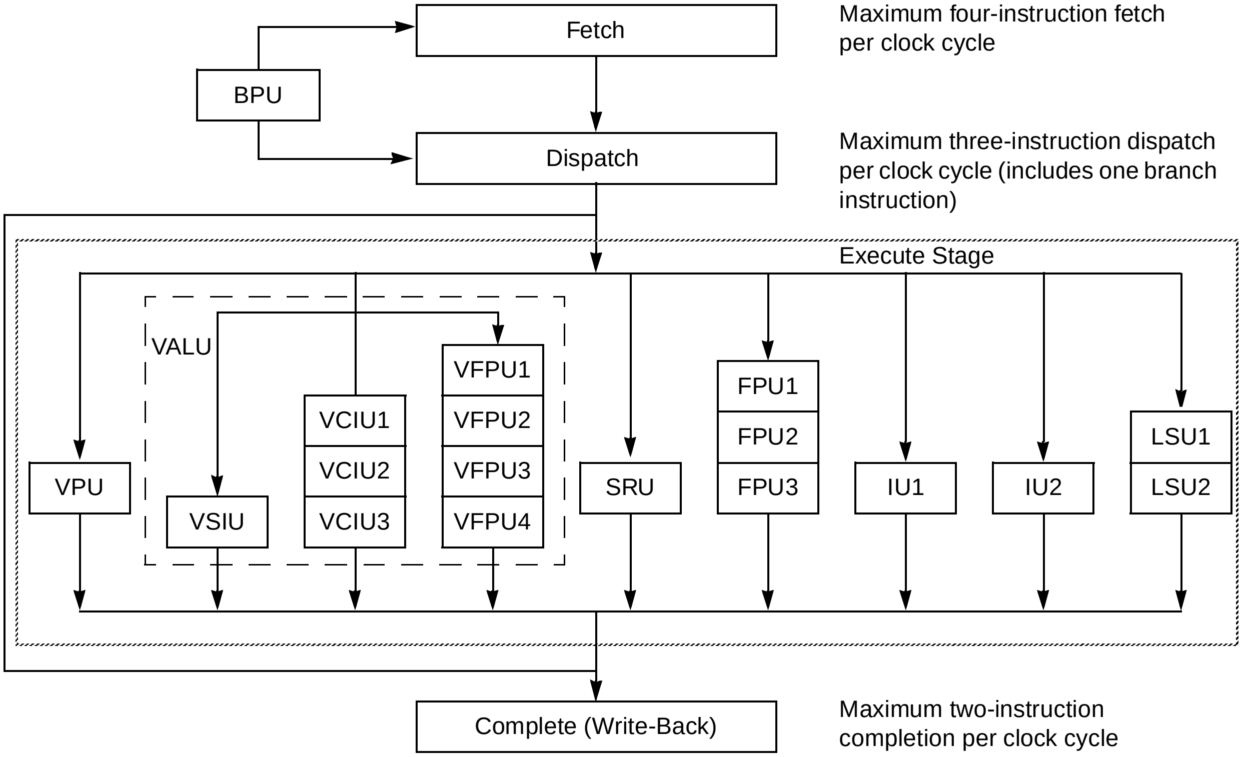

Fetch
Maximum four-instruction fetch per clock cycle
BPU
Dispatch
Maximum three-instruction dispatch per clock cycle (includes one branch instruction)
Execute Stage
VALU
VPU
VSIU
VCIU1
VCIU2
VCIU3
VFPU1
VFPU2
VFPU3
VFPU4
SRU
FPU1
FPU2
FPU3
IU1
IU2
LSU1
LSU2
Complete (Write-Back)
Maximum two-instruction completion per clock cycle

Figure 1-6. Pipeline Diagram

Note that Figure 1-6 does not show features, such as reservation stations and rename buffers that reduce stalls and improve instruction throughput.

The instruction pipeline in the MPC7410 has four major pipeline stages as described below. Because the architecture can be applied to such a wide variety of implementations, instruction timing varies among processors, and the following pipeline description is specific to the MPC7410 and MPC7400.

- The fetch pipeline stage primarily involves retrieving instructions from the memory system and determining the location of the next instruction fetch. The BPU decodes branches during the fetch stage and removes those that do not update CTR or LR from the instruction stream.
- The dispatch stage is responsible for decoding the instructions supplied by the instruction fetch stage and determining which instructions can be dispatched in the current cycle. A rename ID is given to instructions with a target destination. If source operands for the instruction are available, they are read from the appropriate register file or rename register to the execute pipeline stage. If a source operand is not available, dispatch provides a tag that indicates which rename register will supply the operand when it becomes available. At the end of the dispatch stage, the dispatched instructions and their operands are latched by the appropriate execution unit.
- Instructions executed by the IUs, FPU, SRU, LSU, VPU, and VALU are dispatched from the bottom two positions in the instruction queue. In a single clock cycle, a maximum of two instructions can be dispatched to these execution units in any combination. When an instruction is

dispatched, it is assigned a position in the eight-entry completion queue. A branch instruction can be issued on the same clock cycle for a maximum three-instruction dispatch.

- During the execute pipeline stage, each execution unit that has an executable instruction executes the selected instruction (perhaps over multiple cycles), writes the instruction's result into the appropriate rename register, and notifies the completion stage that the instruction has finished execution. In the case of an internal exception, the execution unit reports the exception to the completion pipeline stage and (except for the FPU) discontinues instruction execution until the exception is handled. The exception is not signaled until that instruction is the next to be completed.

  Execution of most floating-point instructions is pipelined within the FPU allowing up to three instructions to be executing in the FPU concurrently. The FPU stages are multiply, add, and round-convert. Execution of most load/store instructions is also pipelined. The load/store unit has two pipeline stages. The first stage is for effective address calculation and MMU translation and the second stage is for accessing the data in the cache.
- The complete pipeline stage maintains the correct architectural machine state and transfers execution results from the rename registers to the GPRs and FPRs (and CTR and LR, for some instructions) as instructions are retired. As with dispatching instructions from the instruction queue, instructions are retired from the two bottom positions in the completion queue. If completion logic detects an instruction causing an exception, all following instructions are cancelled, their execution results in rename registers are discarded, and instructions are fetched from the appropriate exception vector.

1.3.7 AltiVec Implementation

The MPC7410 implements the AltiVec registers and instruction set as they are described by the *AltiVec Technology Programming Environments Manual*. AltiVec technology features are briefly described in the following sections:

- AltiVec registers are described in Table 1-3.
- AltiVec instructions are described in Section 1.3.2.2, “AltiVec Instruction Set.”
- Execution units for AltiVec instructions are described in Section 1.2.2.4.1, “AltiVec Vector Permute Unit (VPU),” and Section 1.2.2.4.2, “AltiVec Vector Arithmetic Logic Unit (VALU).”

The AltiVec implementation is described fully in Chapter 7, “AltiVec Technology Implementation.”

1.3.8 Power Management

The MPC7410 provides four power modes, selectable by setting the appropriate control bits in the MSR and HID0 registers. The four power modes are as follows:

- Full-power—This is the default power state of the MPC7410. The MPC7410 is fully powered and the internal functional units are operating at the full processor clock speed. If the dynamic power management mode is enabled, functional units that are idle will automatically enter a low-power state without affecting performance, software execution, or external hardware.
- Doze—All the functional units of the MPC7410 are disabled except for the time base/decrementer registers and the bus snooping logic. The MPC7400-specific thermal assist unit also remains active

in doze mode. When the processor is in doze mode, an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or machine check brings the MPC7410 into the full-power state. The MPC7410 in doze mode maintains the PLL in a fully powered state and locked to the system external clock input (SYSCLK) so a transition to the full-power state takes only a few processor clock cycles.

- Nap—The nap mode further reduces power consumption by disabling bus snooping, leaving only the decrementer/time base registers, the PLL, and the DLL (for L2 RAM clocks) in a powered state. The MPC7400-specific thermal assist unit also remains active in nap mode. The MPC7410 returns to the full-power state upon receipt of an external asynchronous interrupt, a system management interrupt, a decrementer exception, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$). A return to full-power state from a nap state takes only a few processor clock cycles. When the processor is in nap mode, if $\overline{\text{QACK}}$ is negated, the processor is put in doze mode to support snooping.
- Sleep—Sleep mode minimizes power consumption by disabling all internal functional units, after which external system logic may disable the PLL and SYSCLK. Returning the MPC7410 to the full-power state requires the enabling of the PLL and SYSCLK, followed by the assertion of an external asynchronous interrupt, a system management interrupt, a hard or soft reset, or a machine check input ($\overline{\text{MCP}}$) signal after the time required to relock the PLL.

Chapter 10, "Power Management," provides information about power saving modes for the MPC7410.

1.3.9 Thermal Management—MPC7400 only

The MPC7400-specific thermal assist unit (TAU) provides a way to control heat dissipation. This ability is particularly useful in portable computers, which, due to power consumption and size limitations, cannot use desktop cooling solutions such as fans. Therefore, better heat sink designs coupled with intelligent thermal management is of great importance for high performance portable systems. The thermal assist unit (TAU), three supervisor-level registers, and a thermal management exception to allow for software control of thermal management. Note that the MPC7410 does not support the thermal management facility.

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

The thermal management facility also ensures that the processor's junction temperature does not exceed the operating specification. To avoid the inaccuracies that arise from measuring junction temperature with an external thermal sensor, the MPC7400's on-chip thermal sensor and logic tightly couples the thermal management implementation.

The TAU consists of a thermal sensor, digital-to-analog convertor, comparator, control logic, and the dedicated SPRs described in Section 1.3.1, "PowerPC Registers and Programming Model." The TAU does the following:

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

The TAU is controlled through the privileged **mtspr**/**mfspr** instructions to the three SPRs provided for configuring and controlling the sensor control logic, which function as follows:

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

Instruction cache throttling provides control of the MPC7400's overall junction temperature by determining the interval at which instructions are fetched. This feature is accessed through the ICTC register.

Section 10.3, "Thermal Assist Unit (TAU)—MPC7400 Only," provides information about thermal management modes for the MPC7400.

1.3.10 Performance Monitor

The MPC7410 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 or MMCR1, 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-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 MPC7410.

1.4 Differences between the MPC7410 and the MPC7400

The MPC7410 is a derivative of the MPC7400 microprocessor design. Table 1-7 summarizes the differences between the two microprocessors.

Table 1-7. Differences between the MPC7410 and the MPC7400

Feature	Difference
Private memory	The MPC7410 supports using the L2 SRAMs as direct-mapped private memory. The private memory feature on the MPC7410 is configured by a new supervisor-level, special-purpose register, the L2 private memory control register (L2PMCR). The MPC7400 does not support private memory. As such, the MPC7400 does not implement the L2PMCR.
L2 data bus width	The MPC7410 supports a 32- or 64-bit L2 data bus. The MPC7400 supports only a 64-bit L2 data bus.
L2 address bus width	The MPC7410 adds an L2 address signal, L2ADDR[18], to support up to 2 Mbyte of L2 cache with a 32-bit data bus.
Thermal Assist Unit	The MPC7400 features a thermal assist unit; the MPC7410 does not support the thermal assist unit. See Section 1.3.9, "Thermal Management—MPC7400 only," for more information.
Processor version register (PVR)	The PVR for the MPC7410 is 0x800C_1*nnn*. The PVR for the MPC7400 is 0x000C_0*nnn*.
Core and I/O voltages	The electrical characteristics of the MPC7410 are different from those of the MPC7400. See the corresponding hardware specifications for each device.
Operation frequency and core/clock ratios	The clock AC specifications and PLL configuration of the MPC7410 are different from those of the MPC7400. See the corresponding hardware specifications for each device.

1.5 Differences between the MPC7410 and the MPC750

The design philosophy on the MPC7410 (and the MPC7400) is to change from the MPC750 base only where required to gain compelling multimedia and multiprocessor performance. The MPC7410's core is essentially the same as the MPC750's, except that whereas the MPC750 has a 6-entry completion queue and has slower performance on some floating-point double-precision operations, the MPC7410 has an 8-entry completion queue and a full double-precision FPU. The MPC7410 also adds the AltiVec instruction set, has a new memory subsystem, and can interface to the improved MPX bus. Differences are summarized in Table 1-8.

Table 1-8. Differences between the MPC7410 and the MPC750

Feature	Difference
Core	
Sequencing	The MPC750 has a 6-entry IQ and a 6-entry CQ. For each clock, it can fetch four instructions, dispatch two instructions, fold one branch, and complete two instructions. The MPC7410 is identical, except for an eight-entry CQ, as shown in Figure 1-1. The extra CQ entries reduce the opportunity for dispatch bottlenecks to the MPC7410's additional execution units.

Table 1-8. Differences between the MPC7410 and the MPC750 (continued)

<table>
<tr><th>Feature</th><th colspan="3">Difference</th></tr>
<tr><td>Load/Store Ordering</td><td colspan="3">On the MPC750, load and store operations are assumed to be weakly ordered. That is, the load/store unit (LSU) can perform load operations that occur later in the program ahead of store operation. However, strongly ordered load and store operations can be enforced through setting the caching-inhibited (I) memory/cache access attribute.
On the MPC7410, load and store operations are also assumed to be weakly ordered, and load operations can bypass store operations. However, unlike the MPC750 and other PowerPC microprocessors, the MPC7410 does not enforce load/store ordering when the access is caching-inhibited or write-through guarded. See Section 3.4.4.2, “Sequential Consistency of Memory Accesses,” for more information.</td></tr>
<tr><td rowspan="4">FPU</td><td colspan="3">On the MPC750, single-precision operations involving multiplication have a 3-cycle latency, while their double-precision equivalents take an additional cycle. Because the MPC7410 has a full double-precision FPU, double- and single-precision multiplies have the same latency: 3 cycles. Floating-point divides have the same latency for both designs (17 cycles for single-precision, 31 for double-precision).</td></tr>
<tr><td rowspan="2">MPC750</td><td>Double-precision floating-point multiply</td><td>4 cycles</td></tr>
<tr><td>All other floating-point add and multiply</td><td>3 cycles</td></tr>
<tr><td>MPC7410</td><td>All floating-point add and multiply</td><td>3 cycles</td></tr>
<tr><td>AltiVec technology</td><td colspan="3">The MPC7410 implements all instructions defined by the AltiVec specification. Two dispatchable AltiVec functional units were added, a vector permute unit (VPU) and a vector ALU unit (VALU). The VALU comprises a simple integer unit, a complex integer unit, and a floating-point unit. As shown in Figure 1-1, the MPC7410 also adds 32 128-bit vector registers (VRs) and 6 VR rename registers.
The VPU handles permute and shift operations and the VALU handles calculations. The LSU handles AltiVec load and store operations. To support AltiVec operations, all memory subsystem data buses are 128 bits wide (as opposed to 64 bits in the MPC750). Queues have been added and queue sizes have been increased to sustain heavy AltiVec technology usage.
The AltiVec technology is designed to improve the performance of vector-intensive code in applications such as multimedia and digital signal processing. AltiVec-targeted code can accelerate 2D and 3D graphics functions 3–5 times, especially core functions in 3D engines and game-related 2D functions.</td></tr>
<tr><td colspan="4"><b>Memory Subsystem</b></td></tr>
<tr><td colspan="4">The MPC7410 has a new memory subsystem designed to support AltiVec technology loads, the new MPX bus protocol, and 5-state multiprocessing capabilities. Queues and queue sizes are designed to support more efficient data flow. For example, the MPC750 has a three-entry LSU store queue, while the MPC7410 has a six-entry LSU store queue.
The MPC7410 adds an eight-entry reload buffer, where L1 data cache misses can wait for their data to be loaded. This enables load miss folding and store miss merging.</td></tr>
<tr><td>Load miss folding</td><td colspan="3">In the MPC750, if a second load misses to the same cache block, the second load must wait for the critical word of the first load before it can access its data, and subsequent accesses are also stalled. In the MPC7410, the first load or store causes an entry to be allocated in the reload buffer. A subsequent load to the same cache block is placed aside in the load fold queue (LFQ), and it can return its data immediately when available. Also, subsequent accesses to the cache are not blocked and can be processed.
For example, on the MPC750 if a load or store (access A) misses in the data cache, a subsequent load (access B) to the same cache block must wait until the critical word for A is retired. Because of this, any subsequent loads or stores after access B also cannot access the data cache until the reload for access A completes.
On the other hand, with the MPC7410 if a load or store access A misses in the data cache, up to four subsequent misses to the same cache block can be folded into the LFQ, and subsequent instructions can access the data cache. Loads are blocked only when the reload table or the LFQ are full.</td></tr>
</table>

Table 1-8. Differences between the MPC7410 and the MPC750 (continued)

Feature	Difference	
Store miss merging	In the MPC750, if a second store misses to the same cache block, it must wait for the critical word of the first store before it can write its data. The MPC7410 can merge several stores to the same cache block into the same entry in its reload buffer. If enough stores merge to write all 32 bytes of the cache block (usually via two back-to-back AltiVec store misses), then no data needs to be loaded from the bus and an address-only transaction (KILL) is broadcast instead.	
Cache		
Allocate on reload	Both designs have the same L1 cache size, but differ in their block allocation policy. The MPC750 has an allocate-on-miss policy, while the MPC7410 has an allocate-on-reload policy, which allows better cache allocation and replacement and more efficient use of data bus bandwidth. If access A misses in the cache, the MPC750 immediately identifies the victim block (call it X) if there is one and allocates its space for the new data (call it Y) to be loaded. If a subsequent access (access B) needs this victim block, even if access B occurs before Y has been loaded, then it will miss because as soon as X is victimized it is no longer valid. After Y has loaded (and, if X is modified, after X has been cast out), X must be reloaded, and B must wait until its data is valid again. The MPC7410, on the other hand, delays allocation/victimization until the block reload occurs. In the example above, while Y is being loaded, B can hit block X, and a different block is victimized. This allows more efficient use of the cache and can reduce thrashing. On the MPC7410, allocation occurs in parallel with reload which uses the cache more efficiently.	
	MPC750	MPC7410
	1-cycle load arbitration	1-cycle load arbitration
	1-cycle allocate	4-beat reload
	4-cycle victimization (if castout needed)	
	4-beat reload (64 bits/beat)	
	Total = 6 or 10 cycles	Total = 5 cycles
Outstanding misses	The MPC750 allows one outstanding data cache miss and one outstanding instruction cache miss (accessing the L2 or the bus) at any time. The MPC7410 allows one instruction cache miss and up to eight data side misses. Note that the L2 can queue up to four hits but with a fast L2 (1:1 mode) it is impossible to fill this queue with data cache misses. The L2 miss queue can queue four transactions waiting to access the processor address bus.	
Miss under miss	While processing a miss, the MPC750's data cache allows subsequent loads and stores to hit in the data cache (hit under miss), but it blocks on the next miss until the first miss finishes reloading. The MPC7410 allows subsequent accesses that miss in the data cache to propagate to the L2 and beyond (miss under miss).	

Table 1-8. Differences between the MPC7410 and the MPC750 (continued)

<table>
<tr><th>Feature</th><th>Difference</th></tr>
<tr><td>L2 cache</td><td>The MPC7410 has twice as many on-chip L2 tags per way (8192) than the MPC750 and can support twice the L2 cache size (up to 2 Mbyte). The sectoring configuration differs as follows:
<table>
<tr><th colspan="2">MPC750</th><th colspan="2">MPC7410</th></tr>
<tr><td></td><td></td><td>2 Mbyte</td><td>4 sectors/tag</td></tr>
<tr><td>1 Mbyte</td><td>4 sectors/tag</td><td>1 Mbyte</td><td>2 sectors/tag</td></tr>
<tr><td>512 Kbyte</td><td>2</td><td>512 Kbyte</td><td>1 sector/tag</td></tr>
</table>
Assigning fewer sectors per tag uses the cache more efficiently.
The MPC7410 and MPC750 also have different cache reload policies. On the MPC750, an L1 cache miss that also misses in the L2 causes a reload from the bus to both L1 and L2. On the MPC7410, misses to the L1 instruction cache behave the same way, but misses to the L1 data cache cause data to be reloaded into the L1 only. Thus, with respect to the L1 data cache, the L2 holds only blocks that are cast out; it acts as a giant victim cache for the L1 data cache. This improves performance because the data is duplicated in the L1 data cache and L2 less often.</td></tr>
<tr><td>L2 data bus width</td><td>The MPC7410 supports a 32- or 64-bit L2 data bus.
The MPC7400 supports only a 64-bit L2 data bus.</td></tr>
<tr><td>L2 address bus width</td><td>The MPC7410 L2 address bus has two additional bits:
MPC7410 L2ADDR[18:0]
MPC750 L2ADDR[16:0]</td></tr>
<tr><td>Private memory</td><td>Although not supported on the MPC7400, the MPC7410’s L2 interface supports using the SRAM area as a direct-mapped, private memory space. This feature is supported on the MPC755, but is not supported on the MPC750. 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 L2 cache nor are they visible to the external system bus.</td></tr>
<tr><td>60x bus/ MPX bus</td><td>The MPC7410 supports the 60x bus used by the MPC750, but it also supports a new bus (MPX bus). It implements a 5-state cache-coherency protocol (MERSI) and the MESI and MEI subsets. This provides better hardware support of multiprocessing.
For example, the MPX bus supports data intervention. On the 60x bus, if one processor performs a read of data that is marked modified in another processor’s cache, the transaction is retried and the data is pushed to memory, after which the transaction is restarted. The MPX bus allows data to be forwarded directly to the requesting processor from the processor that has it cached. (The MPC7410 also supports intervention for data marked exclusive and shared.)
The MPC7410 supports up to seven simultaneous transactions on the 60x or MPX bus interface (one in progress and six pending); the MPC750 supports only two.</td></tr>
</table>

Chapter 2
Programming Model

This chapter describes the MPC7410 programming model, emphasizing those features specific to the MPC7410 processor and summarizing those that are common to processors built on the PowerPC ISA. It consists of three major sections, which describe the following:

- Registers implemented in the MPC7410
- Operand conventions
- The MPC7410 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.”

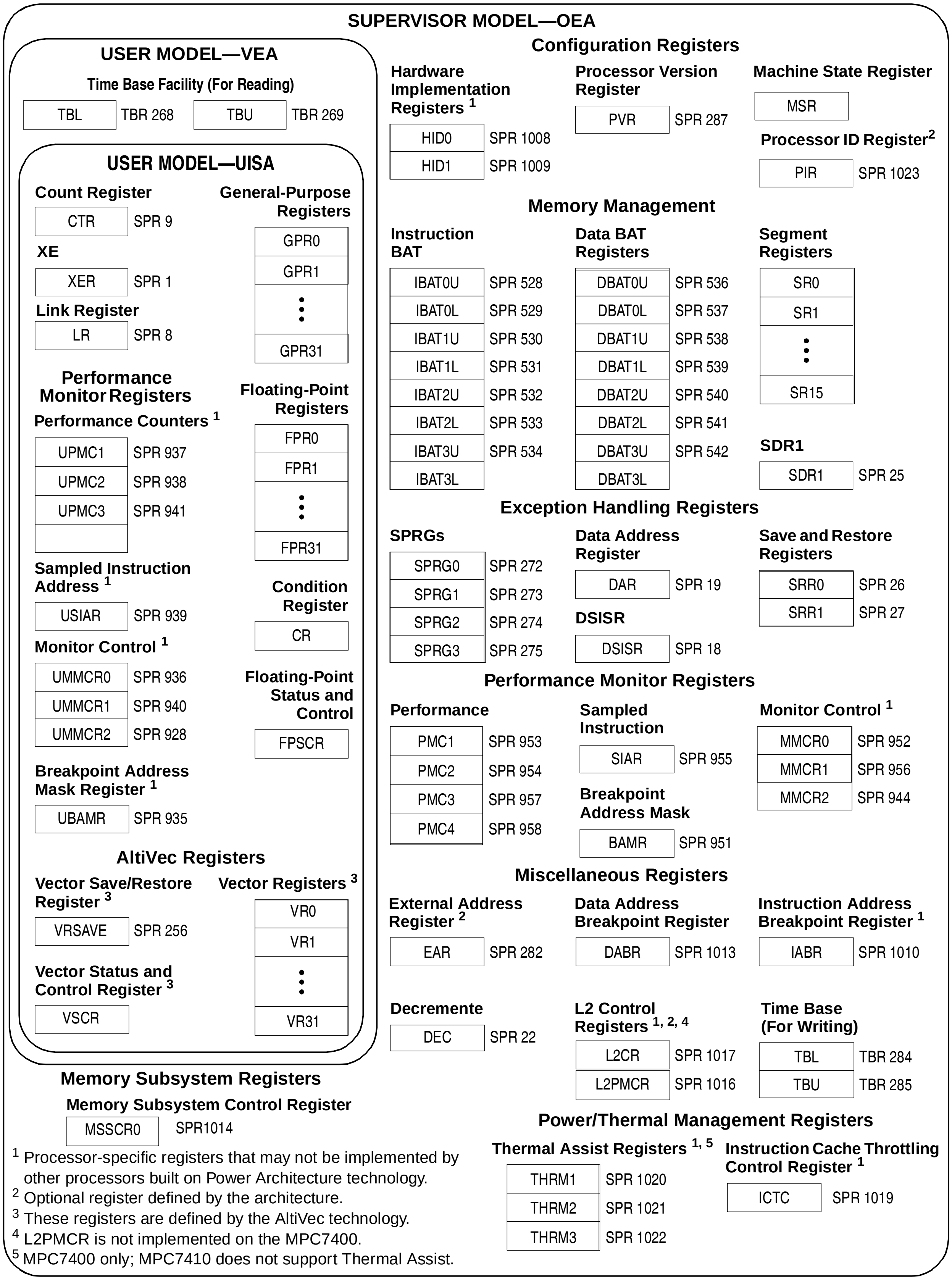
2.1 Register Set

This section describes the registers implemented in the MPC7410. It includes an overview of registers defined by the architecture and the AltiVec technology, highlighting differences in how these registers are implemented in the MPC7410, and a detailed description of MPC7410-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 architecture—user instruction set architecture (UISA), virtual environment architecture (VEA), and operating environment architecture (OEA). The 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 MPC7410 register set.



[1] Processor-specific registers that may not be implemented by other processors built on Power Architecture technology.

[2] Optional register defined by the architecture.

[3] These registers are defined by the AltiVec technology.

[4] L2PMCR is not implemented on the MPC7400.

[5] MPC7400 only; MPC7410 does not support Thermal Assist.

Figure 2-1. Programming Model—MPC7410 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 2-2 are SPRs; for example, VSCR and VRs are AltiVec registers and do not have an SPR number.

2.1.2 Register Set Summary

Table 2-1 summarizes the registers implemented in the MPC7410.

Table 2-1. Register Summary for the MPC7410

Name	SPR	Description	Reference
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 Std. 754.	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
UBAMR[1]	935	User breakpoint address mask register. Used with the events that monitor IABR and DABR hits. UBAMR provides user-level read access to the BAMR register.	11.3.2.1
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.	2.1.5.7 & 11.3.2.1, 2.1.5.7.4 & 11.3.3.1, 2.1.5.7.6 & 11.3.4.1
UPMC1–UPMC4[1]	937, 938 941, 942	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.	2.1.5.7.10, 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.	2.1.5.7.12, 11.3.7.1
VR0–VR31[2]	—	Vector registers (VR*n*). Data source and destination registers for all AltiVec instructions.	7.1.1.4

Table 2-1. Register Summary for the MPC7410 (continued)

Name	SPR	Description	Reference
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.	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 and IABR hits.	2.1.5.7.7, 11.3.5
DABR[4, 5]	1013	Data address breakpoint register. Optional register implemented in the MPC7410 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 MPC7410, 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 architecture.	PEM
HID0[1], [7] HID1[1, 8]	1008, 1009	Hardware implementation-dependent registers. Control various functions, such as enabling checkstop conditions, and locking, enabling, and invalidating the instruction and data caches. The HID1 reflects the state of PLL_CFG[0:3] clock signals.	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.	2.1.5.5

Table 2-1. Register Summary for the MPC7410 (continued)

Name	SPR	Description	Reference
IBAT0U/L,[10] IBAT1U/L,[10] IBAT2U/L,[10] IBAT3U/L,[10] DBAT0U/L,[11] DBAT1U/L,[11] DBAT2U/L,[11] DBAT3U/L,[11]	528, 529 530, 531 532, 533 534, 535 536, 537 538, 539 540, 541 542, 543	Block-address translation (BAT) registers. The 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). 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 MPC7410 implements IBAT[G]; however, attempting to execute code from an IBAT area with G = 1 causes an ISI exception.	PEM, 5.3
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.	2.1.5.6, 10.4
L2CR[1]	1017	L2 cache control register. Includes bits for enabling parity checking, setting the L2-to-processor clock ratio, and identifying the type of RAM used for the L2 cache implementation.	2.1.5.4.2
L2PMCR[1]	1016	L2 private memory control register Used to configure and operate the private memory feature.	2.1.5.4.1
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.	2.1.5.7.1, 11.3.2 2.1.5.7.3, 11.3.3 2.1.5.7.5, 11.3.4

Table 2-1. Register Summary for the MPC7410 (continued)

<table>
<tr><th>Name</th><th>SPR</th><th>Description</th><th>Reference</th></tr>
<tr><td>MSR[7]</td><td>—</td><td>Machine state register. Defines the processor state. The MSR can be modified by the <b>mtmsr</b>, <b>sc</b>, and <b>rfi</b> instructions. It can be read by the <b>mfmsr</b> 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 architecture. Note that setting MSR[EE] masks decrementer and external interrupt exceptions and MPC7410-specific system management, and performance monitor exceptions, and the MPC7400-specific thermal management exception.
<table>
<tr><td>Bit</td><td>Name</td><td>Description</td></tr>
<tr><td>6</td><td>VEC</td><td>AltiVec available. MPC7410 and AltiVec technology specific; optional to the PowerPC ISA.
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 (<b>dst(t)</b>, <b>dstst(t)</b>, and <b>dss</b>); the VRs and the VSCR are available to data streaming instructions even if VEC = 0. VRSAVE can be accessed even if VECþ = 0.</td></tr>
<tr><td>13</td><td>POW</td><td>Power management enable. MPC7410-specific and optional to the PowerPC ISA.
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 2-5.</td></tr>
<tr><td>29</td><td>PMM</td><td>Performance monitor marked mode. MPC7410-specific and optional to the PowerPC ISA. See Chapter 11, “Performance Monitor.”
0 Process is not a marked process.
1 Process is a marked process.</td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
</td><td>PEM,
2.1.3.2,
4.3</td></tr>
<tr><td>MSSCR0[1, 12]</td><td>1014</td><td>Memory subsystem control register. Used to configure and operate many aspects of the memory subsystem.</td><td>2.1.5.3</td></tr>
<tr><td>PIR</td><td>1023</td><td>Processor identification register. Provided for system use. The MPC7410 does not change PIR contents.</td><td>PEM</td></tr>
<tr><td>PMC1–PMC4[4]</td><td>953, 954
957, 958</td><td>Performance monitor counter registers (PMC<i>n</i>). Used to record the number of times a certain event has occurred. UPMCs provide user-level read access to these registers.</td><td>2.1.5.7.9,
11.3.6</td></tr>
<tr><td>PVR</td><td>287</td><td>Processor version register. Read-only register that identifies the version (model) and revision level of the processor.</td><td>PEM,
2.1.3.1</td></tr>
</table>

Table 2-1. Register Summary for the MPC7410 (continued)

Name	SPR	Description	Reference
SDAR, USDAR	—	Sampled data address register. The MPC7410 does not implement the optional registers (SDAR or the user-level, read-only USDAR register) defined by the architecture. However, for compatibility with processors that do, those registers can be written to by boot code without causing an exception. SDAR is SPR 959; USDAR is SPR 943.	2.1.5.7.13
SDR1[13]	25	Sample data register. Specifies the base address of the page table entry group (PTEG) address used in virtual-to-physical address translation.	PEM
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.	2.1.5.7.12 11.3.7
SPRG0– SPRG3	272–275	SPRG*n*. Provided for operating system use.	PEM,
SR0– SR15[14]	—	Segment registers (SR*n*). Note that the MPC7410 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 MPC7410 sets one or more error bits in SRR1. Refer to the individual exceptions for individual SRR1 bit settings.	PEM, 4.3
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.	PEM 2.1.4.1 2.3.5.1

Table 2-1. Register Summary for the MPC7410 (continued)

Name	SPR	Description	Reference
THRM1[15], THRM2[15], THRM3[15]	1020, 1021, 1022	Thermal management registers (THRM*n*). Used to enable and set thresholds for the thermal management facility. THRM1, THRM2—Provide the ability to compare the junction temperature against two user-provided thresholds. Dual thresholds give thermal management software differing degrees of action in lowering the junction temperature. The TAU can be also operated in a single threshold mode in which the thermal sensor output is compared to only one threshold in either THRM1 or THRM2. THRM3—Used to enable the thermal management assist unit (TAU) and to control the comparator output sample time.	2.1.5.6.1

[1] MPC7410-specific register may not be supported on other processors that implement the PowerPC ISA.

[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 ISA.

[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 2-22.

[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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] MPC7400-specific register. The MPC7410 does not support thermal management facility.

The 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.

NOTE

The MPC7410 fully decodes the SPR field of the instruction. If the SPR specified is undefined, an illegal instruction program exception occurs.

2.1.3 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 registers defined by the architecture contain additional implementation-specific information for the MPC7410.

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*.

NOTE

> The processor version number for the MPC7400 is 0x000C; the processor version number for the MPC7410 is 0x800C. The processor revision level starts at 0x0100 for the MPC7400 and 0x1100 for the MPC7410. The revision level is updated for each silicon revision. Table 2-2 describes the MPC7410 PVR bits that are not required by the architecture.

Table 2-2. Additional PVR Bits

B its	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 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 MPC7410 MSR is shown in Figure 2-2.

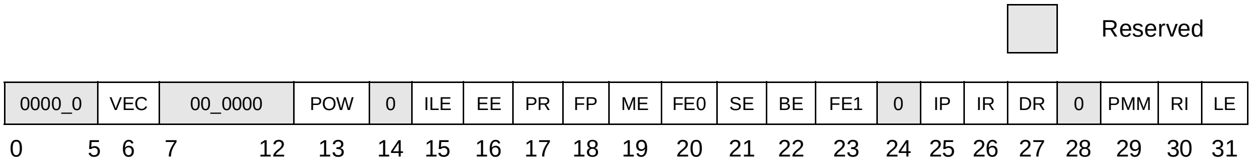


Figure 2-2. Machine State Register (MSR)

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 Management.”
14	—	Reserved. Implementation-specific
15	ILE	Exception little-endian mode. When an exception occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the exception.
16	EE	External interrupt enable 0 The processor delays recognition of external interrupts and decrementer exception conditions. 1 The processor is enabled to take an external interrupt or the decrementer exception.
17	PR[4]	Privilege level 0 The processor can execute both user- and supervisor-level instructions. 1 The processor can only execute user-level instructions.
18	FP[2]	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions and can take floating-point enabled program exceptions.
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**, **isync**, and **sc**. Successful execution means that the instruction caused no other exception.

Table 2-3. MSR Bit Settings (continued)

Bit(s)	Name	Description
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. MPC7410–specific; defined as optional by the architecture. For more information about the performance monitor marked mode bit, see Section 11.4, "Event Counting."
30	RI	Indicates whether system reset or machine check exception is recoverable. 0 Exception is not recoverable. 1 Exception is recoverable. The RI bit indicates whether from the perspective of the processor, it is safe to continue (that is, processor state data such as that saved to SRR0 is valid), but it does not guarantee that the interrupted process is recoverable.
31	LE[6]	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

[1] Optional to the PowerPC architecture

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

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

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

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

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

Note that setting MSR[EE] masks not only the architecture-defined external interrupt and decrementer exceptions but also the MPC7410-specific system management, performance monitor exceptions, and the MPC7400-specific thermal management 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 MPC7410 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 MPC7410 operates in floating-point precise mode.
1	0	Imprecise recoverable. For this setting, the MPC7410 operates in floating-point precise mode.
1	1	Floating-point precise mode

2.1.4 User-Level Registers (VEA)

The 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 special purpose register (**mfspr**) and the move from time base register (**mftb**) instructions. 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 MPC7410-Specific Register Descriptions

The architecture allows for implementation-specific SPRs. This section describes registers that are defined for the MPC7410 but are not included in the PowerPC ISA. Note that in the MPC7410, these registers are all supervisor-level registers. All the registers described in the *AltiVec Technology Programming Environments Manual* are implemented in MPC7410. 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 MPC7410-specific registers is consistent among processors built on the PowerPC ISA, 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 and power management registers. 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 MPC7410. The HID0 register is shown in Figure 2-3.

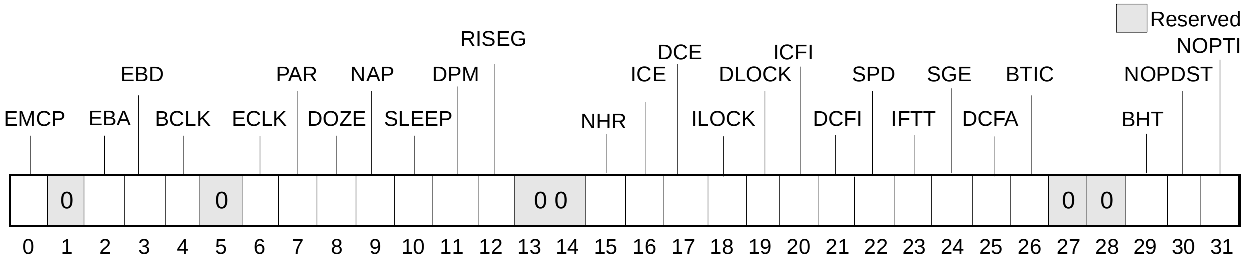


Figure 2-3. Hardware Implementation-Dependent Register 0 (HID0)

The HID0 bits are described in Table 2-5.

Table 2-5. HID0 Field Descriptions

Bits	Name	Description
0	EMCP	Enable $\overline{\text{MCP}}$. The primary purpose of this bit is to mask out further machine check exceptions caused by assertion of $\overline{\text{MCP}}$, similar to how MSR[EE] can mask external interrupts. 0 Masks $\overline{\text{MCP}}$. Asserting $\overline{\text{MCP}}$ stops generation of a machine check exception or a checkstop. 1 Asserting $\overline{\text{MCP}}$ causes a checkstop if MSR[ME] = 0, or a machine check exception if MSR[ME] = 1.
1	—	Reserved Defined as the DBP bit on some earlier processors. Parity generation is always enabled, but parity checking on the address or data buses is enabled only when the corresponding bit HID[EBA] or HID[EBD] is set.
2	EBA	Enable/disable system bus address parity checking 0 Prevents address parity checking. 1 Allows bus address parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. EBA and EBD allow the processor to operate with memory subsystems that do not generate parity.
3	EBD	Enable system bus data parity checking 0 Data parity checking is disabled. 1 Allows a data parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. EBA and EBD allow the processor to operate with memory subsystems that do not generate parity.
4	BCLK	CLK_OUT output enable and clock type selection Used in conjunction with HID0[ECLK] and the $\overline{\text{HRESET}}$ signal to configure CLK_OUT. See Table 2-6.

Table 2-5. HID0 Field Descriptions (continued)

Bits	Name	Description
5	—	Reserved Defined as HID0[5]: EICE on some earlier processors.
6	ECLK	CLK_OUT output enable and clock type selection Used in conjunction with HID0[BCLK] and the $\overline{\text{HRESET}}$ signal to configure CLK_OUT. See Table 2-6.
7	PAR	Disable precharge of $\overline{\text{ARTRY}}$ and SHD[0] or SHD[1] 0 Precharge of $\overline{\text{ARTRY}}$ enabled 1 Alters bus protocol slightly by preventing the processor from driving $\overline{\text{ARTRY}}$ to high (negated) state. If this is done, the system must restore the signals to the high state.
8	DOZE	Doze mode enable Operates in conjunction with MSR[POW]. 0 Doze mode disabled. 1 Doze mode enabled. Doze mode is invoked by setting MSR[POW] while this bit is set. In doze mode, the PLL, time base, and snooping remain active.
9	NAP	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.
10	SLEEP	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] to PLL bypass mode, and then disabling SYSCLK.
11	DPM	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	RISEG	Read I SEG (test only) 0 Data segment registers read by **mfsr**. 1 Instruction segment registers read by **mfsr**. See Section 2.3.6.3.2, "Translation Lookaside Buffer Management Instructions—OEA."
13–14	—	Reserved
15	NHR	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 MPC7410 never writes this bit unless executing an **mtspr**(HID0).

Table 2-5. HID0 Field Descriptions (continued)

Bits	Name	Description
16	ICE	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 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.
17	DCE	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 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.
18	ILOCK	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 unless the L2CR[L2DO] bit is set. 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 and L2 Cache Operation," gives further details.
19	DLOCK	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 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. To prevent locking during a cache access, a **sync** instruction must precede the setting of DLOCK and a **sync** must follow.
20	ICFI	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 MPC7410.

Table 2-5. HID0 Field Descriptions (continued)

Bits	Name	Description
21	DCFI	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 MPC7410.
22	SPD	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.
23	IFTT	I-Fetch TTx encoding differentiation 0 I-cache and D-cache reads are not differentiated. 1 TTx code for all D-cache reads are changed from READ (TTx = 01010) to READ ATOMIC (TTx = 11010). I-cache reads continue to be identified as READ (TTx = 01010). Defined as IFEM on some earlier microprocessors built on the PowerPC ISA.
24	SGE	Store gathering enable 0 Store gathering is disabled. 1 Integer store gathering is performed for write-through accesses to nonguarded space or for cache-inhibited stores to nonguarded space as described in Section 2.3.4.3.5, “Integer Store Gathering.”
25	DCFA	Data cache flush assist (Force data cache to ignore invalid sets on miss replacement selection.) 0 The data cache flush assist facility is disabled. 1 The miss replacement algorithm ignores invalid entries and follows the replacement sequence defined by the PLRU bits. This reduces the series of uniquely addressed load or **dcbz** instructions to eight per set. The bit should be set just before beginning a cache flush routine and should be cleared when the series of instructions is complete.
26	BTIC	Branch target instruction cache enable. Used to enable use of the 64-entry branch instruction cache. 0 The BTIC contents are invalidated and the BTIC behaves as if it were empty. New entries cannot be added until the BTIC is enabled. 1 The BTIC is enabled and new entries can be added. The BTIC is flushed by context synchronization, which is required after a move to HID0. Thus if the synchronization rules are followed, modifying this BTIC bit implicitly flushes the BTIC. See Chapter 6, “Instruction Timing,” for further details.
27	—	Reserved Defined as FBIOB on some earlier processors.
28	—	Reserved Defined as ABE on some earlier processors.

Table 2-5. HID0 Field Descriptions (continued)

Bits	Name	Description
29	BHT	Branch history table enable 0 BHT disabled. The MPC7410 uses static branch prediction as defined by the 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 512 entry branch history table (BHT). The BHT is disabled at power-on reset. All entries are set to weakly, not-taken.
30	NOPDST	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	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.

Table 2-6 shows how HID0[BCLK], HID0[ECLK], and $\overline{\text{HRESET}}$ are used to configure CLK_OUT. See Section 8.5.5.3, “Clock Out (CLK_OUT)—Output,” for more information.

Table 2-6. HID0[BCLK] and HID0[ECLK] CLK_OUT Configuration

$\overline{\text{HRESET}}$	HID0[ECLK]	HID0[BCLK]	CLK_OUT
Asserted	x	x	External bus clock (SYSCLK)
Negated	0	0	Reserved for factory
Negated	0	1	Reserved for factory
Negated	1	0	Core
Negated	1	1	External bus clock (SYSCLK)

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[0:3] signals. The HID1 bits are shown in Figure 2-4.

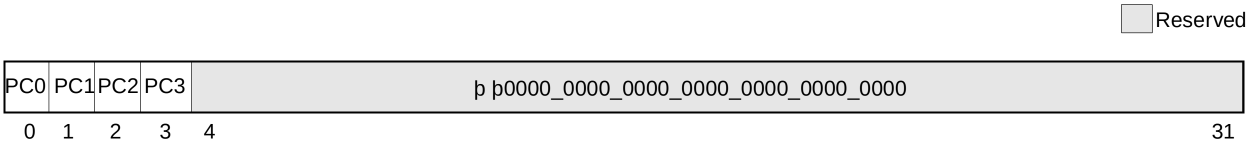


Figure 2-4. Hardware Implementation-Dependent Register 1 (HID1)

The HID1 bits are described in Table 2-7.

Table 2-7. HID1 Field Descriptions

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

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

HID1 can be accessed with **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 2-5, is used to configure and operate the memory subsystem for the MPC7410. It is accessed as SPR 1014. The MSSCR0 is initialized to all 0s except for the read-only bits.

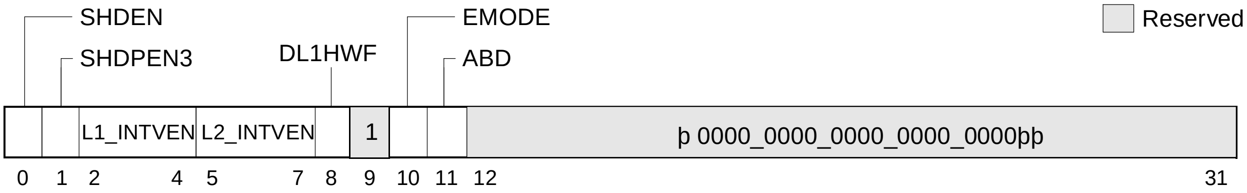


Figure 2-5. Memory Subsystem Control Register (MSSCR0)

Table 2-8 describes MSSCR0 fields.

Table 2-8. MSSCR0 Field Descriptions

Bits	Name	Function
0	SHDEN	Shared-state enable 0 3-state MEI protocol 1 4-state MESI protocol The MPC7410 implements both a 3-state MEI coherency protocol similar to the MPC750 and a 4-state MESI protocol similar to the MPC604e family of processors.
1	SHDPEN3	$\overline{\text{SHD0}}$/$\overline{\text{SHD1}}$ signal enable in 3-state MEI mode 0 $\overline{\text{SHD0}}$/$\overline{\text{SHD1}}$ signals are not sampled and are not driven when SHDEN = 0. $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ are always seen as negated by the processor. 1 $\overline{\text{SHD0}}$/$\overline{\text{SHD1}}$ signals sampled when SHDEN = 0. For some system implementations, MPC7410 can be inserted into an MPC750 socket that has no $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ connection. In this case, this control bit (and SHDEN) should remain cleared to prevent the processor from sampling indeterminate or floating signal input values on these signals. SHDPEN3 has an effect only when SHDEN = 0. If SHDEN = 1, $\overline{\text{SHD0}}$ is sampled if EMODE = 0, and $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ are sampled if EMODE = 1. For multiprocessor systems, when SHDEN = 0, SHDPEN3 must be set and the $\overline{\text{SHD}x}$ signal(s) must be connected between the processors. If either of these conditions are not met, the processor cannot guarantee the atomicity of an **lwarx**/**stwcx.** instruction pair. Note that $\overline{\text{SHD1}}$ is driven or sampled only in MPX bus mode (EMODE = 1), regardless of the state of this control bit. In 60x bus mode (EMODE = 0), the above statements apply to the $\overline{\text{SHD}}$ signal (multiplexed with $\overline{\text{SHD0}}$).
2–4	L1_INTVEN	L1 data cache $\overline{\text{HIT}}$ intervention enable 000 $\overline{\text{HIT}}$ intervention disabled. All Modified intervention is performed using the 60x-style ARTRY/window-of-opportunity write-with-kill push. $\overline{\text{HIT}}$ intervention occurs for snoop hits to lines in the following states: 100 Modified 110 Modified or exclusive 111 Modified, exclusive, or recent. Shared (recent) intervention uses a 5-state MERSI coherency protocol. Bits 001, 010, 011, and 101 are illegal. These bits have an effect only when the processor is configured in MPX bus mode ($\overline{\text{EMODE}}$ signal asserted during $\overline{\text{HRESET}}$, which sets MSSCR0[EMODE]). The following is the only legal combination of values for L1 and L2 intervention enables: L1_INTVEN[0–2]||L2INTVEN[0–2] = 000 || 000 No $\overline{\text{HIT}}$ intervention 100 || 000 110 || 000 111 || 000 100 || 100 110 || 100 111 || 100 110 || 110 111 || 110 111 || 111 Full $\overline{\text{HIT}}$ intervention. MPC7410 does not support different L1_INTVEN or L2_INTVEN settings in different MPC7410 processors in a multiple processor system.
5–7	L2_INTVEN	L2 $\overline{\text{HIT}}$ intervention enable Same definition as for L1_INTVEN.

Table 2-8. MSSCR0 Field Descriptions (continued)

Bits	Name	Function
8	DL1HWF	L1 data cache hardware flush Refer to Section 3.5.2, “Data Cache Hardware Flush Parameter in MSSCR0,” for more details.
9	—	Reserved
10	EMODE	MPX bus mode (read-only) 0 Processor is in 60x bus mode ($\overline{\text{EMODE}}$ was sampled negated at $\overline{\text{HRESET}}$ negation). 1 Processor is in MPX bus mode. ($\overline{\text{EMODE}}$ was sampled asserted at $\overline{\text{HRESET}}$ negation).
11	ABD	Address bus driven (read-only) Valid only when EMODE = 1. 0 Processor drives the address bus only in the interval from $\overline{\text{TS}}$ through $\overline{\text{AACK}}$ (if after $\overline{\text{HRESET}}$ is negated, $\overline{\text{EMODE}}$ is detected as negated). 1 Processor drives the address bus to a stable value every cycle following a qualified bus grant i(f after $\overline{\text{HRESET}}$ was negated $\overline{\text{EMODE}}$ is detected as asserted). This mode is provided to enhance the electrical characteristics of the address bus in MPX bus mode by not allowing the address bus to float to indeterminate values when this processor is parked on the bus.
12–31	—	Reserved

Because the MSSCR0 parameters SHDEN, SHDPEN3, L1_INTVEN, and L2_INTVEN alter how the MPC7410 responds to snoop requests, it is important that changes to these parameters are handled correctly.

The correct sequence necessary to change the values for HDEN, SHDPEN3, L1_INTVEN, and L2_INTVEN is as follows:

1. disable interrupts
2. **dssall**
3. **sync**
4. Flush L1 data cache
5. Flush L2 cache
6. **sync**
7. **mtspr**(MSSCR0)
8. **sync**

Note that it is unnecessary to follow the above sequence when changing the MSSCR0[DL1HWF].

2.1.5.4 Instruction and Data Cache Registers

There are several registers used for configuring and controlling the various L1, and L2 caches. Along with the cache registers (L2PMCR and L2CR), HID0 is used in configuring the caches. Details of how the various cache registers are used is discussed below. See the Chapter 3, “L1 and L2 Cache Operation,” for further details on configuring the cache.

2.1.5.4.1 L2 Private Memory Control Register (L2PMCR)—MPC7410 Only

The L2 private memory control register, shown in Figure 2-6, is a supervisor-level, implementation-specific SPR used to configure and operate the L2 cache. Note that the MPC7400 does not support private memory and does not implement the L2PMCR. It is cleared by a hard reset or power-on reset. The L2PMCR can be accessed with the **mtspr** and **mfspr** instructions using SPR 1016.

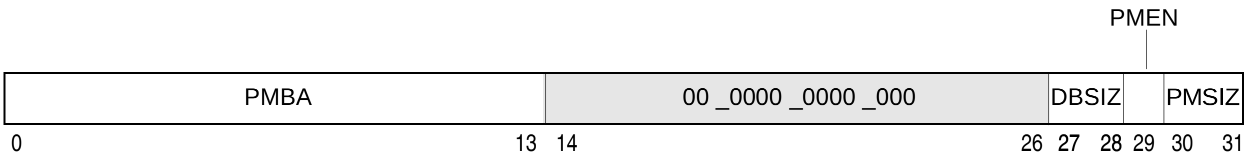


Figure 2-6. L2 Private Memory Control Register (L2PMCR)—MPC7410 Only

The L2 private memory control register is described in Chapter 3, “L1 and L2 Cache Operation.” The L2PMCR bits are described in Table 2-9.

Table 2-9. L2PMCR Field Descriptions—MPC7410 Only

Bits	Name	Description
0–13	PMBA	Private memory base address PA[0:10] for 2 Mbytes PA[0:11] for 1 Mbyte PA[0:12] for 512 Kbytes PA[0:13] for 256 Kbytes
14–26	—	Reserved Must be 0x000 for proper operation.
27–28	DBSIZ	L2 data bus size 00 64 bit 10 32 bit 01 Reserved 11 Reserved
29	PMEN	Private memory enable 0 Private memory disabled 1 Private memory enabled
30–31	PMSIZ	Private memory size 00 2 Mbytes 01 256 Kbytes 10 512 Kbytes 11 1 Mbyte

2.1.5.4.2 L2 Cache Control Register (L2CR)

The L2 cache control register (L2CR), shown in Figure 2-7, is a supervisor-level, implementation-specific SPR used to configure and operate the L2 cache. It is cleared by a hard reset or power-on reset. The L2CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1017.

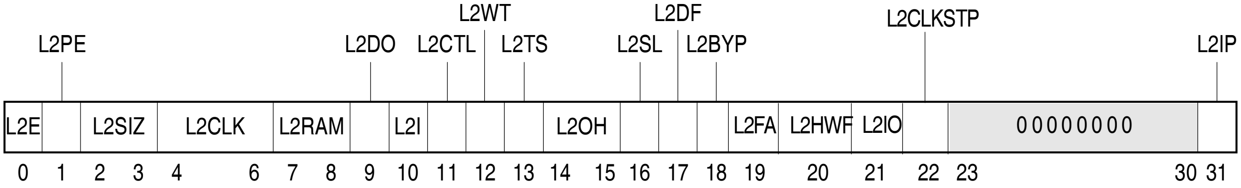


Figure 2-7. L2 Cache Control Register (L2CR)

The L2 cache interface is described in Chapter 3, "L1 and L2 Cache Operation." The L2CR bits are described in Table 2-10.

Table 2-10. 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, the L2 clock must be configured through L2CR[2CLK], and the L2 DLL must stabilize (see the *MPC7410 Hardware Specifications* for further details). All other L2CR bits must be set appropriately. The L2 cache may need to be invalidated globally. If the L2 cache is enabled, the L1 data cache must also be enabled.
1	L2PE	L2 data parity checking enable 0 L2 odd data parity disabled 1 L2 odd data parity enabled Odd parity checking for the L2 data RAM interface. When L2PE is set, it allows a data parity error on the L2 bus to cause a checkstop if MSR[ME] = 0, or a machine check exception if MSR[ME] = 1. The MPC7410 always generates L2 data parity.
2–3	L2SIZ	L2 size Should be set according to the size of the L2 data RAMs as follows: 00 2 Mbyte, 128 bytes (4 sectors) per tag 01 256 Kbyte, 32 bytes (1 sector) per tag 10 512 Kbyte, 32 bytes (1 sector) per tag 11 1 Mbyte, 64 bytes (2 sectors) per tag A 256-Kbyte L2 cache requires a data RAM configuration of 32 Kbytes x 64 bits; a 512-Kbyte L2 cache requires a configuration of 64 Kbyte x 64 bits; a 1-Mbyte L2 cache requires a configuration of 128K x 64 bits.

Table 2-10. L2CR Field Descriptions (continued)

Bits	Name	Description
4–6	L2CLK	L2 clock ratio (core-to-L2 frequency divider) Specifies the clock divider ratio based on the core clock frequency at which the L2 data SRAM interface operates. When these bits are cleared, the L2 clock is stopped and the on-chip DLL for the L2 interface is disabled. For nonzero values, the processor generates the L2 clock and the on-chip DLL is enabled. After the L2 clock ratio is chosen, the DLL must stabilize before the L2 interface can be enabled. (See the *MPC7410 Hardware Specifications* for further details). The resulting L2 clock frequency cannot be slower than the clock frequency of the 60x bus interface. 000 L2 clock and DLL disabled 001 ÷1 010 ÷1.5 011 ÷3.5 100 ÷2 101 ÷2.5 110 ÷3 111 ÷4
7–8	L2RAM	L2 RAM type Configures the L2 SRAM interface for the type of synchronous SRAMs used: • Pipelined (register-register) synchronous burst SRAMs that clock addresses in and clock data out • Late-write synchronous SRAMs, for which the MPC7410 requires a pipelined (register-register) configuration. Late-write RAMs require write data to be valid on the cycle after $\overline{\text{WE}}$ is asserted, rather than on the same cycle as the write enable as with traditional burst RAMs. • Newer generation pipeline burst SRAMs, referred to as PB3-type SRAMs For burst RAM selections, the MPC7410 does not use the burst feature of the SRAM; it generates an address for each access. 00 Reserved 01 PB3 SRAM 10 Pipelined (register-register) synchronous burst SRAM (PB2) 11 Pipelined (register-register) synchronous late-write SRAM
9	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 transactions from the L1 data cache can be cached in the L2 cache. L1 instruction cache operations 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 L2DO and L2IO effectively locks the L2 cache.
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.
11	L2CTL	L2 RAM control (ZZ enable) Enables the automatic operation of the L2ZZ (low-power mode) signal for cache RAMs that support the ZZ function. While L2CTL is set, L2ZZ asserts automatically when the MPC7410 enters nap or sleep mode and negates automatically when the MPC7410 exits nap or sleep mode. This bit should not be set when the MPC7410 is in nap mode and snooping is to be performed through the negation of $\overline{\text{QACK}}$.

Table 2-10. L2CR Field Descriptions (continued)

Bits	Name	Description
12	L2WT	L2 write-through Selects write-through mode (rather than the default write-back mode) so all writes to the L2 cache also write through to the system bus. For these writes, the L2 cache entry is always marked as clean (valid unmodified) rather than dirty (valid modified). This bit must never be asserted after the L2 cache has been enabled as previously-modified lines can get remarked as clean during normal operation.
13	L2TS	L2 test support Causes cache block pushes from the L1 data cache that result from **dcbf** and **dcbst** instructions to be written only into the L2 cache and marked valid, rather than being written only to the system bus and marked invalid in the L2 cache in case of a hit. This bit allows a **dcbz**/**dcbf** instruction sequence to be used with the L1 cache enabled to easily initialize the L2 cache with any address and data information. This bit also keeps **dcbz** instructions from being broadcast on the system and single-beat cacheable store misses in the L2 from being written to the system bus.
14–15	L2OH	L2 output hold Configure output hold time for address, data, and control signals driven by the MPC7410 to the L2 data RAMs. They should generally be set according to the SRAM’s input hold time requirements, for which late-write SRAMs usually differ from burst SRAMs. 00 Shortest output hold 01 Short output hold 10 Long output hold 11 Longest output hold See the *MPC7410 Hardware Specifications* for specific output hold times.
16	L2SL	L2 DLL slow Increases the delay of each tap of the DLL delay line. It is intended to increase the delay through the DLL to accommodate slower L2 RAM bus frequencies. Generally, L2SL should be set if the L2 RAM interface is operated below 150 MHz.
17	L2DF	L2 differential clock Configures the two clock-out signals (L2CLK_OUTA and L2CLK_OUTB) of the L2 interface to operate as one differential clock. In this mode, the B clock is driven as the logical complement of the A clock. This mode supports the differential clock requirements of late-write SRAMs. Generally, this bit should be set when late-write SRAMs are used.
18	L2BYP	L2 DLL bypass The DLL unit receives three input clocks: • A square-wave clock from the PLL unit to phase adjust and export • A non-square-wave clock for the internal phase reference • A feedback clock (L2SYNC_IN) for the external phase reference. Causes clock #2 to be used as clocks #1 and #2. (Clock #2 is the actual clock used by the registers of the L2 interface circuitry.) L2BYP is intended for use when the PLL is being bypassed. If the PLL is being bypassed, the DLL must be operated in divide-by-1 mode, and SYSCLK must be fast enough for the DLL to support.
19	L2FA	L2 flush assist (for software flush) When this bit is negated, all lines castout from the L1 data cache that have a state of CDMRSV=01xxx1 (i.e. C-bit negated), do not allocate in the L2 if they miss. Setting this bit forces every castout from the data cache to allocate an entry in the L2 if that castout misses in the L2 regardless of the state of the C-bit. The L2FA bit must be set and the L2IO bit must be cleared in order to use the software flush algorithm provided in Section 3.7.3.8.2, “L2 Cache Software Flush.”

Table 2-10. 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 3.7.3.8.1, “L2 Cache Hardware Flush,” for more information.
21	L2IO	L2 instruction-only mode Setting this bit enables instruction-only operation in the L2 cache. For this operation, only transactions from the L1 instruction cache are allowed to be reloaded in the L2 cache. Data addresses already in the cache will still hit for the L1 data cache. When both L2DO and L2IO are asserted, the L2 cache is effectively locked.
22	L2CLKSTP	L2 clock stop Enables the automatic stopping of the L2CLK_OUT signals for cache rams that support this function. While L2CLKSTP is set, the L2CLK_OUT signals will automatically be stopped when MPC7410 enters nap or sleep mode, and automatically restarted when MPC7410 exits nap or sleep.
23	L2DRO	L2DLL rollover checkstop enable Enables a potential rollover (or actual rollover) condition of the DLL to cause a checkstop for the processor. A potential rollover condition occurs when the DLL is selecting the last tap of the delay line, and thus can risk rolling over to the first tap with one adjustment while in the process of keeping in sync. Such a condition is improper operation for the DLL, and while this condition is not expected, this bit allows detection for added security. This bit can be set when the DLL is first enabled (set with the L2CLK bits) to detect rollover during initial synchronization. It can also be set when the L2 cache is enabled (with L2E bit) after the DLL has achieved initial lock.
24–30	—	Reserved
31	L2IP	L2 global invalidate in progress (read only) This read-only bit indicates whether an L2 global invalidate operation is in progress. It should be monitored after an L2 global invalidate operation has been initiated by the L2I bit to determine when it has completed.

2.1.5.5 Instruction Address Breakpoint Register (IABR)

The instruction address breakpoint register (IABR), shown in Table 2-8, 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 4.6.15, “Instruction Address Breakpoint Exception (0x01300).” The IABR can be accessed with **mtspr** and **mfspr** using the SPR 1010. The MPC7410 requires that an **mtspr**[IABR] be followed by a context synchronizing instruction. The MPC7410 may not generate a breakpoint response for that context synchronizing instruction if the breakpoint was enabled by

mtspr[IABR] immediately preceding it. The MPC7410 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."

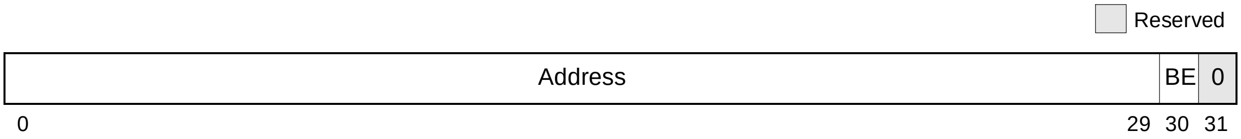


Figure 2-8. Instruction Address Breakpoint Register

The IABR bits are described in Table 2-11.

Table 2-11. 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	—	Reserved

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

2.1.5.6 Thermal Management Registers—MPC7400 Only

The MPC7400 features an on-chip thermal assist unit (TAU) provides the following functions:

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

Control and access to the TAU is through the privileged **mtspr**/**mfspr** instructions to the three THRM registers. Also, junction temperature can be controlled with the ICTC register. Note that the MPC7410 does not support the thermal assist unit.

2.1.5.6.1 Thermal Management Registers (THRM1–THRM3)—MPC7400 Only

THRM1 and THRM2, shown in Figure 2-9, provide the ability to compare the junction temperature against two user-provided thresholds. Having dual thresholds allows thermal management software differing degrees of action in reducing junction temperature. The thermal management unit can also use a single-threshold mode in which the thermal sensor output is compared to only one threshold in either THRM1 or THRM2.

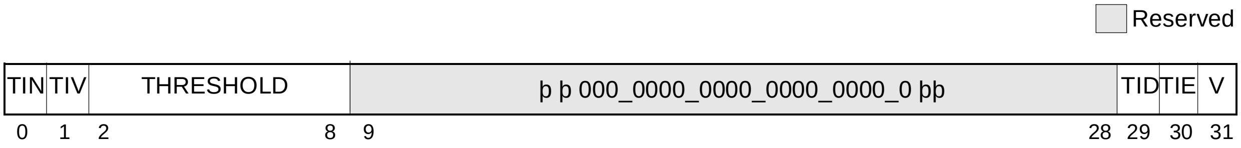


Figure 2-9. Thermal Management Registers 1–2 (THRM1–THRM2)—MPC7400 Only

The fields in THRM1 and THRM2 are described in Table 2-12.

Table 2-12. THRM1–THRM2 Bit Settings—MPC7400 Only

Bits	Field	Description
0	TIN	Thermal management interrupt bit. Read only. The state of this bit is valid only if TIV is set. The interpretation of TIN is controlled by TID. 0 The thermal sensor output has not crossed the threshold specified in the SPR. 1 The thermal sensor output crossed the threshold specified in the SPR.
1	TIV	Thermal management interrupt valid. Read only. 0 The thermal management interrupt (TIN) state is invalid. TIV is cleared by writing to the register. 1 The thermal management interrupt (TIN) state is valid. TIV is cleared by writing to the register.
2–8	Threshold	Threshold value that the output of the thermal sensor is compared to. The threshold range is 0•–127•C, and each bit represents 1•C. Note that this is not the thermal sensor resolution.
9–28	—	Reserved, should be cleared.
29	TID	Thermal management interrupt direction bit. Selects the result of the temperature comparison to set TIN bit and to assert a thermal management interrupt if TIE = 1. 0 TIN is set and an interrupt occurs if the junction temperature exceeds the threshold. 1 TIN is set and an interrupt is indicated if the junction temperature is below the threshold.
30	TIE	Thermal management interrupt enable. Allows system software to make a successive approximation to estimate the junction temperature. 0 If V = 1, TIN records the status of the junction temperature vs. threshold comparison without asserting an interrupt signal. 1 The thermal management interrupt signal is enabled. The thermal management interrupt is masked by setting MSR[EE].
31	V	Valid bit. 0 The threshold, TID, and TIE bits are invalid. 1 The threshold, TID, and TIE bits are valid. Setting THRM1[V], THRM2[V], and THRM3[E] enables operation of the thermal sensor.

The execution of an **mtspr** instruction to THRM*n* anytime during a TAU operation clears THRM*n*[TIV] and restarts the temperature comparison. Executing an **mtspr** instruction to THRM3 clears THRM1[TIV] and THRM2[TIV] and restarts temperature comparison in THRM*n* if THRM3[E] = 1.

Examples of valid THRM1 and THRM2 bit settings are shown in Table 2-13.

Table 2-13. Valid THRM1/THRM2 States—MPC7400 Only

TIN[1]	TIV[1]	TID	TIE	V	Description
x	x	x	x	0	Threshold in the SPR is not used for comparison.
x	x	x	0	1	Threshold is used for comparison; thermal management interrupt assertion is disabled.
x	x	0	0	1	Set TIN and do not assert thermal management interrupt if the junction temperature exceeds the threshold.
x	x	0	1	1	Set TIN and assert thermal management interrupt if the junction temperature exceeds the threshold.

Table 2-13. Valid THRM1/THRM2 States—MPC7400 Only (continued)

TIN[1]	TIV[1]	TID	TIE	V	Description
x	x	1	0	1	Set TIN and do not assert thermal management interrupt if the junction temperature is less than the threshold.
x	x	1	1	1	Set TIN and assert thermal management interrupt if the junction temperature is less than the threshold.
x	0	x	x	1	The state of the TIN bit is not valid.
0	1	0	x	1	The junction temperature is less than the threshold and as a result the thermal management interrupt is not generated for TIE = 1.
1	1	0	x	1	The junction temperature is greater than the threshold and as a result the thermal management interrupt is generated if TIE = 1.

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

The THRM3 register, shown in Figure 2-10, is used to enable the thermal assist unit and to control the comparator output sample time. The thermal assist logic manages the thermal management interrupt generation and time-multiplexed comparisons in dual-threshold mode as well as other control functions.

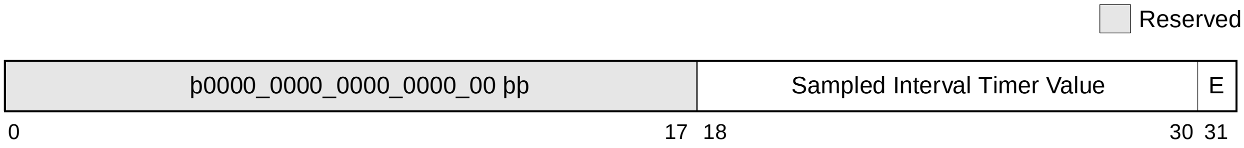


Figure 2-10. Thermal Management Register 3 (THRM3)—MPC7400 Only

The bits in THRM3 are described in Table 2-14.

Table 2-14. THRM3 Bit Settings—MPC7400 Only

Bits	Name	Description
0–17	—	Reserved for future use. System software should clear these bits when writing to THRM3.
0–14	—	Reserved, should be cleared. When writing to THRM3[0–14], the system software should read from THRM3[0–14] first to preserve the values.
18–30	SITV	Sample interval timer value. Number of elapsed system bus clock cycles before a junction temperature vs. threshold comparison result is sampled in order for TIN to be set and an interrupt to be generated. The value should be greater than 20 µs. This is necessary due to the thermal sensor, DAC, and the analog comparator settling time being greater than the bus cycle time.
31	E	Enables the thermal sensor compare operation if either THRM1[V] or THRM2[V] = 1.

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

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

2.1.5.6.2 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 2-11. The overall junction temperature reduction comes from the dynamic power management of each functional unit when the MPC7410 is idle in between instruction fetches. Phase-locked loop (PLL) and delay-locked loop (DLL) configurations are unchanged.

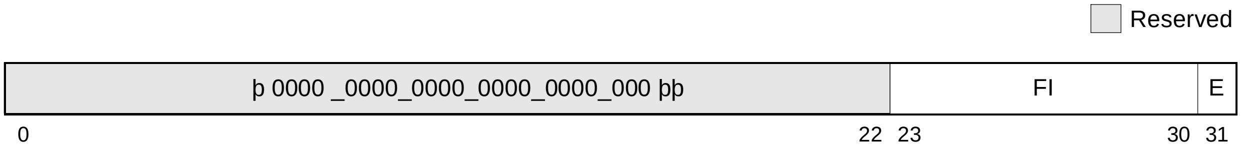


Figure 2-11. Instruction Cache Throttling Control Register (ICTC)

Table 2-15 describes the bit fields for the ICTC register.

Table 2-15. ICTC Field Descriptions

Bits	Name	Description
0–22	—	Reserved The bits should be cleared.
23–30	FI	Instruction forwarding interval expressed in processor clocks 0x00 0 clock cycle. 0x01 1 clock cycle . . . 0xFF 255 clock cycles
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]. 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.

2.1.5.7 Performance Monitor Registers

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

2.1.5.7.1 Monitor Mode Control Register 0 (MMCR0)

The monitor mode control register 0 (MMCR0), shown in Figure 2-12, 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 2.1.5.7.2, “User Monitor Mode Control Register 0 (UMMCR0).”

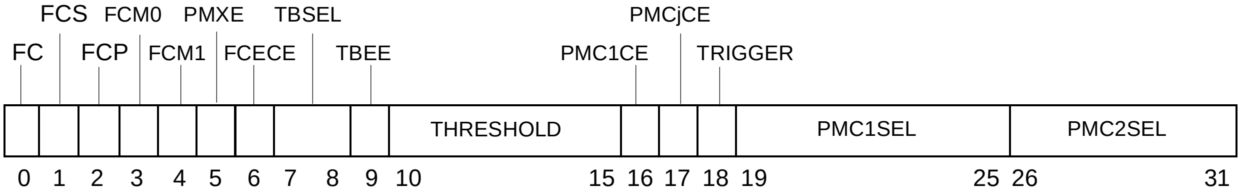


Figure 2-12. Monitor Mode Control Register 0 (MMCR0)

This register is automatically cleared at power-up. Reading this register does not change its contents. Table 2-16 describes MMCR0 fields.

Table 2-16. 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 2-16. 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, which is a value such that only events that exceed the value are counted (PMC1 events 11, 19, and 20). 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. Note that MMCR2[THRESHMULT] chooses whether this value is multiplied by 2 or 32.
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 2-16. MMCR0 Field Descriptions (continued)

Bits	Name	Description
17	PMCjCE	PMCj condition enable. Controls whether counter negative conditions due to a negative value in any PMCj (that is, in any PMC except PMC1) are enabled. 0 Counter negative conditions for all PMCjs are disabled. 1 Counter negative conditions for all PMCjs 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 PMCjs are not incremented until PMC1 is negative or an enabled timebase or event occurs, at which time the PMCjs 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 PMCjs when PMC1 becomes negative without causing a performance monitor exception. Then freeze all PMCs (and optionally cause a performance monitor exception) when a PMCj becomes negative. The PMCjs then reflect the events that occurred after PMC1 became negative and before PMCj becomes negative. This use requires the following MMCR0 bit settings. –TRIGGER = 1 –PMC1CE = 0 –PMCjCE = 1 –TBEE = 0 –FCECE = 1 –PMXE = 1 (if a performance monitor exception is desired) • Resume counting in the PMCjs when PMC1 becomes negative, and cause a performance monitor exception without freezing any PMCs. The PMCjs 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.

2.1.5.7.2 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.

2.1.5.7.3 Monitor Mode Control Register 1 (MMCR1)

The monitor mode control register 1 (MMCR1) functions as an event selector for performance monitor counter registers 3,and 4 (PMC3and, PMC4). The MMCR1 register is shown in Figure 2-13.

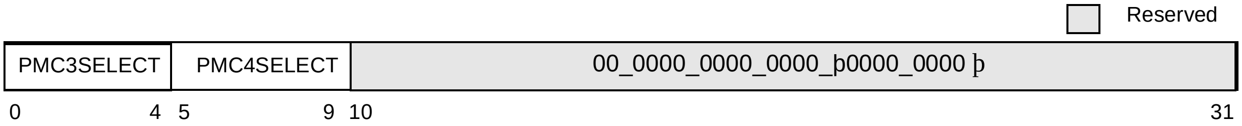


Figure 2-13. Monitor Mode Control Register 1 (MMCR1)

Bit settings for MMCR1 are shown in Table 2-17. The corresponding events are described in Section 2.1.5.7.9, “Performance Monitor Counter Registers (PMC1–PMC4).”

Table 2-17. 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.
10–31	—	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 2.1.5.7.4, “User Monitor Mode Control Register 1 (UMMCR1).”

2.1.5.7.4 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.

2.1.5.7.5 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 2-14.

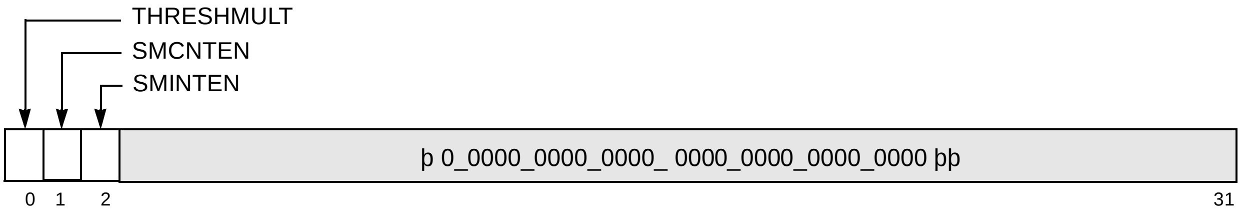


Figure 2-14. Monitor Mode Control Register 2 (MMCR2)

Table 2-18 describes MMCR2 fields.

Table 2-18. 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	SMCNTEN	SMCNTEN is used to mask the request from a peripheral performance monitor. 0 Ignore $\overline{\text{PMON\_IN}}$. 1 Start counting when $\overline{\text{PMON\_IN}}$ is asserted. Note that counting is subject to other enabling control bits in MMCR0.
2	SMINTEN	SMINTEN is used to mask the performance monitor exception request from a peripheral performance monitor. 0 Ignore $\overline{\text{SMI}}$. 1 When $\overline{\text{SMI}}$ is asserted, take a performance monitoring interrupt if enabled in MMCR0 and MSR[EE]. This event can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]). When SMINTEN = 1, the MPC7410 never takes an $\overline{\text{SMI}}$.

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 2.1.5.7.6, “User Monitor Mode Control Register 2 (UMMCR2).”

2.1.5.7.6 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.

2.1.5.7.7 Breakpoint Address Mask Register (BAMR)

The breakpoint address mask register (BAMR), shown in Figure 2-15, is used in conjunction with the events that monitor IABR and DABR hits.

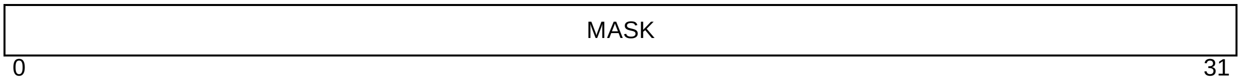


Figure 2-15. Breakpoint Address Mask Register (BAMR)

Table 2-19 describes BAMR fields.

Table 2-19. BAMR Field Descriptions

Bit	Name	Description
0–31	MASK	Used with events (PMC1 events 9 and 10) that monitor IABR and DABR hits The addresses to be compared for an IABR or DABR match are affected by the value in BAMR: • IABR hit (PMC1, event 8) 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] • DABR hit (PMC1, event 9) occurs if DABR_CMP (that is, DABR AND BAMR) = effective_address_compare (that is, EA AND BAMR). DABR_CMP[0–28] = DABR[0–28] AND BAMR[0–28] effective_addr_cmp[0–28] = effective_addr[0–28] AND BAMR[0–28] Be aware that breakpoint events 9 and 10 of PMC1 can be used to trigger ISI and DSI exceptions when the performance monitor detects an enabled overflow. This feature supports debug purposes and occurs only when IABR[30] or DABR[30–31] are set. To avoid taking one of the above interrupts, make sure that IABR[30] and/or DABR[30–31] are cleared.

BAMR can be accessed with **mtspr** and **mfspr** using SPR 951. For synchronization requirements on the register see Section 2.3.2.4, “Synchronization.”

User-level software can read the contents of BAMR by issuing an **mfspr** instruction to UBAMR, described in Section 2.1.5.7.8, “User Breakpoint Address Mask Register (UBAMR).”

2.1.5.7.8 User Breakpoint Address Mask Register (UBAMR)

The contents of BAMR are reflected to UBAMR, which can be read by user-level software. UBAMR can be accessed with the **mfspr** instructions using SPR 935.

2.1.5.7.9 Performance Monitor Counter Registers (PMC1–PMC4)

PMC1–PMC4, shown in Figure 2-16, are 32-bit counters that can be programmed to generate a performance monitor exception when they overflow.

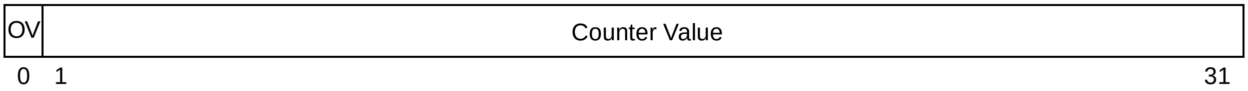


Figure 2-16. Performance Monitor Counter Registers (PMC1–PMC4)

The bits contained in the PMC registers are described in Table 2-20.

Table 2-20. PMCj 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[PMCjCE] 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 PMCj (j > 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[PMCjCE] 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

2.1.5.7.10 User Performance Monitor Counter Registers (UPMC1–UPMC4)

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

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

2.1.5.7.11 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 2-17.

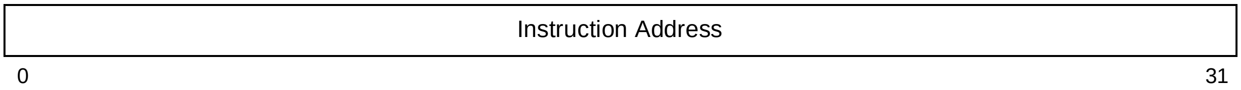


Figure 2-17. Sampled Instruction Address Registers (SIAR)

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.

2.1.5.7.12 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.

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

The MPC7410 does not implement the sampled data address register (SDA) or the user-level, read-only USDA registers. However, for compatibility with processors that do, those registers can be written to by boot code without causing an exception. SDA is SPR 959; USDA is SPR 943.

2.1.6 Reset Settings

Table 2-21 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 2-21. Settings Caused by Hard Reset (Used at Power-On)

Resource	Setting
BAMR	0x0000_0000
BATs	Undefined
Caches (L1/L2)[1]	Invalidated and Disabled.
CR	Undefined
CTR	Undefined
DABR	Breakpoint is disabled. Address is undefined.
DAR	0x0000_0000
DEC	0xFFFF_FFFF

Table 2-21. Settings Caused by Hard Reset (Used at Power-On) (continued)

Resource	Setting
DSISR	0x0000_0000
EAR	0x0000_0000
FPRs	Undefined
FPSCR	0x0000_0000
GPRs	Undefined
HID0	0x0000_0000
HID1	0x0000_0000
IABR	0x0000_0000 (Breakpoint is disabled.)
ICTC	0x0000_0000
L2CR	0x0000_0000
L2PMCR	0x0000_0000
LR	0x0000_0000
MMCR*n*	0x0000_0000
MSSCR0	0x0040_0000 (EMODE and ABD depend on hardware signals.)
MSSSR0	0x0040_00000x0000_0000
MSR	0x0000_0040 (only IP set)
PIR	0x0000_0000
PMC*n*	Undefined
PVR	0x800C_xxxx, where xxxx depends on the revision level, starting at 1100
Reservation address	Undefined
Reservation flag	Cleared
SDR1	0x0000_0000
SIAR	0x0000_0000
SPRG0–SPGR3	0x0000_0000
SRs	Undefined
SRR0	0x0000_0000
SRR1	0x0000_0000
TBU and TBL	0x0000_0000
THRM1–THRM3	0x0000_0000
TLBs	Undefined
UBAMR	0x0000_0000
UMMCR*n*	0x0000_0000
UPMC*n*	0x0000_0000

Table 2-21. Settings Caused by Hard Reset (Used at Power-On) (continued)

Resource	Setting
USIAR	0x0000_0000
VRs	Undefined
VRSAVE	0x0000_0000
VSCR	0x0001_0000
XER	0x0000_0000

[1] The processor automatically begins operations by issuing an instruction fetch. Because caching is inhibited at start-up, this generates a single-beat load operation on the bus.

2.2 Operand Conventions

This section describes the operand conventions as they are represented in two levels of the 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 Std. 754 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 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 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 MPC7410 does not provide hardware support for floating-point memory that is not word-aligned. If a floating-point operand is not word-aligned, the MPC7410 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 MPC7410 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 Std. 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 MPC7410 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 MPC7410. 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 MPC7410. Instructions that are implementation specific are described in Section 2.6.2, “AltiVec Instructions with Specific Implementations for the MPC7410.”

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 ISA 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 ISA should not assume the existence of mnemonics not described in that document.

2.3.1 Classes of Instructions

The MPC7410 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 built on the PowerPC ISA, 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 MPC7410.

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 ISA, except as stated in the instruction descriptions in Chapter 8, “Instruction Set,” of *The Programming Environments Manual*. The MPC7410 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 MPC7410 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 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 architecture can define any of these instructions to perform new functions.
- Instructions defined in the architecture but not implemented in a specific implementation. For example, instructions that can be executed on 64-bit processors built on Power Architecture technology are considered illegal by 32-bit processors such as the MPC7410.

  The following primary opcodes are defined for 64-bit implementations only and are illegal on the MPC7410:

  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 MPC7410 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 architecture.

2.3.1.4 Reserved Instruction Class

Reserved instructions are allocated to specific implementation-dependent purposes not defined by the 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 architecture defines four types of reserved instructions:

- Instructions in the POWER™ architecture not part of the UISA. For details on POWER architecture incompatibilities and how they are handled by processors built on the PowerPC ISA, see Appendix B, “POWER Architecture Cross Reference,” in *The Programming Environments Manual*.
- Implementation-specific instructions required for the processor to conform to the PowerPC ISA (none of these are implemented in the MPC7410)
- 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 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 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 2-22 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 2-22. Control Registers Synchronization Requirements

Register	Bits	Synchronization Requirements
BAMR	Any	A context synchronizing instruction must follow the **mtspr**.
DABR	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
DBATs	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
EAR	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing register.
HID0	BHT	A context synchronizing instruction must follow the **mtspr**.
	BTIC	
	DPM	
	NAP	
	NHR	
	SLEEP	
	SPD	
	BCLK	A **sync** and context synchronizing instruction must follow a **mtspr.**
	ECLK	
	EMCP	
	EBA	
	EBD	
	PAR	
	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	
	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	
IABR	Any	A context synchronizing instruction must follow a **mtspr**.

Table 2-22. Control Registers Synchronization Requirements (continued)

Register	Bits	Synchronization Requirements
IBATs	Any	A context synchronizing instruction must follow a **mtspr**.
MSR	VEC	A context synchronizing instruction must follow a **mtmsr** instruction.
	FE0	
	FE1	
	FP	
	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.
L2CR, L2PMCR	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 MPC7410—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 MPC7410 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 specifyingHID1 or PVR (read-only registers) executes as a no-op.
- An attempt to access memory that is not available (page fault) causes the ISI or DSI exception handler to be invoked.
- The execution of an **sc** instruction invokes the system call exception handler that permits a program to request the system to perform a service.
- The execution of a trap instruction invokes the program exception trap handler.
- The execution of an instruction that causes a floating-point exception while exceptions are enabled in the MSR invokes the program exception handler.

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

2.3.3 Instruction Set Overview

This section provides a brief overview of the PowerPC instructions implemented in the MPC7410 and highlights any special information with respect to how the MPC7410 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 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 UISA Instructions

The 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 2-23 lists the integer arithmetic instructions defined by the architecture.

Table 2-23. Integer Arithmetic Instructions

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

Table 2-23. Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
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 MPC7410 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 2-24 summarizes the integer compare instructions.

Table 2-24. 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 2-25 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 2-25. 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 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 2-26.

Table 2-26. 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 2-27.

Table 2-27. 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 architecture supports a floating-point system as defined in the IEEE Std. 754, but requires software support to conform with that standard. All floating-point operations conform to the IEEE Std. 754, 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 2-28.

Table 2-28. 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 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 2-29.

Table 2-29. 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 2-30. 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 2-31.

Table 2-31. 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 2-32.

Table 2-32. 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 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 MPC7410, 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 2-33 summarizes the floating-point move instructions.

Table 2-33. 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 Notes—The following describes how the MPC7410 handles misalignment:

The MPC7410 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 MPC7410, TLB reloads are performed transparently and only a page fault causes a restart.

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 MPC7410), 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 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 MPC7410 implementation of integer load instructions:

- The 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 MPC7410; these instructions operate with the same latency as other load instructions.
- The 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 MPC7410. These instructions operate with the same latency as the other load/store instructions.

- The 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 MPC7410.
- The architecture defines the **lwarx** and **stwcx.** as a way to update memory atomically. In the MPC7410, 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), but as with other memory accesses, DSI exceptions can result for other reasons such as a protection violations or page faults.

Table 2-34 summarizes the integer load instructions.

Table 2-34. 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 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 2-35 summarizes the integer store instructions.

Table 2-35. 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 MPC7410 performs store gathering for write-through accesses to nonguarded space or to cache-inhibited stores to nonguarded space if the stores are 4 bytes and they are word-aligned. These stores are combined in the load/store unit (LSU) to form a double word and are sent out on the system bus as a single-beat operation. However, stores can be gathered only if the successive stores that meet the criteria are queued and pending.

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

Store gathering is not performed for the following:

- Stores to guarded cache-inhibited or write-through space
- Byte-reverse store
- **stwcx.** and **ecowx** accesses
- Floating-point stores
- Store operations attempted during a hardware table search
- Store operations in LE = 1 mode

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 2-36 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 2-36. 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 architecture defines the Load Multiple Word (**lmw**) instruction with **r**A in the range of registers to be loaded as an invalid form.

Table 2-37. 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 2-38 summarizes the integer load and store string instructions.

Table 2-38. 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

Table 2-38. Integer Load and Store String Instructions (continued)

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

In the MPC7410 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 MPC7410 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 Notes—The MPC7410 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 MPC7410, 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 architecture defines a load with update instruction with **r**A = 0 as an invalid form. Table 2-39 summarizes the floating-point load instructions.

Table 2-39. 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)

Table 2-39. Floating-Point Load Instructions (continued)

Name	Mnemonic	Syntax
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 2-40 summarizes the floating-point store instructions.

Table 2-40. 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 architecture

Some floating-point store instructions require conversions in the LSU. Table 2-41 shows conversions the LSU makes when executing a Store Floating-Point Single instruction.

Table 2-41. Store Floating-Point Single Behavior

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

Table 2-41. Store Floating-Point Single Behavior (continued)

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

Table 2-42 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 2-42. 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 MPC7410. 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 MPC7410 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 MPC7410, 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 MPC7410, all branch instructions (**b**, **ba**, **bl**, **bla**, **bc**, **bca**, **bcl**, **bcla**, **bclr**, **bclrl**, **bcctr**, **bcctrl**) are executed by the BPU. Some of these instructions can redirect instruction execution conditionally on the value of CR bits. When the CR bits resolve, the branch instruction is either marked as correct or mispredicted. Correcting a mispredicted branch requires that the MPC7410 flush speculatively executed instructions and restore the machine state to immediately after the branch.This correction can be done immediately upon resolution of the condition register bits.

2.3.4.4.2 Branch Instructions

Table 2-43 lists the branch instructions defined by the 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 2-43. 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 2-44, and the Move Condition Register Field (**mcrf**) instruction are also defined as flow control instructions.

Table 2-44. 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 architecture defines these forms of the instructions as invalid.

2.3.4.4.4 Trap Instructions

The trap instructions shown in Table 2-45 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 2-45. 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 2-46 and also Section 2.3.6.1, “System Linkage Instructions—OEA,” for additional information.

Table 2-46. 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 2-47 summarizes the instructions for reading from or writing to the condition register.

Table 2-47. 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 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 MPC7410 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 2-48 lists the **mtspr** and **mfspr** instructions.

Table 2-48. 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 2-49 lists the SPR numbers for user-level SPR accesses.

Encodings for the MPC7410-specific user-level SPRs are listed in Table 2-50.

Table 2-49. User-level 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)	**mfspr**, **mftb**
TBU[2]	269	01000	01101	User (VEA)	**mfspr**, **mftb**
VRSAVE	256	01000	00000	User (AltiVec/UISA)	Both
XER	1	00000	00001	User (UISA)	Both

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

[2] The TB registers are referred to as TBRs rather than SPRs and can be written to using the mtspr instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using either the mftb instruction and specifying TBR 268 for TBL and TBR 269 for TBU.

Table 2-50. User-level SPR Encodings for MPC7410-Defined Registers

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
UBAMR	935	11101	00110	User	**mfspr**
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**

Table 2-50. User-level SPR Encodings for MPC7410-Defined Registers (continued)

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
UPMC4	942	11101	01110	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 3.4.4.4, “Atomic Memory References,” for additional information about these instructions and about related aspects of memory synchronization. See Table 2-51 for a summary.

Table 2-51. 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 MPC7410 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 when the data cache is 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 architecture. In as execution of **lwarx** or **stwcx.** instructions to memory marked write-through or cache-inhibited causes 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 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 MPC7410 encounters one of these invalid instruction forms, it sets CR0 to an undefined value.

2.3.5 VEA Instructions

The 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 and L2 Cache Operation," for more information. Table 2-52 shows the **mftb** instruction.

Table 2-52. 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.Note that the MPC7410 ignores the extended opcode differences between **mftb** and **mfspr** by ignoring bit 25 and treating both instructions identically.

Implementation Note—In the MPC7410, note the following:

- The MPC7410 allows user-mode read access to the time base counter through the use of the Move from Time Base (**mftb**) and the Move from Time Base Upper (**mftbu**) instructions. As a 32-bit implementation of the architecture, the MPC7410 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 and L2 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.

Table 2-53 describes the memory synchronization instructions defined by the VEA.

Table 2-53. 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 cache 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 MPC7410 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 MPC7410 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.

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 and L2 Cache Operation,” for more information about cache topics. The following sections describe how these operations are treated with respect to the MPC7410’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 MPC7410 interprets cache control instructions (**icbi**, **dcbi**, **dcbf**, **dcbz**, and **dcbst**) as if they pertain only to the local L1 and L2 caches. A **dcbz** (with M set) is always broadcast on the bus interface if it does not hit as modified in either on-chip cache.

The MPC7410 always broadcasts an **icbi**. All cache control instructions to direct-store space are no-ops. For information how cache control instructions affect the L2 cache, see 3.7.6, “L2 Cache Operation.”

Table 2-54 summarizes the cache instructions defined by the VEA. Note that these instructions are accessible to user-level programs.

Table 2-54. 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 MPC7410 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 MPC7410 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 intent-to-modify or read-claim, and the data is marked as exclusive in the L1 data cache. More specifically, the following cases occur depending on where the line currently exists or does not exist in the MPC7410. • 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 or recent states, 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 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 or recent states in the L2, a subsequent store will hit on this shared line and incur the associated latency penalties. • dcbtst misses in L1 data cache and L2 caches. In this case, MPC7410 will request the line from memory with intent-to-modify 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 2-54. 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, two beats 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, two beats 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 2-54. 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 always broadcast on the bus (independent of the WIMG setting). **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 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 MPC7410 and are summarized in Table 2-55.

Table 2-55. External Control Instructions

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

The 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 2-56). 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 2-56. 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 MPC7410, 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 2-57 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 2-57. 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	The shadow SRs in the instruction MMU can be read by setting HID0[RISEG] before executing **mfsr**.
Move from Segment Register Indirect	**mfsrin**	**r**D,**r**B	—

The processor control instructions used to access the MSR and the SPRs is discussed in this section. Table 2-58 lists instructions for accessing the MSR.

Table 2-58. 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 2-59.

Table 2-59. 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 2-49. Encodings for MPC7410-specific, supervisor-level SPRs are listed in Table 2-50. Simplified mnemonics are provided for **mtspr** and **mfspr** in Appendix F, “Simplified Mnemonics,” in *The Programming Environments Manual*. For a discussion of context synchronization requirements when altering certain SPRs, refer to Appendix E, “Synchronization Programming Examples,” in *The Programming Environments Manual*.

Table 2-60 lists the SPR numbers for supervisor-level SPR accesses.

Table 2-60. Supervisor-level SPR Encodings

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
DABR[2]	1013	11111	10101	Supervisor (OEA)	Both
DAR	19	00000	10011	Supervisor (OEA)	Both
DBAT0L	537	10000	11001	Supervisor (OEA)	Both
DBAT0U	536	10000	11000	Supervisor (OEA)	Both
DBAT1L	539	10000	11011	Supervisor (OEA)	Both
DBAT1U	538	10000	11010	Supervisor (OEA)	Both
DBAT2L	541	10000	11101	Supervisor (OEA)	Both
DBAT2U	540	10000	11100	Supervisor (OEA)	Both
DBAT3L	543	10000	11111	Supervisor (OEA)	Both
DBAT3U	542	10000	11110	Supervisor (OEA)	Both
DEC	22	00000	10110	Supervisor (OEA)	Both
DSISR	18	00000	10010	Supervisor (OEA)	Both
EAR[2]	282	01000	11010	Supervisor (OEA)	Both
IBAT0L	529	10000	10001	Supervisor (OEA)	Both
IBAT0U	528	10000	10000	Supervisor (OEA)	Both
IBAT1L	531	10000	10011	Supervisor (OEA)	Both
IBAT1U	530	10000	10010	Supervisor (OEA)	Both
IBAT2L	533	10000	10101	Supervisor (OEA)	Both
IBAT2U	532	10000	10100	Supervisor (OEA)	Both

Table 2-60. Supervisor-level SPR Encodings (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
IBAT3L	535	10000	10111	Supervisor (OEA)	Both
IBAT3U	534	10000	10110	Supervisor (OEA)	Both
LDSTDB[3]	1012	11111	10100	Supervisor (OEA)	Both
MMCR0[2]	952	11101	11000	Supervisor	Both
MMCR1[2]	956	11101	11100	Supervisor	Both
PIR	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
SDR1	25	00000	11001	Supervisor (OEA)	Both
SIAR[4]	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 [4]	284	01000	11100	Supervisor (OEA)	**mtspr**
TBU [2]	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 architecture

[3] The LDSTDB is reserved for factory use only. Writing any bits in this register may have boundedly undefined results.

[4] The TB registers are referred to as TBRs rather than SPRs and can be written to using the mtspr instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using either the mftb instruction and specifying TBR 268 for TBL and TBR 269 for TBU.

Encodings for the supervisor-level MPC7410-specific SPRs are listed in Table 2-50.

Table 2-61. Supervisor-level SPR Encodings for MPC7410-Defined Registers

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
BAMR	951	11101	10111	Supervisor	Both
HID0	1008	11111	10000	Supervisor	Both
HID1	1009	11111	10001	Supervisor	**mfspr**
IABR	1010	11111	10010	Supervisor	Both
ICTC	1019	11111	11011	Supervisor	Both
L2CR	1017	11111	11001	Supervisor	Both
L2PMCR	1016	11111	11000	Supervisor	Both
MMCR2	944	11101	10000	Supervisor	Both
MSSCR0	1014	11111	10110	Supervisor	Both
THRM1	1020	11111	11100	Supervisor	Both
THRM2	1021	11111	11101	Supervisor	Both
THRM3	1022	11111	11110	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.

2.3.6.3 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 2-62 lists the only supervisor-level cache management instruction.

Table 2-62. Supervisor-Level Cache Management Instruction

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Invalidate	**dcbi**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations. For cache hits, the cache block is marked I regardless of prior state. A **dcbi** is broadcast if M = 1 (coherency enforced). The instruction acts like a store with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked. The exception priorities (from highest to lowest) for **dcbi** are as follows: 1 BAT protection violation—DSI exception 2 TLB protection violation—DSI exception

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.

See Chapter 7, “Memory Management,” for more information about TLB operations. Table 2-63 summarizes the operation of the TLB instructions in the MPC7410.

Table 2-63. 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.
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 MPC7410. 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. 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 ISA should not assume the existence of mnemonics not described in this document.

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 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 2-64 lists the integer arithmetic instructions defined by the architecture.

Table 2-64. 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 2-64. Vector Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Vector Subtract Unsigned Integer Saturate	**vsububs** **vsubuhs** **vsubuws**	vD,vA,vB
Vector Subtract Signed Integer Saturate	**vsubsbs** **vsubshs** **vsubsws**	vD,vA,vB
Vector Subtract and Write Carry-out Unsigned Word	**vsubcuw**	vD,vA,vB
Vector Multiply Odd Unsigned Integer [b,h] Modulo	**vmuloub** **vmulouh**	vD,vA,vB
Vector Multiply Odd Signed Integer [b,h] Modulo	**vmulosb** **vmulosh**	vD,vA,vB
Vector Multiply Even Unsigned Integer [b,h] Modulo	**vmuleub** **vmuleuh**	vD,vA,vB
Vector Multiply Even Signed Integer [b,h] Modulo	**vmulesb** **vmulesh**	vD,vA,vB
Vector Multiply-High and Add Signed Half-Word Saturate	**vmhaddshs**	vD,vA,vB, vC
Vector Multiply-High Round and Add Signed Half-Word Saturate	**vmhraddshs**	vD,vA,vB,vC
Vector Multiply-Low and Add Unsigned Half-Word Modulo	**vmladduhm**	vD,vA,vB,vC
Vector Multiply-Sum Unsigned Integer [b,h] Modulo	**vmsumubm** **vmsumuhm**	vD,vA,vB,vC
Vector Multiply-Sum Signed Half-Word Saturate	**vmsumshs**	vD,vA,vB,vC
Vector Multiply-Sum Unsigned Half-Word Saturate	**vmsumuhs**	vD,vA,vB,vC
Vector Multiply-Sum Mixed Byte Modulo	**vmsummbm**	vD,vA,vB,vC
Vector Multiply-Sum Signed Half-Word Modulo	**vmsumshm**	vD,vA,vB,vC
Vector Sum Across Signed Word Saturate	**vsumsws**	vD,vA,vB
Vector Sum Across Partial (1/2) Signed Word Saturate	**vsum2sws**	vD,vA,vB
Vector Sum Across Partial (1/4) Unsigned Byte Saturate	**vsum4ubs**	vD,vA,vB
Vector Sum Across Partial (1/4) Signed Integer Saturate	**vsum4sbs** **vsum4shs**	vD,vA,vB
Vector Average Unsigned Integer	**vavgub** **vavguh** **vavguw**	vD,vA,vB
Vector Average Signed Integer	**vavgsb** **vavgsh** **vavgsw**	vD,vA,vB
Vector Maximum Unsigned Integer	**vmaxub** **vmaxuh** **vmaxuw**	vD,vA,vB

Table 2-64. 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 2-66) can optionally set the CR6 field of the 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 2-65.

Table 2-65. 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 2-66 summarizes the vector integer compare instructions.

Table 2-66. Vector Integer Compare Instructions

Name	Mnemonic	Syntax
Vector Compare Greater than Unsigned Integer	**vcmpgtub[.]** **vcmpgtuh[.]** **vcmpgtuw[.]**	**v**D,**v**A,**v**B

Table 2-66. Vector Integer Compare Instructions (continued)

Name	Mnemonic	Syntax
Vector Compare Greater than Signed Integer	**vcmpgtsb[.]** **vcmpgtsh[.]** **vcmpgtsw[.]**	vD,vA,vB
Vector Compare Equal to Unsigned Integer	**vcmpequb[.]** **vcmpequh[.]** **vcmpequw[.]**	vD,vA,vB

2.5.1.3 Vector Integer Logical Instructions

The vector integer logical instructions shown in Table 2-67 perform bit-parallel operations on the operands.

Table 2-67. Vector Integer Logical Instructions

Name	Mnemonic	Syntax
Vector Logical AND	**vand**	vD,vA,vB
Vector Logical OR	**vor**	vD,vA,vB
Vector Logical XOR	**vxor**	vD,vA,vB
Vector Logical AND with Complement	**vandc**	vD,vA,vB
Vector Logical NOR	**vnor**	vD,vA,vB

2.5.1.4 Vector Integer Rotate and Shift Instructions

The vector integer rotate instructions are summarized in Table 2-68.

Table 2-68. Vector Integer Rotate Instructions

Name	Mnemonic	Syntax
Vector Rotate Left Integer	**vrlb** **vrlh** **vrlw**	vD,vA,vB

The vector integer shift instructions are summarized in Table 2-69.

Table 2-69. Vector Integer Shift Instructions

Name	Mnemonic	Syntax
Vector Shift Left Integer	**vslb** **vslh** **vslw**	vD,vA,vB

Table 2-69. Vector Integer Shift Instructions

Name	Mnemonic	Syntax
Vector Shift Right Integer	**vsrb** **vsrh** **vsrw**	vD,vA,vB
Vector Shift Right Algebraic Integer	**vsrab** **vsrah** **vsraw**	vD,vA,vB

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 Std.754 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 2-70.

Table 2-70. Vector Floating-Point Arithmetic Instructions

Name	Mnemonic	Syntax
Vector Add Floating-Point	**vaddfp**	vD,vA,vB
Vector Subtract Floating-Point	**vsubfp**	vD,vA,vB
Vector Maximum Floating-Point	**vmaxfp**	vD,vA,vB
Vector Minimum Floating-Point	**vminfp**	vD,vA,vB

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 2-71.

Table 2-71. 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 2-72.

Table 2-72. 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
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 2-73.

Table 2-73. 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 2-74.

Table 2-74. 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 architecture; see “Byte Ordering,” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual* for information about little-endian byte ordering.

Table 2-75 summarizes the vector load instructions.

Table 2-75. 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 MPC7410, 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 2-76 summarizes the vector alignment instructions.

Table 2-76. 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 2-77 provides a summary of the vector store instructions.

Table 2-77. 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 MPC7410, 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 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 2-78 describes the vector pack instructions.

Table 2-78. 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
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 2-79 describes the unpack instructions.

Table 2-79. 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 2-80 describes the merge instructions.

Table 2-80. 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 2-81. Vector Splat Instructions

Name	Mnemonic	Syntax
Vector Splat Integer	**vspltb** **vsplth** **vspltw**	vD, vB, UIMM
Vector Splat Immediate Signed Integer	**vspltisb** **vspltish** **vspltisw**	vD, 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 2-78 describes the vector permute instruction.

Table 2-82. Vector Permute Instruction

Name	Mnemonic	Syntax
Vector Permute	**vperm**	vD, vA,vB,vC

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 2-83 describes the **vsel** instruction.

Table 2-83. Vector Select Instruction

Name	Mnemonic	Syntax
Vector Select	**vsel**	vD,vA,vB,vC

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 ≤ count ≤ 127). Of these 7 bits, the high-order

4 bits give the number of complete bytes by which to shift and are used by **vslo** and **vsro**; the low-order 3 bits give the number of remaining bits by which to shift and are used by **vsl** and **vsr**.

Table 2-84 describes the vector shift instructions.

Table 2-84. 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 2-85 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 2-85. 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 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 and L2 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 MPC7410 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 and L2 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 2-86 summarizes the directed prefetch cache instructions defined by the AltiVec VEA. Note that these instructions are accessible to user-level programs.

Table 2-86. 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 MPC7410
Data Stream Touch for Store Transient	**dststt**	**r**A,**r**B,STRM	Not recommended for use in MPC7410
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 MPC7410

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 MPC7410 implements these instructions as suggested. They follow all the cache allocation and replacement policies described in Section 3.6, "Cache Operations," but they leave their addressed cache entries in the LRU state. In addition, all LRU instructions are also interpreted to be transient and are also treated as described in Section 7.1.2.2, "Transient Instructions and Caches."

Chapter 3
L1 and L2 Cache Operation

The MPC7410 microprocessor contains separate 32-Kbyte, eight-way set associative level 1 (L1) instruction and data caches to allow the execution units and registers rapid access to instructions and data. In addition, the MPC7410 microprocessor features an integrated level 2 (L2 cache) cache controller.

This chapter describes the organization of the on-chip L1 instruction and data caches, cache coherency protocols, cache control instructions, various cache operations, the L2 cache controller, and the interaction between the caches, the load/store unit (LSU), the instruction unit, the memory subsystem, and the bus interface unit (BIU).

Note that in this chapter, the term ‘multiprocessor’ is used in the context of maintaining cache coherency. These multiprocessor devices could be actual processors or other devices that can access system memory, maintain their own caches, and function as bus masters requiring cache coherency.

AltiVec Technology and the Cache Implementation

The implementation of AltiVec technology in the MPC7410 has implications that affect the cache model, specifically:

- AltiVec transient instructions (**dstt**, **dststt**, **lvxl**, **stvxl**), described in Section 3.4.2.1, “AltiVec Transient Hint Support”
- Store miss merging, described in Section 3.6.5, “Store Miss Merging”
- AltiVec LRU instructions (**lvxl**, **stvxl**), described in Section 3.6.8.1, “AltiVec LRU Instruction Support”
- External system bus transactions caused by caching-inhibited AltiVec loads and stores, or write-through AltiVec stores, described in Section 3.9, “Caches and System Bus Transactions”

3.1 L1 Instruction and Data Caches

The MPC7410 L1 cache implementation has the following characteristics:

- Two separate 32-Kbyte instruction and data caches (Harvard architecture).
- Both instruction and data caches are eight-way set associative.
- The cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.
- Both the instruction and data caches have 32-byte cache blocks. A cache block is the block of memory that a coherency state describes, also referred to as a cache line.
- Six status bits for each data cache block allow encoding for coherency and victimization, as follows:
  - Castout (C)

    - Dirty (D)
    - Modified (M)
    - Recent (R)
    - Shared (S)
    - Valid (V)
- A single coherency status bit for each instruction cache block allows encoding for the following two possible states:
    - Invalid (INV)
    - Valid (VAL)
- The MPC7410 supports a five- (MERSI) modified/exclusive/recent/shared/invalid, four- (MESI), or three-state (MEI) coherency protocol.
- The L1 data cache supports load-miss folding.
- The L1 data cache supports store-miss merging.
- Each cache can be invalidated or locked by setting the appropriate bits in the hardware implementation-dependent register 0 (HID0), a special-purpose register (SPR) specific to the MPC7410.
- The caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each set. The caches also support AltiVec LRU instructions.

The MPC7410 supports a fully-coherent 4-Gbyte physical memory address space. Bus snooping is used to ensure the coherency of global memory with respect to the data cache.

On a cache miss, cache blocks are filled in four beats of 64 bits each. The burst fill is performed as a critical-double-word-first operation.

For the instruction cache, the critical double word is simultaneously written to the cache and forwarded to the instruction queue, thus minimizing stalls due to cache fill latency. The instruction cache is not blocked to internal accesses while a load completes, providing for hits under misses.

For the data cache, an entire cache block is collected in a reload buffer before being loaded into the cache. This allows the data cache to service multiple outstanding misses while at the same time staying available to subsequent load and store hits.

The instruction and data caches are integrated into the MPC7410 as shown in Figure 3-1.

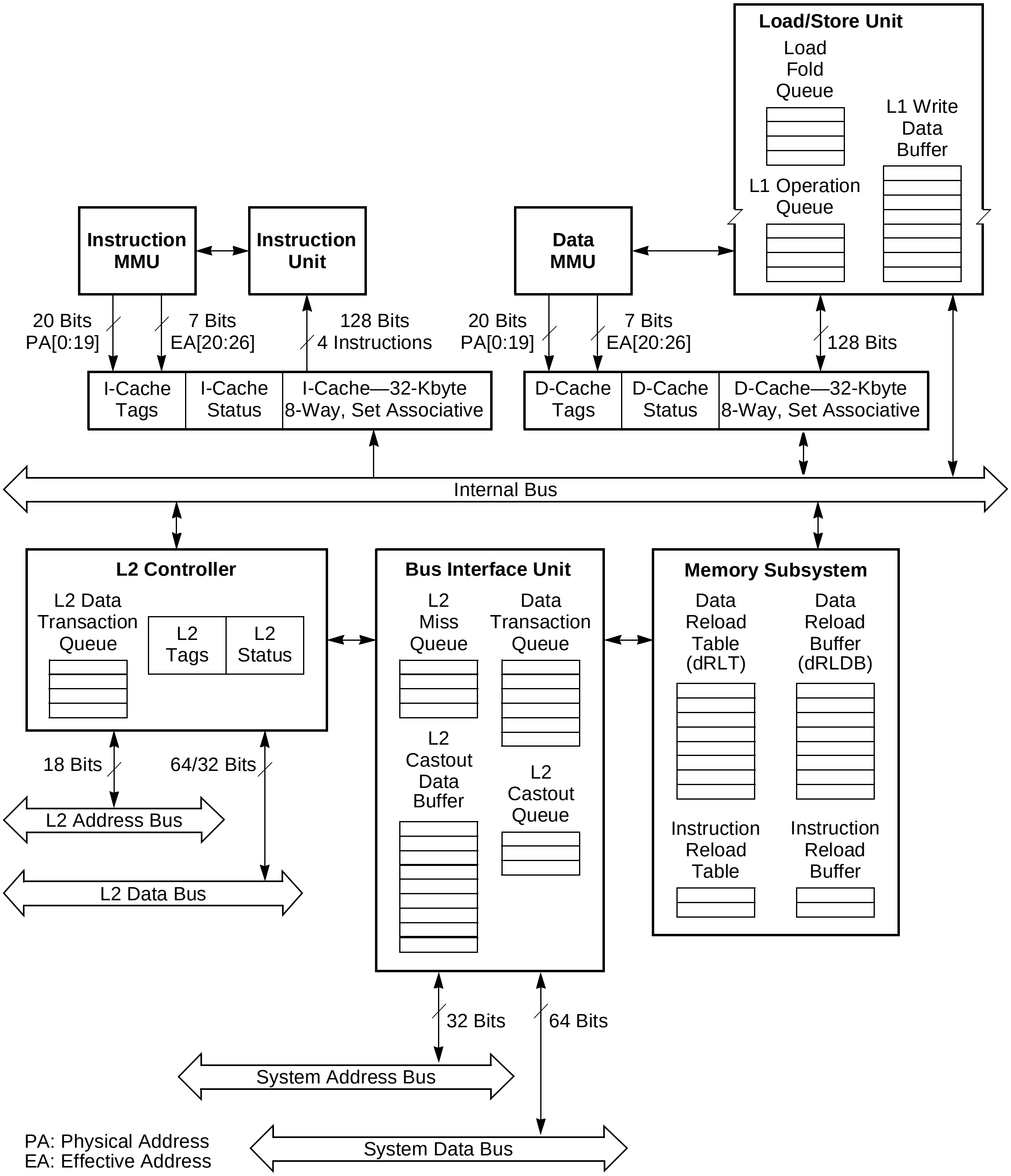


Figure 3-1. Cache/Memory Subsystem/BIU Integration

Both caches are tightly coupled to the MPC7410’s L2 cache controller and bus interface unit to allow efficient access to the L2 cache or the system memory controller and other bus masters. The bus interface unit receives requests for bus operations from the instruction and data caches, and executes the operations per the 60x or MPX bus protocol. The BIU provides address queues, prioritizing logic, and bus control logic. The BIU captures snoop addresses for data cache, address queue, and memory reservation (**lwarx** and **stwcx.**) operations.

The memory subsystem provides an eight-entry data reload table (dRLT) and an associated eight-entry data reload buffer (dRLDB) for performing loads and store reloads and store miss merging. A four-entry load fold queue (LFQ) holds consecutive load misses to outstanding load miss operations. A four-entry L1 operation queue (L1OPQ) holds outstanding cache operations, cast-outs, and caching-inhibited or caching- allowed/write-through stores. An eight-entry L1 write data buffer holds data for cast-outs and caching-inhibited or caching-allowed/write-through stores. A two-entry instruction reload table (iRLT) and an associated two-entry instruction reload buffer (iRLDB) performs instruction cache miss reloads and holds the instruction until it is reloaded into the L2 cache.

The data cache supplies data to the general-purpose registers (GPRs), floating-point registers (FPRs), and vector registers (VRs) by means of the load/store unit (LSU). The MPC7410’s LSU is directly coupled to the data cache to allow efficient movement of data to and from the GPRs, FPRs, and VRs. The LSU provides all logic required to calculate effective addresses, handles data alignment to and from the data cache, and provides sequencing for load and store string and multiple operations. Write operations to the data cache can be performed on a byte, half-word, word, double-word, or quad-word basis.

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

3.2 Data Cache Organization

The data cache is organized as 128 sets of eight blocks as shown in Figure 3-2.

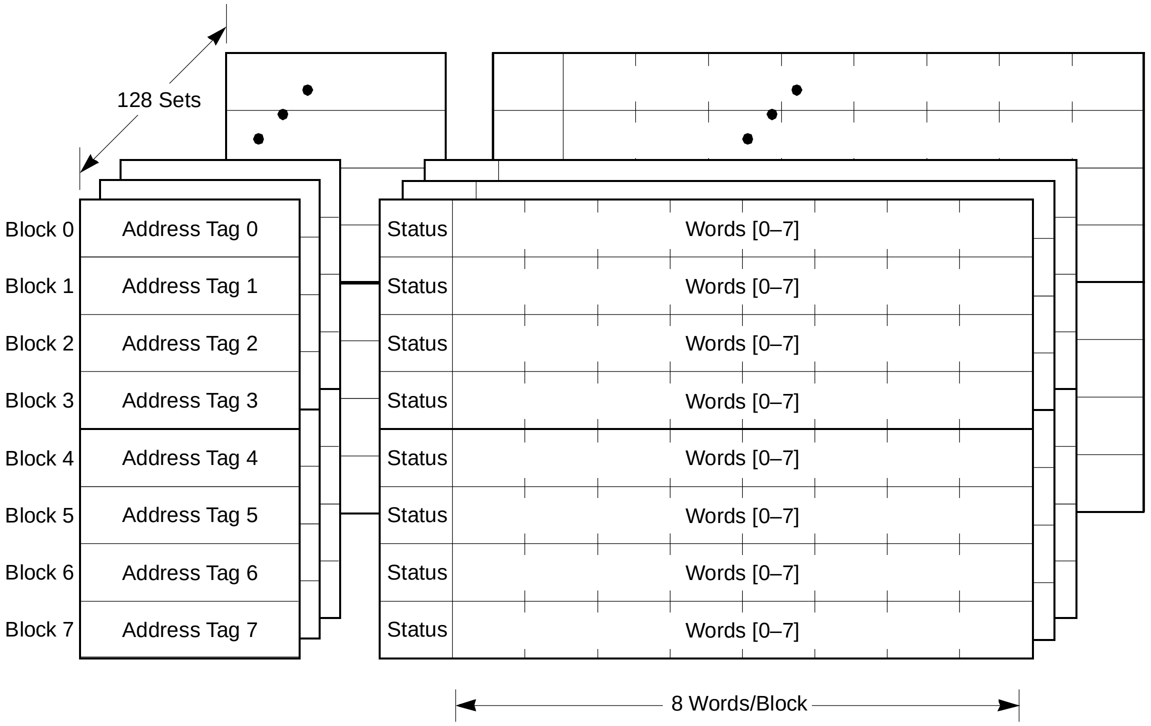


Figure 3-2. Data Cache Organization

Each block consists of 32 bytes of data, six status bits, and an address tag. Note that in the architecture, the term 'cache block,' or simply 'block,' when used in the context of cache implementations, refers to the unit of memory at which coherency is maintained. For the MPC7410, this is the 32-byte cache line. This value may be different for other implementations.

Each cache block contains eight contiguous words from memory that are loaded from an eight-word boundary (that is, bits A[27:31] of the logical (effective) addresses are zero); as a result, cache blocks are aligned with page boundaries. Address bits A[20:26] provide the index to select a cache set. The tags consist of physical address bits PA[0:19]. Address translation occurs in parallel with set selection (from A[20:26]). The data cache tags are dual-ported and non-blocking, for efficient load/store and snooping operations. Logical address bits A[27:31] locate a byte within the selected block.

There are six status bits associated with each cache block. These bits are used to implement the modified/exclusive/recent/shared/invalid (MERSI), MESI, and MEI cache coherency protocols and to support the AltiVec transient instructions. The coherency protocols are described in Section 3.4, "Memory and Cache Coherency."

3.3 Instruction Cache Organization

The instruction cache also consists of 128 sets of eight blocks, as shown in Figure 3-3.

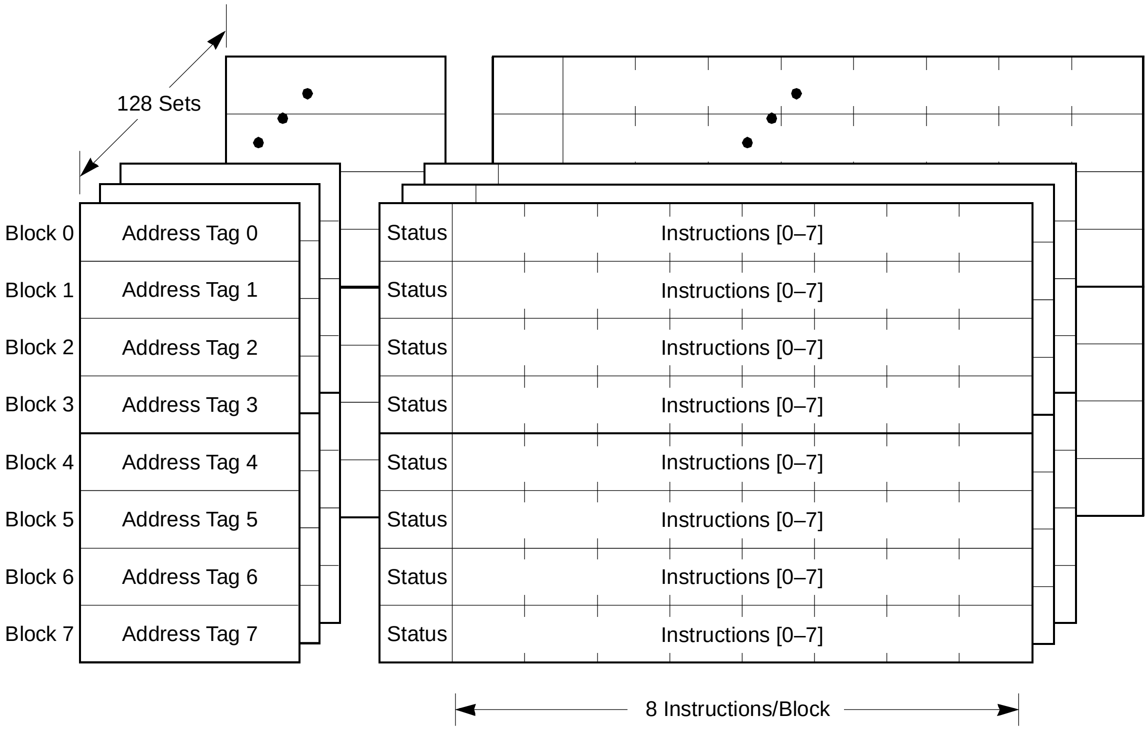


Figure 3-3. Instruction Cache Organization

Each block consists of 8 instructions, a single status bit, and an address tag. As with the data cache, each instruction cache block is loaded from an eight-word boundary (that is, bits A[27:31] of the logical (effective) addresses are zero); as a result, cache blocks are aligned with page boundaries. Also, address bits A[20:26] provide the index to select a set, and bits A[27:29] select an instruction within a block. The tags consist of bits PA[0:19]. Address translation occurs in parallel with set selection (from A[20:26]).

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

```
dcbst    # update memory
sync     # wait for update
icbi     # remove (invalidate) copy in instruction cache
sync     # wait for ICBI operation to be globally performed
isync    # remove copy in own instruction buffer
```

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

3.4 Memory and Cache Coherency

The primary objective of a coherent memory system is to provide the same image of memory to all devices using the system. Coherency allows synchronization and cooperative use of shared resources. Otherwise, multiple copies of a memory location, some containing stale values, could exist in a system resulting in errors when the stale values are used. Each potential bus master must follow rules for managing the state of its cache. This section describes the coherency mechanisms of the architecture and the cache coherency protocols that the MPC7410 data cache supports.

Note that unless specifically noted, the discussion of coherency in this section applies to the MPC7410's data cache only. The instruction cache is not snooped. Instruction cache coherency must be maintained by software. However, the MPC7410 does support a fast instruction cache invalidate capability as described in Section 3.5.1.6, "Instruction Cache Flash Invalidation."

3.4.1 Memory/Cache Access Attributes (WIMG Bits)

Some memory characteristics can be set on either a memory management block or page basis by using the WIMG bits in the BAT registers or page table entries (PTE), respectively. These bits allow both uniprocessor and multiprocessor system designs to exploit numerous system-level performance optimizations. The WIMG attributes control the following functionality:

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

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

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

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

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

Software must exercise care with respect to the use of these bits if coherent memory support is desired. Careless specification of these bits may create situations that present coherency paradoxes to the processor. These coherency paradoxes can occur within a single processor or across several processors. It is important to note that in the presence of a paradox, the operating system software is responsible for correctness.

In particular, a coherency paradox can occur when the state of these bits is changed without appropriate precautions (such as flushing the pages that correspond to the changed bits from the caches of all processors in the system) or when the address translations of aliased real addresses specify different values for certain WIMG bit values. The MPC7410 supports aliasing for WIMG = 100x and WIMG = 000x; however, the MPC7410 does not support aliasing WIMG = 101x and WIMG = 001x. Specifically, this means that for a given physical address, the MPC7410 only supports simultaneous memory/cache access attributes for that physical address of caching-allowed, write-through, memory-coherency-not-required (WIMG = 100x) and caching-allowed, write-back, memory-coherency-not-required (WIMG = 000x).

For real addressing mode (that is, for accesses performed with address translation disabled—MSR[IR] = 0 or MSR[DR] = 0 for instruction or data access, respectively), the WIMG bits are automatically generated as 0b0011 (all memory is write-back, caching-allowed, memory-coherency-required, and guarded).

3.4.1.1 Out-of-Order Accesses to Guarded Memory

Guarded memory may be accessed out of order if the load is guaranteed to be executed. In this case, the entire cache block containing the referenced data may be loaded into the cache.

In addition, out-of-order accesses to non-guarded space (G = 0), from both the instruction and data caches, can be disabled by setting speculative access disable bit, HID0[SPD].

For the MPC7410, a guarded load is not allowed to access the system interface until that load is at the bottom of the completion buffer. This means that all prior load accesses to the system interface must have already returned data to the processor before the subsequent guarded load is allowed to access the system address bus. This prevents the MPC7410 from pipelining a guarded load with any other type of load on the system interface. Note that this has a large negative effect on load miss bandwidth performance. For this reason, it is not recommended to have guarded loads in code streams that require high system bandwidth utilization.

3.4.2 Coherency Support

The MPC7410 provides full hardware support for cache coherency and ordering instructions (**dcbz**, **dcbi**, **dcbf**, **sync**, **icbi**, and **eieio**) and full hardware implementation of the TLB management instructions (**tlbie**, and **tlbsync**). Snooping, described in Section 3.9.3, “Snooping,” is integral to the memory subsystem design and operation. The MPC7410 is self-snooping and can $\overline{\text{ARTRY}}$ its own **tlbie**, **tlbsync**, **icbi**, and **sync** broadcasts.

Each 32-byte cache block in the data cache contains 6 status bits (CDMRSV). The MPC7410 uses these bits to support the coherency protocols and to direct castout and reload operations. The L1 data cache status bits and the conditions that cause them to be set or cleared are defined in Table 3-1.

Table 3-1. Data Cache Status Bits

Status Bit	Name	Meaning	Set Conditions	Clear Conditions
C	Castout	The cache block should be castout from the L1 data cache to the L2 cache when selected for replacement	Non-transient reload from BIU	Transient hit
D	Dirty	The cache block has been stored to since it was reloaded into the L1 data cache	Store miss reload from BIU or L2 Write-back store hit on ¬S & ¬R	**dcbst** hit
M	Modified	The cache block is modified with respect to the external system interface	Store miss reload from BIU or L2 Write-back store hit on ¬S & ¬R	**dcbst** hit Snoop clean hit Snoop read hit
R	Recent	This is the most recent processor to perform a read transaction to the cache block while other processors have a shared copy	Load miss reload from BIU with $\overline{\text{SHD}}$ response Load miss reload from L2 cache with L2 cache status = R	Snoop read hit
S	Shared	The cache block is shared with other processors and is read-only	Load miss reload from BIU with $\overline{\text{SHD}}$ response Load miss reload from L2 cache with L2 cache status = R or S	None
V	Valid	The cache block is valid	Reload from BIU or L2 cache	**dcbi**, **dcbf** hit Write-back store hit to R or S (see Section 3.6.6, “Store Hit to a Data Cache Block Marked Recent or Shared,”) **dcbz**, **dcba** hit (see Section 3.5.3.3, “Data Cache Block Zero (dcbz)) snoop invalidate hit

Every L1 data cache block’s state is defined by its CDMRSV status bits. Table 3-2 describes the allowed states for the status bits.

Table 3-2. Allowed Data Cache States

CDMRSV value						Extended State	MERSI state	Comments
C	D	M	R	S	V			
x	x	x	x	x	0	I	I	Invalid line
1	0	0	0	0	1	EC	E	Reload from BIU, or **dcbst** hit on MCD

Table 3-2. Allowed Data Cache States (continued)

CDMRSV value						Extended State	MERSI state	Comments
C	D	M	R	S	V			
0	0	0	0	0	1	E	E	Load miss reload from L2 cache, or transient load miss reload from BIU, or transient load hit
1	1	0	0	0	1	ECD	E	Snooped clean hit on MCD, caused push
0	1	0	0	0	1	ED	E	Snooped clean hit on MD, caused push
1	0	0	0	1	1	SC	S	Load miss reload from BIU
0	0	0	0	1	1	S	S	Load miss reload from L2 cache, or transient load hit
1	1	0	0	1	1	SCD	S	Snooped read hit on MCD, caused push
0	1	0	0	1	1	SD	S	Snooped read hit on MD, caused push
1	0	0	1	1	1	RC	R	Reload from BIU
0	0	0	1	1	1	R	R	Reload from L2 cache, or transient load miss reload from BIU, or transient hit
0	0	1	0	0	1	M	M	Load miss reload from L2 cache
1	1	1	0	0	1	MCD	M	Store hit on E or M, or caching-allowed store miss reloaded from BIU
0	1	1	0	0	1	MD	M	Store hit on E or M after reloading from L2 cache, or after a transient hit

Note that any state not shown in Table 3-2 is not allowed. Also note that any valid line with either the C or D bit set is cast out from the data cache when it is selected for replacement.

3.4.2.1 AltiVec Transient Hint Support

The C status bit in the data cache tags may be cleared if a transient type access hits in the data cache. In addition, the C bit is not set upon reload from the BIU if the miss is a transient type access. The **dstt**, **dststt**, **lvxl**, and **stvxl** instructions are considered to be transient.

3.4.3 Coherency Protocols

When configured for either MPX bus or 60x bus modes, the MPC7410 can be configured to support a four-state MESI protocol (similar to the MPC604-family microprocessors) or a three-state MEI protocol (similar to the MPC603- and 750-family microprocessors).

When configured for MPX bus mode, the MPC7410 supports an additional five-state cache coherency protocol, referred to as the MERSI protocol. The additional state in this protocol is the recent state. This state is used for shared data intervention. It indicates that a cache block is shared and is the most recently read version of the data. A cache block is placed in the R state when it is loaded after a shared snoop response was detected. The cache block is downgraded to the S state when another snoop read access for this line is performed. The cache block in the recent state is the one used to supply intervention data. This ensures that only one processor supplies data for intervention. The MERSI coherency protocol together with the MPX bus protocol allows for data-only intervention between caches.

The MESI or MEI coherency protocol is selected by the MSSCR0[SHDEN] parameter. SHDEN = 0b1 indicates that the MPC7410 uses the shared state and follows the MESI protocol. SHDEN = 0b0 indicates that MPC7410 does not use the shared state and follows the MEI protocol. The MERSI protocol is a superset of the MESI protocol requiring SHDEN = 1. The MERSI coherency protocol is selected by enabling full L1 intervention in MSSCR0 (L1_INTVEN = 0b111) when SHDEN = 0b1.

Table 3-3 summarizes the coherency protocols and intervention supported in 60x bus mode (MSSCR0[EMODE] = 0b0). The intervention types are described in Table 3-6.

Table 3-3. Coherency Protocols in 60x Bus Mode

Coherency Protocol	SHDEN	Intervention Type[1]	L1_INTVEN
MEI	0	Window-of-opportunity for hits on modified	N/A
MESI	1	Window-of-opportunity for hits on modified	N/A

[1] See Section 3.4.3.2, “Intervention,” for information about Intervention types.

Note that L1_INTVEN is only recognized when the MPC7410 is configured for MPX bus mode.

Table 3-4 summarizes the coherency protocols and interventions supported in MPX bus mode (MSSCR0[EMODE] = 0b1). The intervention types are described in Table 3-6.

Table 3-4. Coherency Protocols in MPX Bus Mode

Coherency Protocol	SHDEN	Intervention Type[1]	L1_INTVEN		
			MI[2]	EI[3]	SI[4]
MEI	0	Window-of-opportunity for hits on modified	0	0	0
		Cache-to-cache/window-of-opportunity for hits on modified	1	0	0
		Cache-to-cache/window-of-opportunity for hits on modified Cache-to-cache for hits on exclusive	1	1	x
MESI	1	Window-of-opportunity for hits on modified	0	0	0
		Cache-to-cache/window-of-opportunity for hits on modified	1	0	0
		Cache-to-cache/window-of-opportunity for hits on modified Cache-to-cache for hits on exclusive	1	1	0
MERSI	1	Cache-to-cache/window-of-opportunity for hits on modified Cache-to-cache for hits on exclusive and recent	1	1	1

[1] See Section 3.4.3.2, “Intervention,” for information about Intervention types

[2] MI is the modified intervention enable bit in L1_INTVEN

[3] EI is the exclusive intervention enable bit in L1_INTVEN

[4] SI is the shared intervention enable bit in L1_INTVEN

Note that the snoop intervention when L1_INTVEN = 0b000 is the same as that for 60x bus mode. Also note that when SHDEN = 0b0, the SI bit of the L1_INTVEN parameter has no effect (that is, when cleared, the SHDEN parameter overrides the SI bit).

3.4.3.1 Snoop Response

Table 3-5 describes the snoop responses used by the MPC7410. See Chapter 8, “Signal Descriptions,” and Chapter 9, “System Interface Operation,” for detailed signal timing and bus protocol information.

Table 3-5. Snoop Response Summary

Snoop Response	State Transition Diagram Symbol	Description
No response	— (no symbol)	The processor does not contain any memory at the snooped address or the coherency protocol does not require a response. The snoop has been fully serviced and no internal pipeline collisions occurred that would require a busy response.
$\overline{\text{SHD}}$ asserted	S	The processor contains data from the snooped address or a reservation on the snooped address.
$\overline{\text{ARTRY}}$ asserted	A	The processor cannot service the snoop due to an internal pipeline collision (busy). The same address tenure must be re-run at a later time.
$\overline{\text{ARTRY}}$ and $\overline{\text{SHD}}$ asserted	AS	The processor contains a modified copy of data from the snooped address and is prepared to perform a window-of-opportunity (W) snoop push.
$\overline{\text{HIT}}$ asserted for one cycle	H1 (MPX bus mode only)	The processor contains a modified copy of data from the snooped address and is prepared to perform cache-to-cache/window-of-opportunity (CW) intervention.
$\overline{\text{HIT}}$ asserted for two cycles	H2 (MPX bus mode only)	The processor contains an exclusive or recent copy of data from the snooped address and is prepared to perform cache-to-cache (C) intervention. This is an optional extended meaning of $\overline{\text{HIT}}$ response that indicates that data snarfing by the system is not necessary.

3.4.3.2 Intervention

Table 3-6 briefly describes the intervention types used by the MPC7410. See Chapter 9, "System Interface Operation," for signaling protocol information for each intervention type.

Table 3-6. Snoop Intervention Summary

Intervention Type	State Transition Diagram Symbol	Description
No intervention	— (No symbol)	The processor does not contain any memory at the snooped address or the coherency protocol does not require intervention.
Window-of-opportunity	W	Window-of-opportunity snoop push for hits on modified data. The processor performs a write-with-kill, snoop-push transaction in the next address tenure. The MPC7410 asserts $\overline{\text{BR}}$ in the window of opportunity to initiate the snoop push operation. The window of opportunity is defined as the second cycle after an $\overline{\text{AACK}}$ that has been $\overline{\text{ARTRY}}$ed. Only the intervening master can assert $\overline{\text{BR}}$ in the window of opportunity. When a master asserts $\overline{\text{BR}}$ in the window of opportunity, it uses it to perform a snoop push (write-with-kill) to the most previous snoop address (unless the master still has a write-with-kill pending due to a previous window-of-opportunity request that is not yet satisfied). The MPC7410 always presents a cache-block aligned address (that is, A[27–31] = 0b0_0000) for every window-of-opportunity snoop push.
Cache-to-cache/ window-of-opportunity	CW (MPX bus mode only)	Cache-to-cache intervention or window-of-opportunity snoop push for hits on modified data. The processor has queued up a data-only write transaction to provide data to the snooping master (cache-to-cache intervention). If another master asserts $\overline{\text{ARTRY}}$ coincident with the assertion of $\overline{\text{HIT}}$, the MPC7410 cancels the queued-up data-only write transaction and asserts $\overline{\text{BR}}$ in the window of opportunity to perform a write-with-kill, snoop push in the next address tenure (window-of-opportunity snoop push).
Cache-to-cache	C (MPX bus mode only)	Cache-to-cache intervention for hits on exclusive or shared data. The processor has queued up a data-only write transaction to provide data to the snooping master (cache-to-cache intervention). If another master asserts $\overline{\text{ARTRY}}$ coincident with the assertion of $\overline{\text{HIT}}$, the MPC7410 cancels the queued-up data-only transaction but does not attempt to perform a snoop push. The cache block state is already changed to the new state due to the snoop. Thus, the intervening processor (the one that asserted $\overline{\text{HIT}}$) does not contain the cache block in a state suitable for intervention when the retried snoop transaction is rerun on the bus.

3.4.3.3 Simplified Transaction Types

For the purposes of snooping bus transactions, the MPC7410 treats related (but distinct) transaction types as a single simplified transaction type. Table 3-7 defines the mapping of simplified transaction types to actual transaction types.

Table 3-7. Simplified Transaction Types

Simplified Transaction Type	Actual Transaction Type MESI or MERSI Protocol (SHDEN = 1)	Actual Transaction Type MEI Protocol (SHDEN = 0)
Read	Read Read-atomic	—
RWITM	RWITM RWITM-atomic RCLAIM	Read Read-atomic RWITM RWITM-atomic RCLAIM
RWNITC	RWNITC—Acts like a read transaction for snoop response purposes; acts like a clean transaction for MESI state change purposes.	RWNITC—Acts like a RWITM transaction for snoop response purposes; acts like a clean transaction for MEI state change purposes.
Write	Write-with-flush Write-with-flush-atomic	
Flush	Flush	
Clean	Clean	
Kill	Kill Write-with-kill	
Reskill (Used for reservation snooping only)	RWITM RWITM-atomic RCLAIM Write-with-flush Write-with-flush-atomic Kill Write-with-kill	

Note that when SHDEN = 0b0, the MPC7410 snoops read transactions as if they were RWITM transactions. Also when SHDEN = 0b0, any MPC7410-initiated read transaction that generates a $\overline{\text{SHD}}$-assertion response is treated as an invalidate operation.

In the following state transition diagrams, RWNITC is not explicitly shown. For state transitions (for example, modified to exclusive) RWNITC is treated like a clean operation. For intervention purposes (for example a W or H intervention) RWNITC is treated like a read operation.

3.4.3.4 MESI State Transitions

In the following state transition diagrams, all snooped transactions are assumed to be global ($\overline{\text{GBL}}$ asserted), caching-allowed ($\overline{\text{CI}}$ negated), and write-back ($\overline{\text{WT}}$ negated). If either $\overline{\text{CI}}$ or $\overline{\text{WT}}$ is asserted, then the state transitions remain the same, but no data intervention occurs. Instead, a window-of-opportunity snoop push is performed only for snoop hits to modified cache blocks.

The state diagrams use symbols on the transition lines for snoop response and intervention type. For example, H1S-CW would denote a one-cycle $\overline{\text{HIT}}$ and $\overline{\text{SHD}}$ asserted snoop response and a cache-to-cache/window-of-opportunity intervention type. See Table 3-5 and Table 3-6 for the symbols used in the state diagrams.

3.4.3.4.1 MESI Protocol in 60x Bus Mode and MPX Bus Mode (with L1_INTVEN = 0b000)

The following state diagrams show the MESI state transitions when the MPC7410 is configured for 60x bus mode and for MPX bus mode when hit intervention is disabled (L1_INTVEN = 0b000).

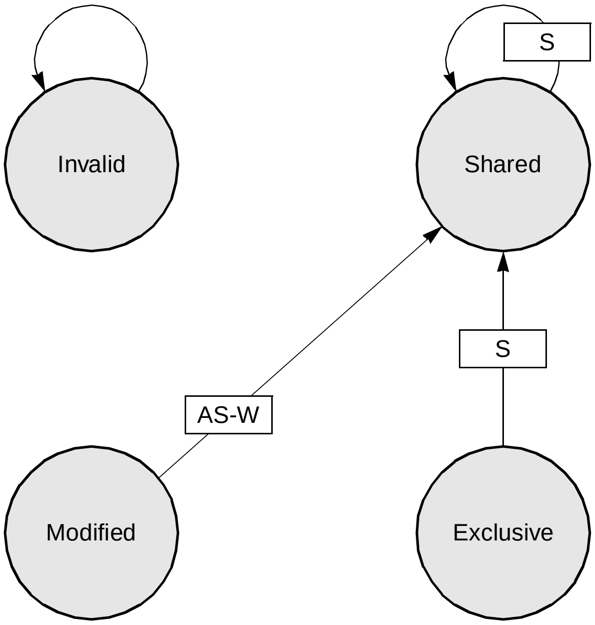


Figure 3-4. Read Transaction—60x and MPX Bus Modes, L1_INTVEN = 0b000

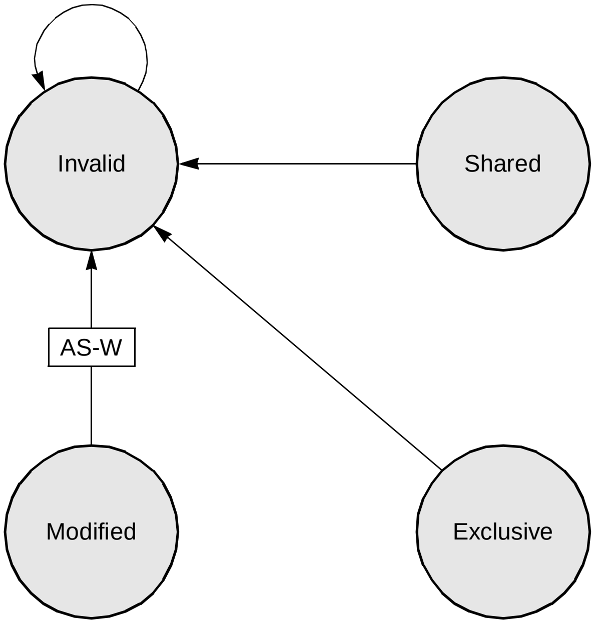


Figure 3-5. RWITM, Write, and Flush Transactions—60x and MPX Bus Modes, L1_INTVEN = 0b000

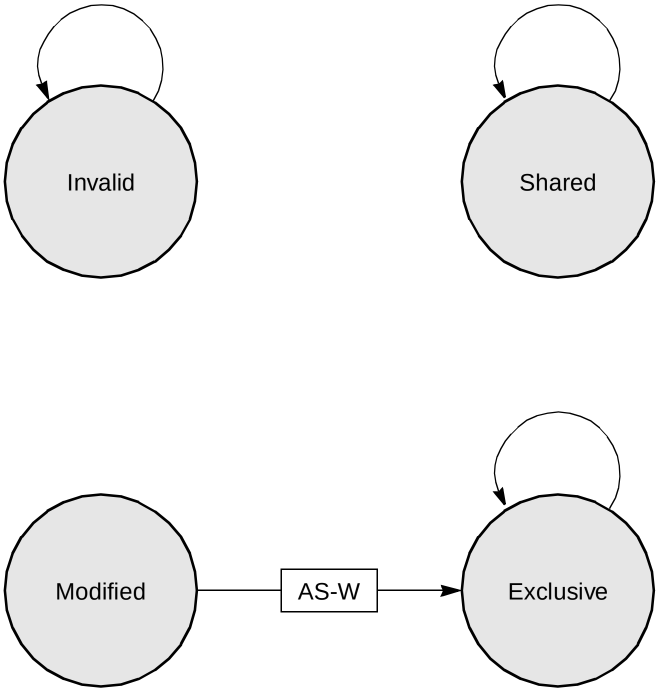
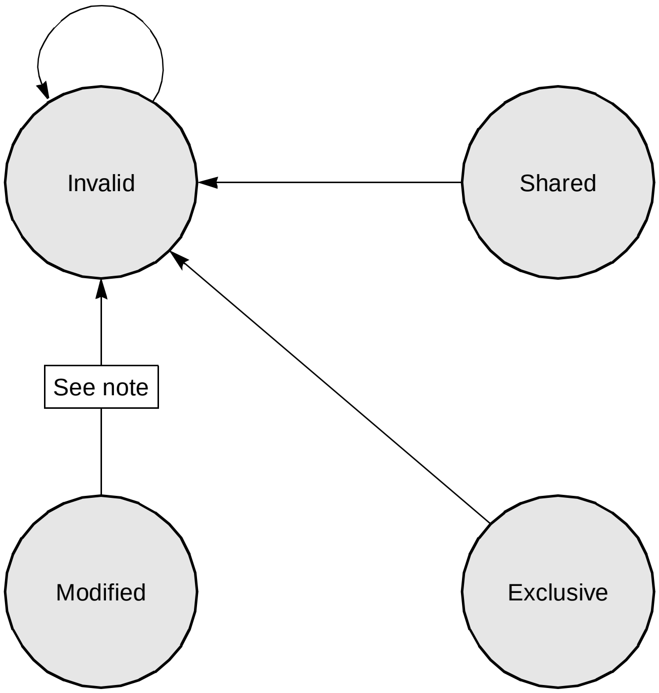


Figure 3-6. Clean Transaction—60x and MPX Bus Modes, L1_INTVEN = 0b000



Note: If another master asserts $\overline{\text{ARTRY}}$, the MPC7410 performs a window-of-opportunity style push. Otherwise, there is no intervention.

Figure 3-7. Kill Transaction—60x and MPX Bus Modes, L1_INTVEN = 0b000

3.4.3.4.2 MESI Protocol in MPX Bus Mode with Modified Intervention Enabled

The following state diagrams show the MESI state transitions when the MPC7410 is configured for MPX bus mode with only modified intervention enabled (L1_INTVEN = 0b100).

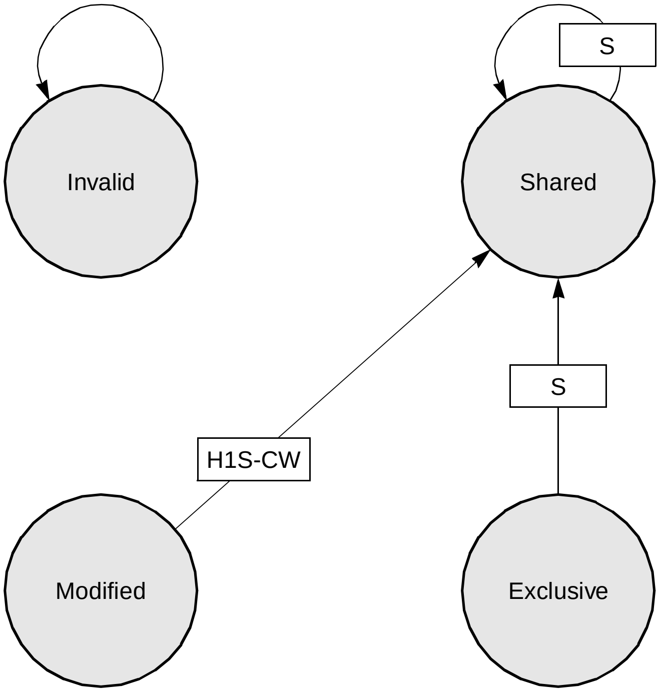


Figure 3-8. Read Transaction—MPX Bus Mode, L1_INTVEN = 0b100

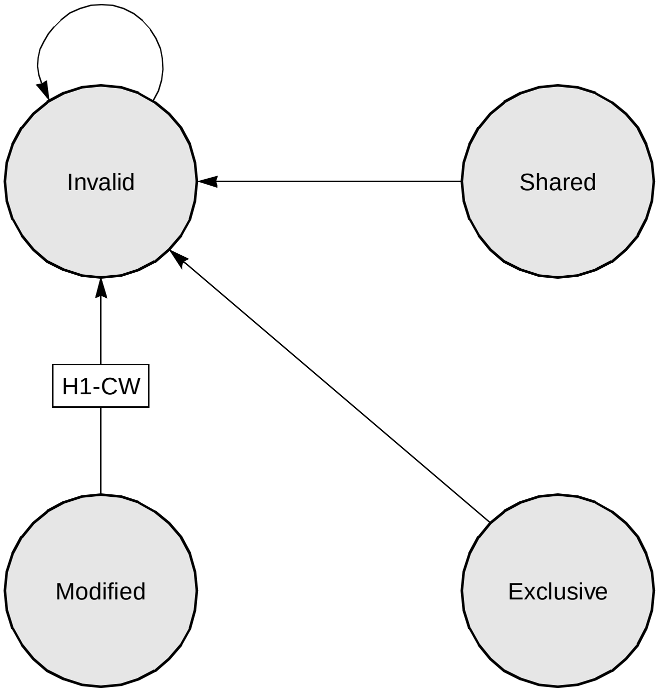


Figure 3-9. RWITM and Flush Transactions—MPX Bus Mode, L1_INTVEN = 0b100

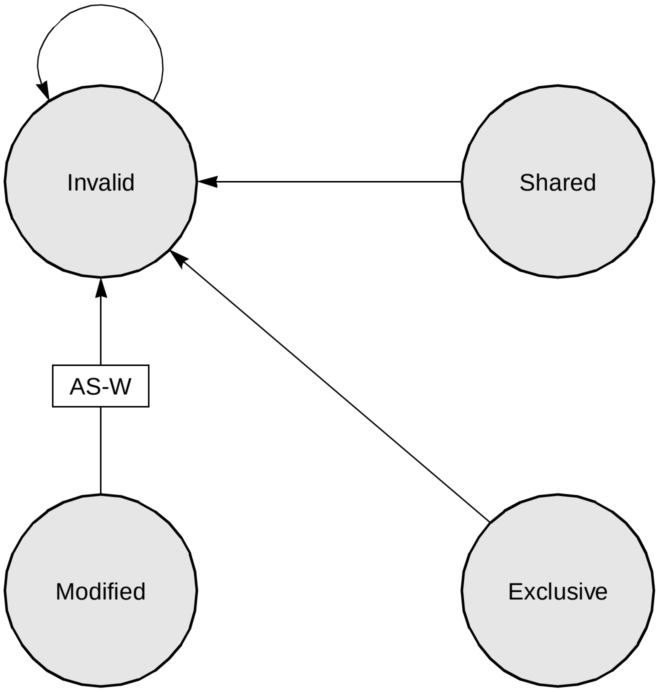


Figure 3-10. Write Transaction—MPX Bus Mode, L1_INTVEN = 0b100

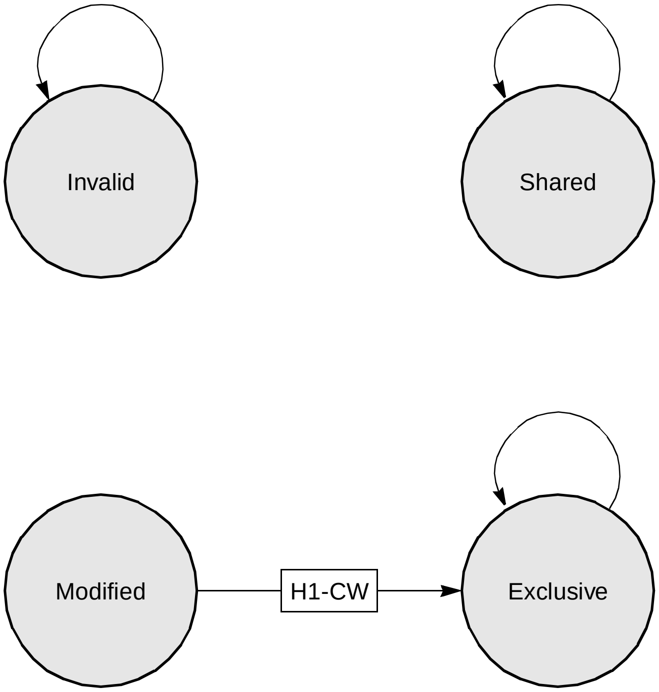
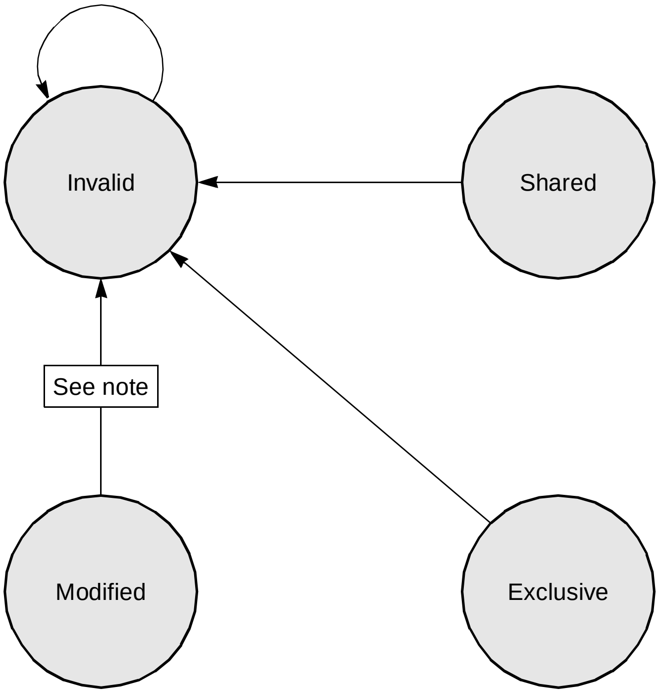


Figure 3-11. Clean Transaction—MPX Bus Mode, L1_INTVEN = 0b100



Note: If another master asserts $\overline{\text{ARTRY}}$, the MPC7410 performs a window-of-opportunity style push. Otherwise, there is no intervention.

Figure 3-12. Kill Transaction—MPX Bus Mode, L1_INTVEN = 0b100

3.4.3.4.3 MESI Protocol in MPX Bus Mode (with L1_INTVEN = 0b110)

The following state diagrams show the MESI state transitions when the MPC7410 is configured for MPX bus mode with modified and exclusive intervention (but not shared intervention) enabled (L1_INTVEN = 0b110).

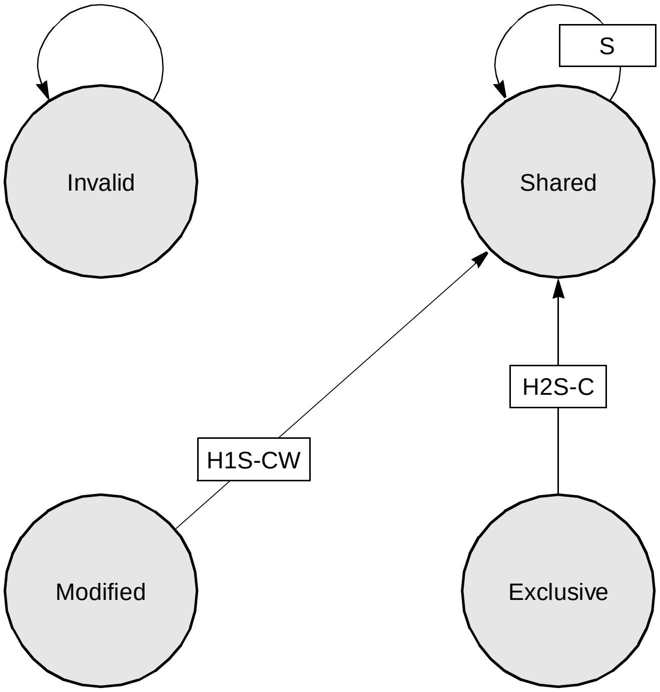


Figure 3-13. Read Transaction—MPX Bus Mode, L1_INTVEN = 0b110

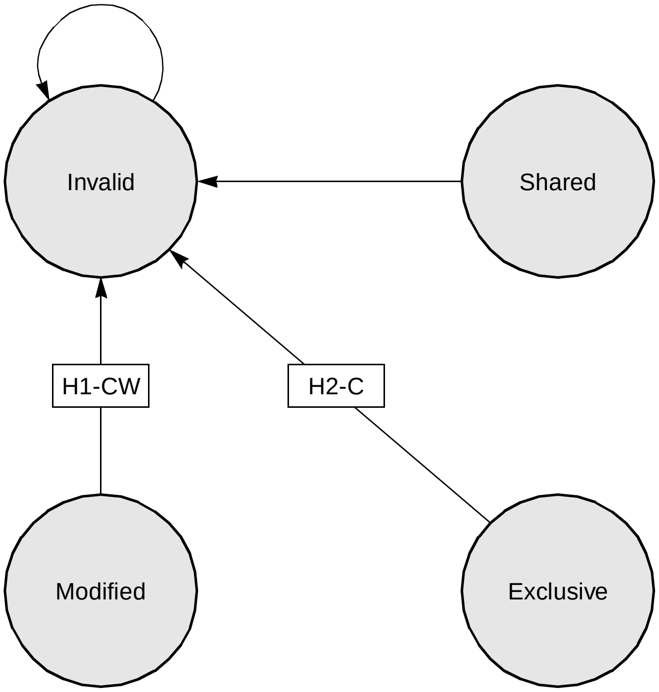


Figure 3-14. RWITM Transaction—MPX Bus Mode, L1_INTVEN = 0b110

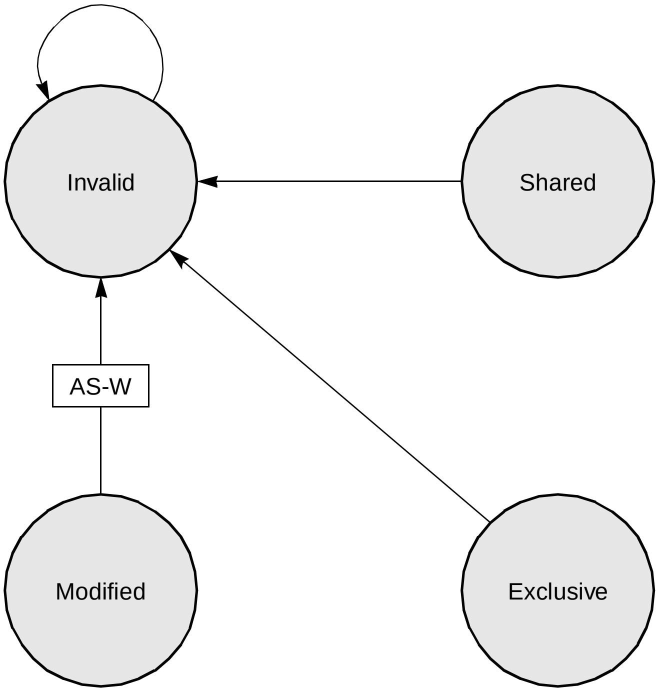


Figure 3-15. Write Transaction—MPX Bus Mode, L1_INTVEN = 0b110

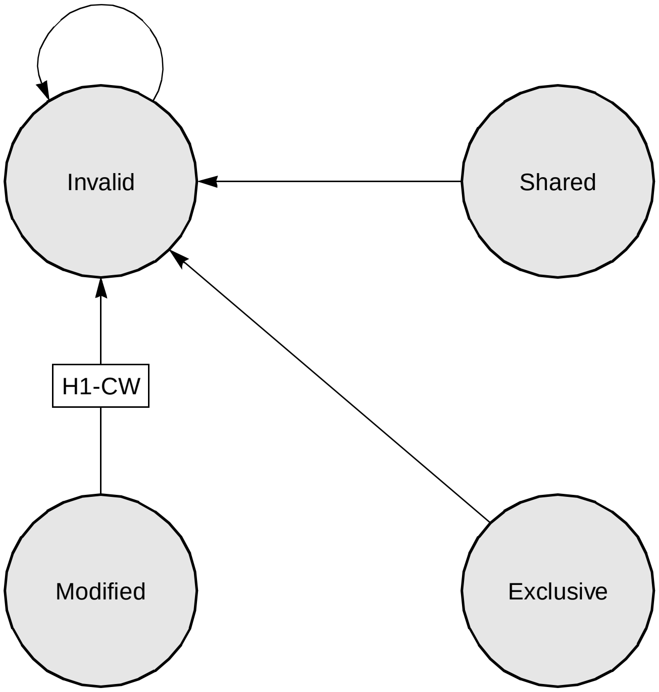


Figure 3-16. Flush Transaction State Diagram—MPX Bus Mode, L1_INTVEN = 0b110

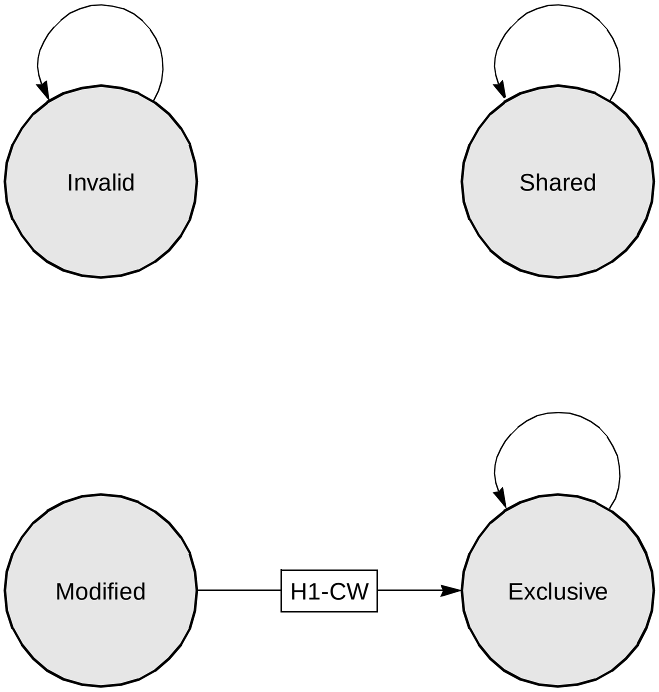
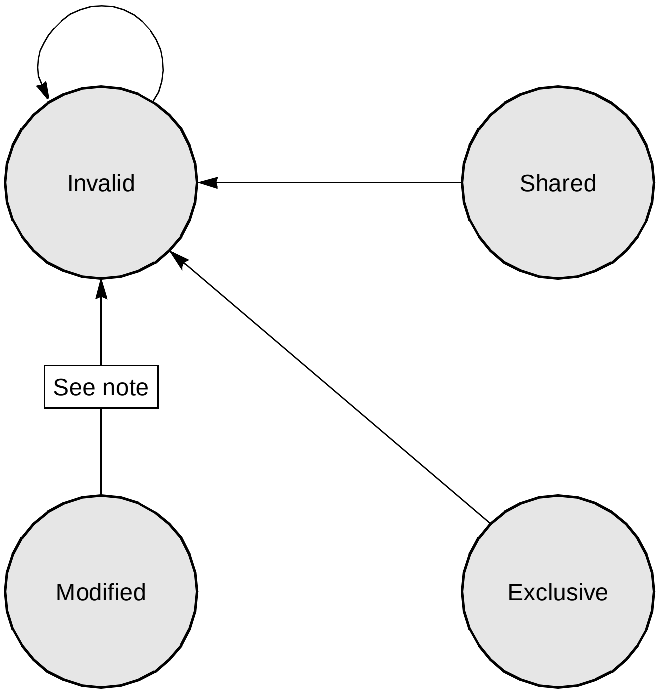


Figure 3-17. Clean Transaction—MPX Bus Mode, L1_INTVEN = 0b110



Note: If another master asserts $\overline{\text{ARTRY}}$, the MPC7410 performs a window-of-opportunity style push. Otherwise, there is no intervention.

Figure 3-18. Kill Transaction—MPX Bus Mode, L1_INTVEN = 0b110

3.4.3.5 MERSI State Transitions

The following state diagrams show the MERSI state transitions when the MPC7410 is configured for MPX bus mode with full (modified, exclusive, and shared) hit intervention enabled ([L1_INTVEN = 0b111).

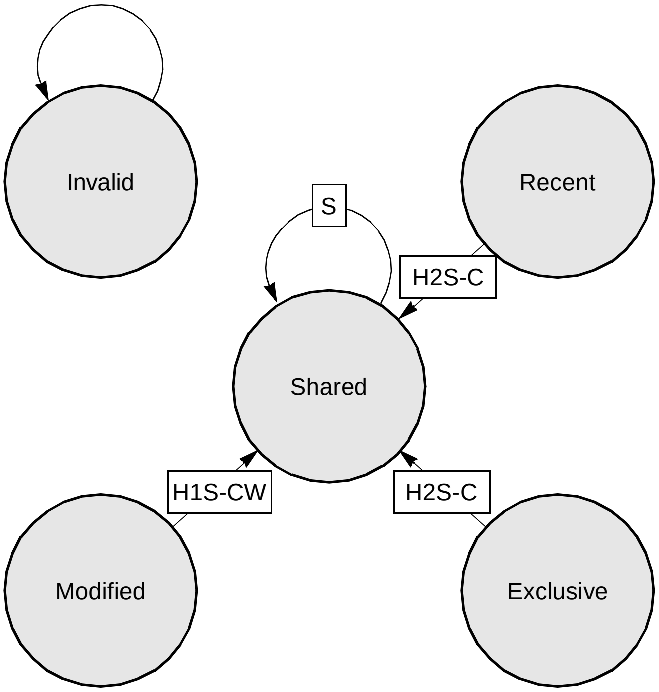


Figure 3-19. Read Transaction—MPX Bus Mode, L1_INTVEN = 0b111

Note that when the MPC7410 detects a snoop hit for a read transaction for a cache block marked recent (R), it asserts $\overline{\text{SHD}}$ and $\overline{\text{HIT}}$, and transitions the cache block to the shared (S) state. When the MPC7410 detects a snoop hit for data in the S state, it asserts $\overline{\text{SHD}}$, but it does not try to intervene by asserting $\overline{\text{HIT}}$. In this manner, only one version of shared data is ever available for intervention. This is strictly an optional extension and is not needed for masters that do not support shared intervention.

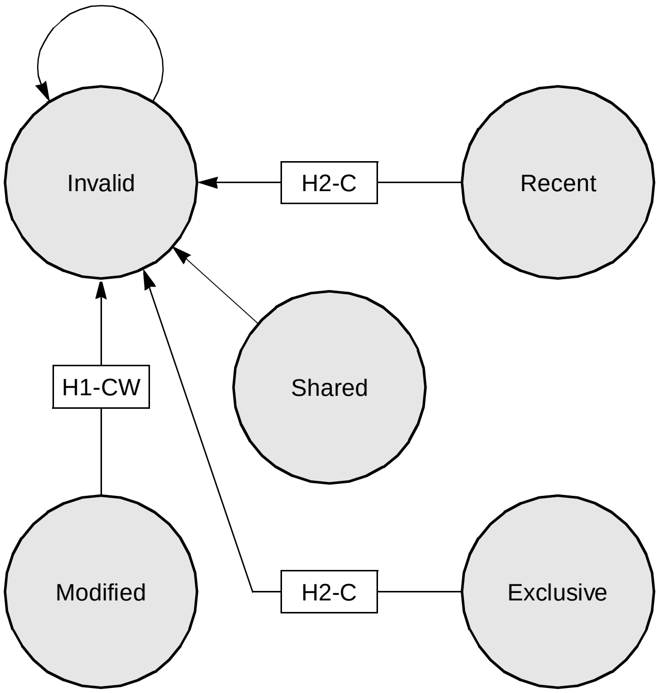


Figure 3-20. RWITM Transaction—MPX Bus Mode, L1_INTVEN = 0b111

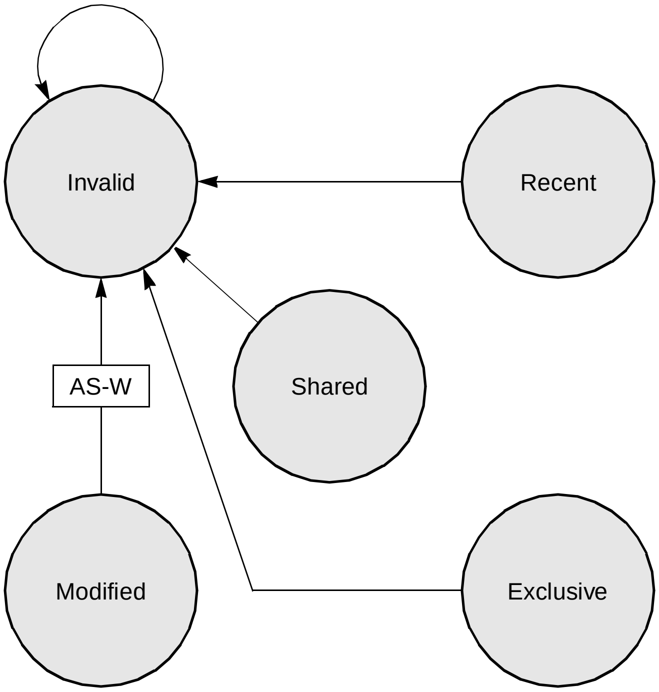


Figure 3-21. Write Transaction—MPX Bus Mode, L1_INTVEN = 0b111

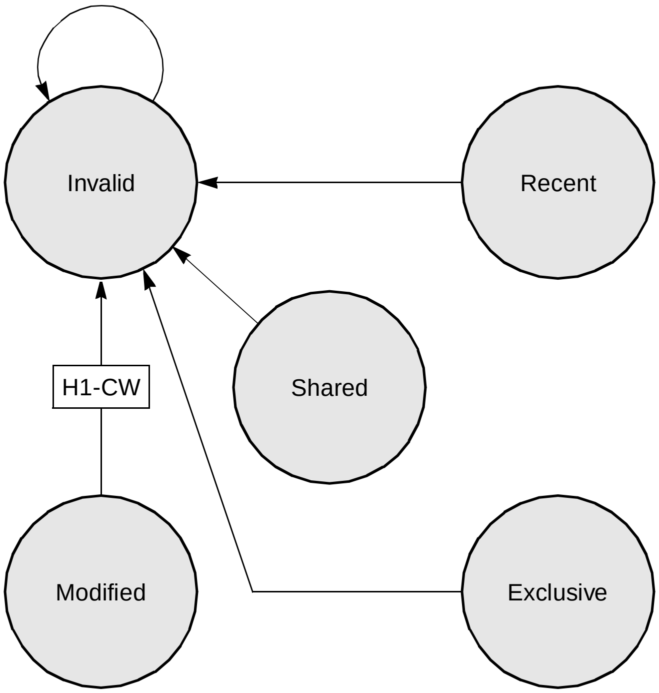


Figure 3-22. Flush Transaction—MPX Bus Mode, L1_INTVEN = 0b111

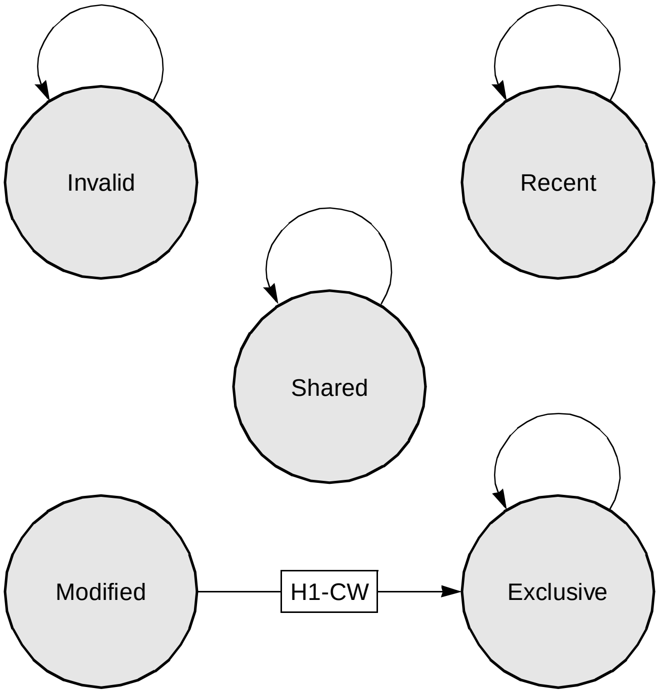
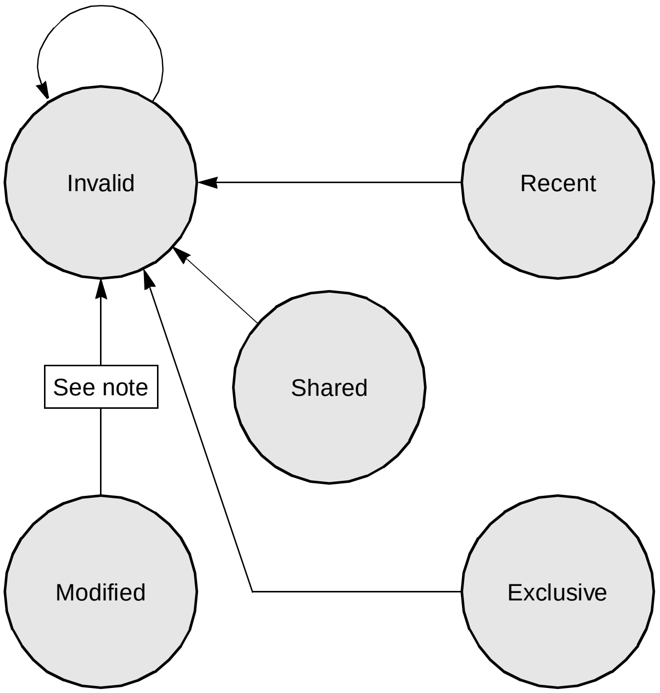


Figure 3-23. Clean Transaction—MPX Bus Mode, L1_INTVEN = 0b111



Note: If another master asserts ARTRY, the MPC7410 performs a window-of-opportunity style push. Otherwise, there is no intervention.

Figure 3-24. Kill Transaction—MPX Bus Mode, L1_INTVEN = 0b111

3.4.3.6 Reservation Snooping

The MPC7410 snoops all transactions against the contents of the reservation address register independent of the cache snooping. The following state diagrams show the response to those snoops.

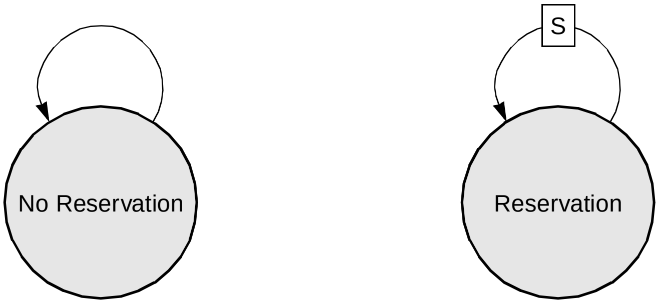


Figure 3-25. Read Transaction Snoop Hit on the Reservation Address Register

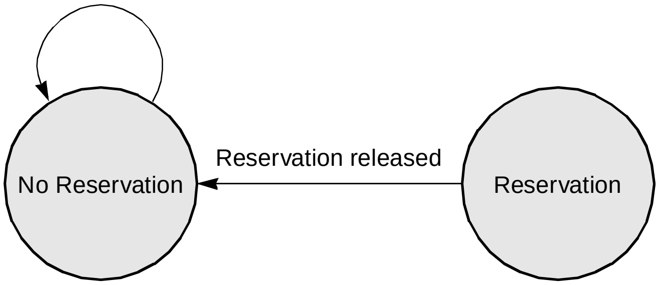


Figure 3-26. Reskill Transaction Snoop Hit on the Reservation Address Register

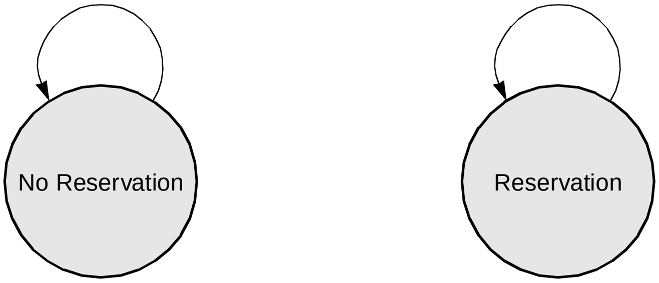


Figure 3-27. Transaction (other than Read or Reskill) Snoop Hit on the Reservation Address Register

3.4.3.7 State Changes for Self-Generated Bus Transactions

The MPC7410 snoops its own transactions and monitors the response from other masters. The following figures show the state changes for self-generated bus transactions. State transitions and snoop responses are shown. Each diagram denotes a specific bus transaction that the MPC7410 generates. The snoop responses from other masters in the system are shown beside each state transition line.

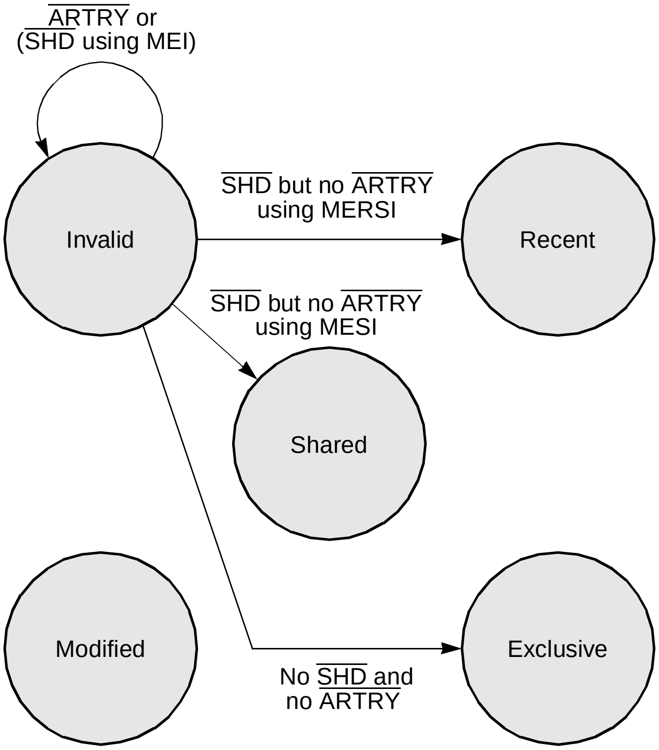


Figure 3-28. Self-Generated Data Read/Read-Atomic Transaction

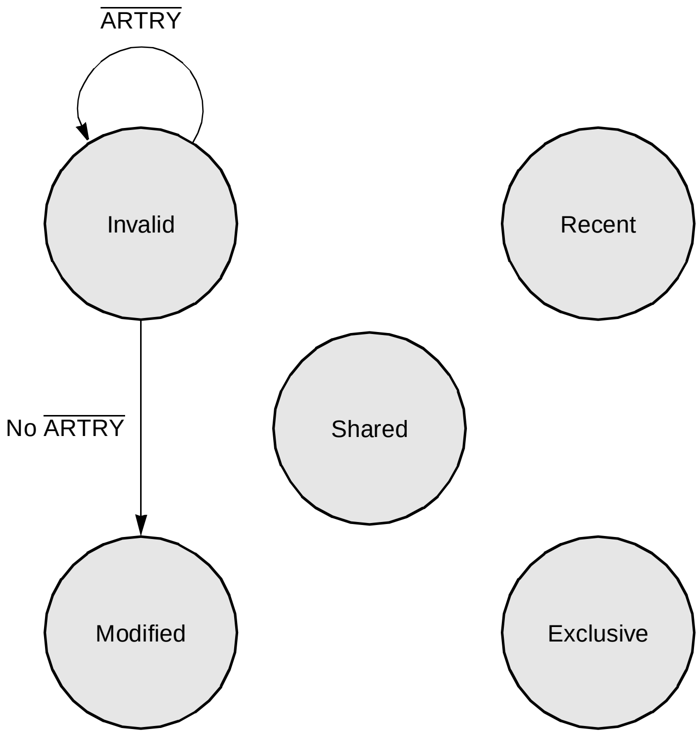


Figure 3-29. Self-Generated Data RWITM/RWITM-Atomic/Kill (Caused by dcbz Miss) Transaction

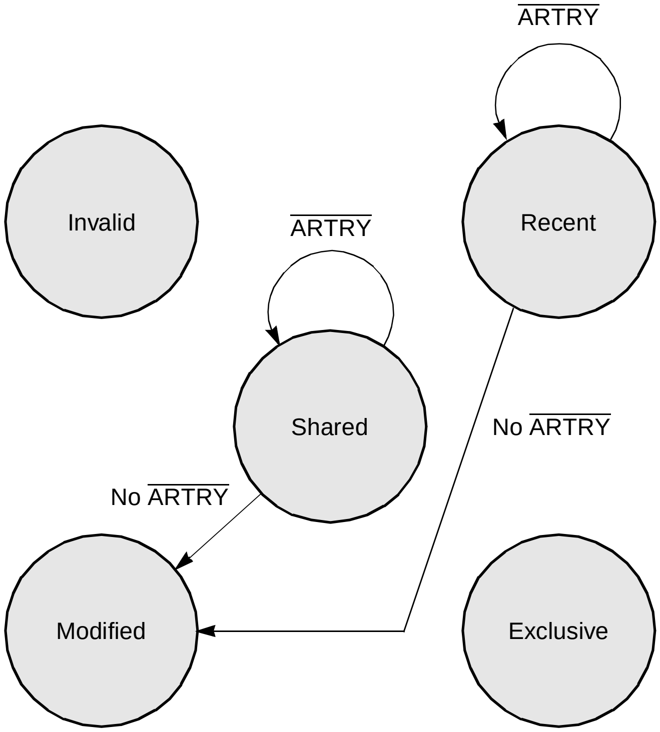

$\overline{\text{ARTRY}}$
Invalid
$\overline{\text{ARTRY}}$
Recent
Shared
No $\overline{\text{ARTRY}}$
No $\overline{\text{ARTRY}}$
Modified
Exclusive

Figure 3-30. Self-Generated Kill (Caused by Write Hit on S or R) Transaction

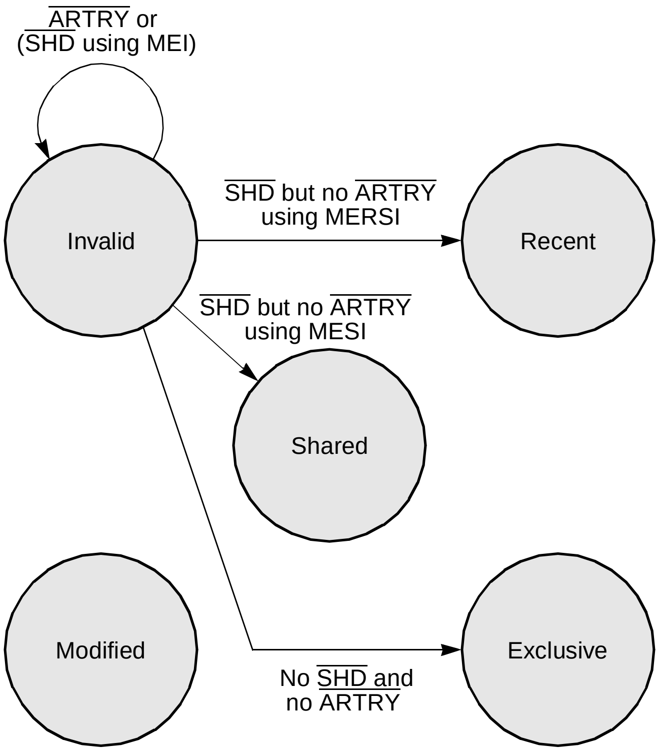


Figure 3-31. Self-Generated Read (Caused by Instruction Fetch) Transaction

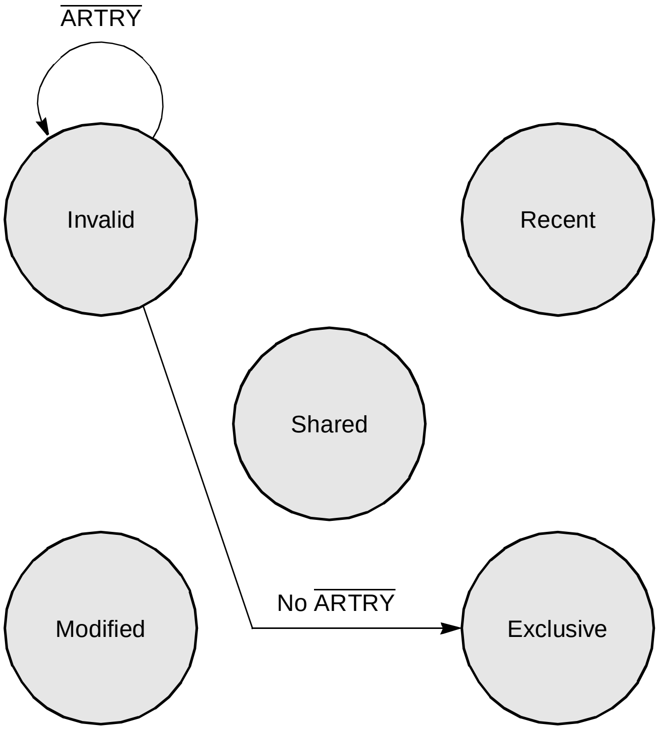


Figure 3-32. Self-Generated RCLAIM Transaction

3.4.4 MPC7410-Initiated Load/Store Operations

Load and store operations are assumed to be weakly ordered on the MPC7410. In general, the load/store unit (LSU) can perform load operations that occur later in the program ahead of store operations, even when the access is caching-inhibited or when data cache is disabled. Any load followed by any store is performed in order. See Section 3.4.4.2, “Sequential Consistency of Memory Accesses” for more information.

The MPC7410 does not provide support for direct-store segments. Operations attempting to access a direct-store segment will invoke a DSI exception. For additional information about DSI exceptions, refer to Section 4.6.3, “DSI Exception (0x00300).”

3.4.4.1 Performed Loads and Stores

The architecture defines a performed load operation as one that has the addressed memory location bound to the target register of the load instruction. The architecture defines a performed store operation as one where the stored value is the value that any other processor will receive when executing a load operation (that is, of course, until it is changed again). With respect to the MPC7410, caching-allowed (WIMG = x0xx) loads and caching-allowed, write-back (WIMG = 00xx) stores are performed when they have arbitrated to address the cache block in the L1 data cache, the L2 cache, or the system bus. Note that loads are considered performed at the L1 data cache and L2 cache only if the respective cache contains a valid copy of that address. Write-back stores are considered performed at the L1 data cache and L2 cache only if the respective cache contains a valid, non-shared copy of that address. Caching-inhibited (WIMG = x1xx) loads, caching-inhibited (WIMG = x1xx) stores, and write-through (WIMG = 10xx) stores are performed when they have been successfully presented to the external system bus.

3.4.4.2 Sequential Consistency of Memory Accesses

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

The MPC7410 achieves sequential consistency by operating a single pipeline to the cache/MMU. All memory accesses are presented to the MMU in exact program order and therefore exceptions are determined in order.

Table 3-8 defines the load/store ordering on the MPC7410 for each memory/cache access attribute setting.

Table 3-8. MPC7410 Load/Store Ordering

Cache/Memory Access Attributes	WIMG[1]	Store—Store Ordered	Load—Load Ordered	Store—Load Ordered	Load—Store Ordered
Caching-Inhibited, Guarded	01x1	Yes	Yes	Requires **eieio**	Yes
Caching-Inhibited, Non-Guarded	01x0	Yes	Yes	Requires **sync**	Yes
Write-Through, Guarded	10x1	Yes	Yes	Requires **sync**	Yes
Write-Through, Non-Guarded	10x0	Yes	Requires **eieio**	Requires **sync**	Yes

Table 3-8. MPC7410 Load/Store Ordering

Cache/Memory Access Attributes	WIMG[1]	Store—Store Ordered	Load—Load Ordered	Store—Load Ordered	Load—Store Ordered
Write-Back, Coherency-Required	001x	Requires **eieio**	Requires **eieio**	Requires **sync**	Yes
Write-Back, Coherency-Not-Required	000x	Requires **eieio**	Requires **eieio**	Requires **sync**	Yes

[1] The architecture states that combinations where WIMG = 11xx are not supported.

Loads are allowed to bypass stores once exception checking has been performed for the store, but data dependency checking is handled in the load/store unit so that a load will not bypass a store with an address match. Newer caching-allowed loads can bypass older caching-allowed loads only if the two loads are to different 32-byte address granules. Newer caching-allowed write-back stores can bypass older caching-allowed write-back stores if they do not store to overlapping bytes of data.

Note that although memory accesses that miss in the cache are forwarded to the reload buffer for future arbitration for the L2 cache and external bus, all potential synchronous exceptions have been resolved before the cache. In addition, although subsequent memory accesses can address the cache, full coherency checking between the cache and the memory queue is provided to avoid dependency conflicts.

3.4.4.3 Enforcing Store Ordering

Unlike previous PowerPC ISA microprocessor implementations, the MPC7410 does reorder cache-inhibited memory accesses and write-through, guarded memory accesses. As shown in Table 3-8, certain memory accesses require an **eieio** or a **sync** instruction to ensure ordering. These instructions are used to enforce storage ordering.

If store gathering is enabled, the **eieio** instruction may be used to keep stores from being gathered. If an **eieio** instruction is detected in the store queues, then store gathering is not performed. The **eieio** instruction causes a system bus broadcast, which may be used to prevent external devices, such as a bus bridge chip, from gathering stores.

3.4.4.4 Atomic Memory References

The architecture defines the Load Word and Reserve Indexed (**lwarx**) and the Store Word Conditional Indexed (**stwcx.**) instructions to provide an atomic update function for a single, aligned word of memory. These instructions can be used to develop a rich set of multiprocessor synchronization primitives. Note that atomic memory references constructed using **lwarx**/**stwcx.** instructions depend on the presence of a coherent memory system for correct operation. These instructions should not be expected to provide atomic access to noncoherent memory. For detailed information on these instructions, refer to Chapter 2, “Programming Model,” in this book and Chapter 8, “Instruction Set,” in *The Programming Environments Manual*.

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

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

Executing an **lwarx** or **stwcx.** instruction to areas marked write-through or when the L1 data cache is enabled and locked causes a DSI exception.

If the page table entry is marked caching-allowed (WIMG = x0xx), and an **lwarx** access misses in the cache, then the MPC7410 performs a cache block fill. If the page is marked caching-inhibited (WIMG = x1xx) and the access misses, then the **lwarx** instruction appears on the bus as a single-beat load. All bus operations that are a direct result of either an **lwarx** instruction or an **stwcx.** instruction are placed on the bus with a special encoding. Note that this does not force all **lwarx** instructions to generate bus transactions, but rather provides a means for identifying when an **lwarx** instruction does generate a bus transaction. If an implementation requires that all **lwarx** instructions generate bus transactions, then the associated pages should be marked as caching-inhibited. Note also that the MPC7410 uses the **lwarx** encoding to differentiate instruction fetches from data loads when HID0[IFTT] is set.

The MPC7410 implements a reservation signal ($\overline{\text{RSRV}}$) as on the MPC604- and the MPC750-family processors. The state of the reservation is always presented onto the $\overline{\text{RSRV}}$ output signal. This can be used to determine when an internal condition has caused a change in the reservation state.

3.5 Cache Control

The MPC7410’s L1 caches are controlled by programming specific bits in the HID0 and MSSCR0 special-purpose registers and by issuing dedicated cache control instructions. Section 3.5.1, “Cache Control Parameters in HID0,” describes the HID0 cache control bits, Section 3.5.2, “Data Cache Hardware Flush Parameter in MSSCR0,” describes the data cache hardware flush control in MSSCR0, and Section 3.5.3, “Cache Control Instructions,” describes the cache control instructions.

3.5.1 Cache Control Parameters in HID0

The HID0 special-purpose register contains several bits that invalidate, disable, and lock the instruction and data caches. The following sections describe these facilities.

3.5.1.1 Enabling and Disabling the Data Cache

The data cache may be enabled or disabled by using the data cache enable bit, HID0[DCE]. HID0[DCE] is cleared on power-up, disabling the data cache. Snooping is not performed when the data cache is disabled. Note that if the data cache is disabled, the L2 cache must also be disabled. The L2 cache is enabled/disabled by L2CR[L2E].

When the data cache is in the disabled state (HID0[DCE] = 0), the cache tag status bits are ignored, and all accesses are propagated to the system bus as single-beat transactions. Note that the $\overline{\text{CI}}$ (cache inhibit) signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) independent of the state of HID0[DCE]. Also note that disabling the data cache does not affect the translation logic; translation for data accesses is controlled by MSR[DR].

The setting of the DCE bit must be preceded by a **sync** instruction to prevent the cache from being enabled or disabled in the middle of a data access. In addition, the cache must be globally flushed before it is disabled to prevent coherency problems when it is re-enabled.

The **dcbz** instruction causes an alignment exception when the data cache is disabled. The touch load (**dcbt** and **dcbtst**) instructions are no-ops when the data cache is disabled; however, address translation is still performed for these instructions. Other cache instructions (**dcbf**, **dcbst**, and **dcbi**) do not affect the data cache when it is disabled.

3.5.1.2 Data Cache Locking

The contents of the data cache can be locked by setting the data cache lock bit, HID0[DLOCK]. For a locked data cache, there are no new tag allocations. Store hits and snoop hits are the only operations that can cause a tag state change in a locked data cache. The setting of the DLOCK bit must be preceded by a **sync** instruction to prevent the data cache from being locked during a data access.

The MPC7410 treats a load hit to a locked data cache the same as a load hit to an unlocked data cache. That is, the data cache services the load with the requested data. However, a load that misses in a locked data cache is passed to the reload buffer and propagated to the L2 cache or system bus as a caching-allowed, 32-byte burst read. But even though the reload buffer is filled with an entire cache block, the data cache is not updated with the new data. This allows for load miss folding for subsequent accesses to the cache block in the reload buffer without updating the locked cache.

As with load hits, write-back store hits to a locked data cache are treated the same as write-back store hits to an unlocked cache. Write-back store misses to a locked data cache are treated as if they were marked write-through. Note that because write-back store misses to a locked data cache are treated as write-through, store reordering may occur on the system bus when the processor is in the MPX bus mode (MSSCR0[EMODE] = 0b1). This can only occur if snoops are performed to the target address of the store when the address is not contained in the data cache but is contained in the L2 cache. To prevent this reordering, software must disable the exclusive and recent types of L2 cache $\overline{\text{HIT}}$ intervention when the data cache is locked by setting MSSCR0[L2_INTVEN] = 0b*n*00.

The MPC7410 treats snoop hits to a locked data cache the same as snoop hits to an unlocked data cache. However, any cache block invalidated by a snoop hit remains invalid until the cache is unlocked.

3.5.1.3 Data Cache Flash Invalidation

The data cache flash invalidate bit, HID0[DCFI], is used to invalidate the entire data cache in a single operation. Note that there is no broadcast of a Flash invalidate operation and any modified data in the cache will be lost. Individual data cache blocks are invalidated using the **dcbi** instruction. See Section 3.5.3.7, "Data Cache Block Invalidate (dcbi)," for more information about the **dcbi** instruction.

DCFI is set through an **mtspr** operation. The MPC7410 automatically clears DCFI in the clock cycle after it is set (provided that the data cache is enabled in the HID0 register). Note that some microprocessors accomplish data cache flash invalidation by setting and clearing HID0[DCFI] with two consecutive **mtspr** instructions (that is, the bit is not automatically cleared by the microprocessor). Software that has this sequence of operations does not need to be changed to run on the MPC7410.

The data cache is automatically invalidated when the MPC7410 is powered up and during a hard reset. However, a soft reset does not automatically invalidate the data cache. Software must set HID0[DCFI] to invalidate the entire data cache after a soft reset.

3.5.1.4 Enabling and Disabling the Instruction Cache

The instruction cache may be enabled or disabled through the use of the instruction cache enable bit, HID0[ICE]. HID0[ICE] is cleared on power-up, disabling the instruction cache. The setting of the ICE bit must be preceded by an **isync** instruction to prevent the cache from being enabled or disabled in the middle of an instruction fetch. The **icbi** instruction is not affected by disabling the instruction cache.

When the instruction cache is in the disabled state (HID[ICE] = 0), the cache tag status bits are ignored, and all instruction fetches are propagated to the system bus as single-beat transactions. Note that the $\overline{\text{CI}}$ signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) independent of the state of HID0[ICE]. Also note that disabling the instruction cache does not affect the translation logic; translation for instruction accesses is controlled by MSR[IR].

3.5.1.5 Instruction Cache Locking

The contents of the instruction cache can be locked by setting the instruction cache lock bit, HID0[ILOCK]. For a locked instruction cache, there are no new tag allocations. Snoop hits are the only operations that can cause a tag state change in a locked instruction cache. The setting of the ILOCK bit must be preceded by an **isync** instruction to prevent the instruction cache from being locked during an instruction fetch.

An instruction fetch that hits in a locked instruction cache is serviced by the cache. An instruction fetch that misses in a locked instruction cache is propagated to the system bus as a 32-byte burst read. However, the data is not loaded into the instruction cache. The data is loaded into the L2 cache (unless L2CR[L2DO] = 1).

Note that the $\overline{\text{CI}}$ signal always reflects the state of the caching-inhibited memory/cache access attribute (the I bit) independent of the state of HID0[ILOCK].

3.5.1.6 Instruction Cache Flash Invalidation

The instruction cache flash invalidate bit, HID0[ICFI], is used to invalidate the entire instruction cache in a single operation. Note that there is no broadcast of a flash invalidate operation. Individual instruction cache blocks are invalidated using the **icbi** instruction. See Section 3.5.3.8, “Instruction Cache Block Invalidate (icbi),” for more information about the **icbi** instruction.

ICFI is set through an **mtspr** operation. Once set, the MPC7410 automatically clears ICFI in the next clock cycle (provided that the instruction cache is enabled in the HID0 register). Note that some microprocessors accomplish instruction cache flash invalidation by setting and clearing HID0[ICFI] with two consecutive **mtspr** instructions (that is, the bit is not automatically cleared by the microprocessor). Software that has this sequence of operations does not need to be changed to run on the MPC7410.

The instruction cache is automatically invalidated when the MPC7410 is powered up and during a hard reset. However, a soft reset does not automatically invalidate the instruction cache. Software must set HID0[ICFI] to invalidate the entire instruction cache after a soft reset.

3.5.2 Data Cache Hardware Flush Parameter in MSSCR0

The MPC7410 provides a hardware flush mechanism to ease flushing of the data cache. It is controlled by MSSCR0[dL1HWF]. When the processor detects a state transition from 0 to 1 in dL1HWF, the MPC7410 initiates a hardware flush of the data cache.

The flush is performed by starting with low cache indices and increments through way 0 of the cache one index at a time until the maximum index value is obtained. Then, the index is reset to zero and the same process is repeated for ways 1, 2, 3, 4, 5, 6, and 7 of the data cache. For each index and way of the cache, the processor generates a non-global Write-with-Kill operation to the system bus for all modified cache blocks. At the end of the hardware flush, all lines in the data cache are invalidated.

During the flush, all memory subsystem requests to the data cache are stalled until the flush is complete. Snoops, however, are fully serviced by the data cache during the flush.

When the data cache tags have been fully flushed of all valid entries, the dL1HWF bit is cleared by hardware. Note that when dL1HWF is cleared, data cache flushes can still exist in the L1OPQ or below. A final **sync** instruction is required to guarantee that all data from the data cache has been written to the system address interface.

The recommended sequence to flush the data cache follows:

1. disable interrupts
2. **dssall**
3. **sync**
4. set MSSCR0[dL1HWF] = 1
5. **sync**

The data cache hardware flush mechanism is not present in earlier microprocessor implementations. Using MSSCR0[dL1HWF] is the preferred mechanism for flushing the data cache on the MPC7410.

3.5.3 Cache Control Instructions

The architecture defines instructions for controlling both the instruction and data caches (when they exist). The cache control instructions: **dcbt**, **dcbtst**, **dcbz**, **dcbst**, **dcbf**, **dcba**, **dcbi**, and **icbi**—are intended for the management of the local L1 and L2 caches. The MPC7410 interprets the cache control instructions as if they pertain only to its own L1 or L2 caches. These instructions are not intended for managing other caches in the system (except to the extent necessary to maintain coherency).

The MPC7410 snoops all global ($\overline{\text{GBL}}$ asserted) cache control instruction broadcasts. The **dcbst**, **dcbf**, and **dcbi** instructions cause a broadcast on the system bus (when M = 1) to maintain coherency. The **icbi** instruction is always broadcast, regardless of the state of the memory-coherency-required attribute. The MPC7410 treats any cache control instruction directed to a direct-store segment [T = 1] as a no-op.

3.5.3.1 Data Cache Block Touch (dcbt)

The Data Cache Block Touch (**dcbt**) instruction provides potential system performance improvement through the use of a software-initiated prefetch hint. Note that PowerPC ISA implementations are not required to take any action based on the execution of these instructions, but they may choose to prefetch the cache block corresponding to the effective address into their cache.

If the effective address of a **dcbt** instruction is directed to a direct-store segment [T = 1], or if HID0[NOPTI] = 1, the MPC7410 treats the instruction as a no-op without translation. This means that a table search operation is not initiated and the reference (R) bit is not set.

If the effective address of a **dcbt** instruction is not directed to a direct-store segment [T = 0] and HID0[NOPTI] = 0, the effective address is computed, translated, and checked for protection violations as defined in the architecture. The **dcbt** instruction is treated as a load to the addressed byte with respect to address translation and protection.

The MPC7410 treats the **dcbt** instruction as a no-op if any of the following occur:

- A valid address translation is not found in the BAT, TLB, or through a table search operation
- Load accesses are not permitted to the addressed page (protection violation)
- The BAT or PTE is marked caching-inhibited (I = 1)
- The cache is locked or disabled

Under these conditions, table search operations are performed and the reference bit is set, even though the instruction is treated as a no-op.

If none of the conditions for a no-op are met, the MPC7410 checks if the addressed cache block is in the L1 data cache. If the cache block is not in the L1 data cache, the MPC7410 checks if the addressed cache block is in the L2 cache. If the cache block is not in the L2 cache, the MPC7410 initiates a burst read (with no intent to modify) on the system bus.

The data brought into the cache as a result of this instruction is validated in the same manner that a load instruction would be (that is, it is marked as exclusive or shared). The memory reference of a **dcbt** instruction causes the reference bit to be set. Note also that the successful execution of the **dcbt** instruction affects the state of the TLB and cache LRU bits as defined by the PLRU algorithm (see Section 3.6.8, “Cache Block Replacement Selection”).

3.5.3.2 Data Cache Block Touch for Store (dcbtst)

The Data Cache Block Touch for Store (**dcbtst**) instruction behaves similarly to the **dcbt** instruction except for the following:

- If the target address of a **dcbtst** instruction is marked write-through (W = 1), the instruction is treated as a no-op
- If the **dcbtst** hits in the L1 data cache, the state of the block is not changed

- If the **dcbtst** misses in the L1 data cache, but hits in the L2 cache, the data is brought into the L1 data cache and is marked with the same state as in the L2 cache
- If the **dcbtst** misses in both the L1 data cache and the L2 cache, the cache block fill request is signaled on the bus as a read-with-intent-to-modify (60x-bus mode) or as a read-claim (MPX bus mode) and the data is marked exclusive when it is brought into the L1 data cache from the system bus

Note that since the **dcbtst** instruction is treated like a load in the cache hierarchy, cache blocks fetched by the **dcbtst** can not participate in the store-miss-merging mechanism. From a programming point of view, it is not wise to use a **dcbtst** unless the **dcbtst** can be placed sufficiently far ahead of any subsequent store to that same cache block such that the **dcbtst** can fully reload the L1 data cache before the store is attempted. If the store is attempted while the **dcbtst** cache block fill is still outstanding, the store will stall until the **dcbtst** has reloaded the L1. This can back up the load/store unit's committed store queue (CSQ). If the **dcbtst** instruction cannot be placed sufficiently ahead of the subsequent store instruction, it may be better to omit the **dcbtst** entirely.

If **dcbtst** (or **dstst**) is being used to prefetch a 32-byte coherency granule that will eventually be fully consumed by 32-byte's worth of stores (that is, two back-to-back AltiVec **stvx** instructions), the inclusion of touch-for-store may reduce performance if the system is bandwidth-limited. This is due to the fact that a touch-for-store must perform both a 32-byte coherency operation on the address bus (two or more bus cycles) and a 32-byte data transfer (four or more bus cycles). On the other hand, caching-allowed, write-back stores that merge to 32-bytes only require a 32-byte coherency operation (two or more bus cycles) because of the store-miss-merging mechanism. Since these store misses are already fully pipelined on MPC7410, placing a touch-for-store before a series of adjacent stores that will naturally merge may in fact degrade performance due to data bus bandwidth limitations.

3.5.3.3 Data Cache Block Zero (dcbz)

The effective address EA is computed, translated, and checked for protection violations as defined in the architecture. The **dcbz** instruction is treated as a store to the addressed byte with respect to address translation and protection.

For the **dcbz** instruction, after translating the EA, the MPC7410 establishes a block of all zeros in the reload buffer. The MPC7410 then performs one of the following coherency actions:

- If the corresponding memory page or block is marked memory-coherency-not-required, the block of zeros from the reload buffer is immediately written to the data cache.
- If the corresponding memory page or block is marked memory-coherency-required, and the **dcbz** hits to a cache block marked modified or exclusive, the block of zeros from the reload buffer is immediately written to the data cache.
- If the corresponding memory page or block is marked memory-coherency-required, and the **dcbz** hits to a cache block marked shared or recent, an address-only bus transaction (kill) is run prior to the block of zeros from the reload buffer being written to the data cache.
- If the corresponding memory page or block is marked memory-coherency-required, and the **dcbz** misses in the cache, an address-only bus transaction (kill) is run prior to the block of zeros from the reload buffer being written to the data cache.

Note that after any required coherency operations have been performed, the block of zeros from the reload buffer is written to the data cache, and the cache block is marked modified. The **dcbz** instruction does not alter the state of the L2 cache; however, it does check the L2 cache for normal cache coherent ownership by the MPC7410.

Executing a **dcbz** instruction to a disabled or locked data cache generates an alignment exception. Executing a **dcbz** instruction to an EA with caching-inhibited or write-through attributes also generates an alignment exception. BAT and TLB protection violations generate DSI exceptions.

3.5.3.4 Data Cache Block Store (dcbst)

The effective address is computed, translated, and checked for protection violations as defined in the architecture. This instruction is treated as a load with respect to address translation and memory protection.

If the address hits in the cache and the cache block is in the modified state, the modified block is written back to memory and the cache block is placed in the exclusive state. If the address hits in the cache and the cache block is in any state other than modified, an address-only broadcast (clean) is performed.

The function of this instruction is independent of the WIMG bit settings of the block or PTE containing the effective address. However, if the address is marked memory-coherency- required, the execution of **dcbst** causes an address broadcast on the system bus. Execution of a **dcbst** instruction does not affect the data cache or L2 cache if they are disabled.

A BAT or TLB protection violation generates a DSI exception.

3.5.3.5 Data Cache Block Flush (dcbf)

The effective address is computed, translated, and checked for protection violations as defined in the architecture. This instruction is treated as a load with respect to address translation and memory protection.

If the address hits in the cache, and the block is in the modified state, the modified block is written back to memory and the cache block is invalidated. If the address hits in the cache, and the cache block is in the exclusive or shared state, the cache block is invalidated. If the address misses in the cache, no action is taken.

The function of this instruction is independent of the WIMG bit settings of the block or PTE containing the effective address. However, if the address is marked memory-coherency- required, the execution of **dcbf** broadcasts an address-only FLUSH transaction on the system bus. Execution of a **dcbf** instruction does not affect data cache or L2 cache if they are disabled.

A BAT or TLB protection violation generates a DSI exception.

3.5.3.6 Data Cache Block Allocate (dcba)

The MPC7410 implements the data cache block allocate (**dcba**) instruction. This is currently an optional instruction in the virtual environment architecture (VEA); however, it may become required in future versions of the architecture. The **dcba** instruction provides potential system performance improvement through the use of a software-initiated pre-store hit. This allows software to establish a block in the data cache in anticipation of a store into that block, without loading the block from memory.

The MPC7410 executes the **dcba** instruction the same as a **dcbz** instruction, with one major exception. In cases when **dcbz** causes an exception, a **dcba** will no-op. Note that this means that a **dcba**/DABR address match does not cause an exception.

3.5.3.7 Data Cache Block Invalidate (dcbi)

The effective address is computed, translated, and checked for protection violations as defined in the architecture. This instruction is treated as a store with respect to address translation and memory protection.

If the address hits in the cache, the cache block is invalidated, regardless of the state of the cache block. Because this instruction may effectively destroy modified data, it is privileged (that is, **dcbi** is available to programs at the supervisor privilege level, MSR[PR] = 0).

The function of this instruction is independent of the WIMG bit settings of the block or PTE containing the effective address. However, if the address is marked memory-coherency- required, the execution of **dcbi** broadcasts an address-only kill transaction on the system bus. Execution of a **dcbi** instruction does not affect data cache or L2 cache if they are disabled.

A BAT or TLB protection violation for a **dcbi** translation generates a DSI exception.

3.5.3.8 Instruction Cache Block Invalidate (icbi)

The **icbi** instruction invalidates a matching entry in the instruction cache. During execution, the effective address for the instruction is translated through the data MMU, and broadcasts on the system bus using the memory-coherency attribute from translation. The MPC7410 always snoops global **icbi** transactions from the bus (even if it is the bus master that is broadcasting) and sends it to the instruction cache for cache block address comparison and invalidation. The MPC7410 snoops its own **icbi** broadcast regardless of the state of the $\overline{\text{GBL}}$ signal. The **icbi** instruction invalidates a matching cache entry regardless of whether the instruction cache is disabled or locked. The L2 cache is not affected by the **icbi** instruction.

An **icbi** instruction should always be followed by a **sync** and an **isync** instruction. This ensures that the effects of the **icbi** are seen by the instruction fetches following the **icbi** itself. For self-modifying code, the following sequence should be used to synchronize the instruction stream:

1. **dcbst** (push new code from data cache and L2 cache out to memory)
2. **sync** (wait for the **dcbst** to complete)
3. **icbi** (invalidate the old instruction cache entry in this processor and, by broadcasting the **icbi** to the bus, invalidate the entry in all snooping processors)
4. **sync** (wait for the **icbi** to complete its bus operation)
5. **isync** (re-sync this processor’s instruction fetch)

The second **sync** instruction ensures completion of all prior **icbi** instructions. Note that the second **sync** instruction is not shown in Section 5.1.5.2, “Instruction Cache Instructions,” in *The Programming Environments Manual*. This **sync** is required on the MPC7410.

Since the **sync** instruction strongly serializes the MPC7410’s memory subsystem, performance of code containing several **icbi** instructions can be improved by batching the **icbi** instructions together such that only one **sync** instruction is used to synchronize all the **icbi** instructions in the batch.

3.6 Cache Operations

This section describes the MPC7410 cache operations.

3.6.1 Data Cache Block Fill Operations

The MPC7410’s data cache blocks are filled (sometimes referred to as a cache reload) from an eight-entry reload buffer. Thirty two bytes of data are first collected in one of the reload data buffer entries before being reloaded into the data cache. This allows the data cache to service multiple outstanding misses while at the same time staying available to subsequent load and store hits. This behavior is described in Section 3.6.4, “Load Miss Folding,” and Section 3.6.5, “Store Miss Merging.”

A data cache block fill is caused by a load miss or write-back store miss in the cache. The cache block that corresponds to the missed address is updated by a burst transfer of the data from the L2 cache or system memory after any necessary coherency actions have completed.

3.6.2 Instruction Cache Block Fill Operations

The MPC7410’s instruction cache blocks are loaded in four beats of 64 bits each, with the critical double word loaded first. The instruction cache is not blocked to internal accesses while the fetch (caused by a cache miss) completes. This functionality is sometimes referred to as ‘hits under misses,’ because the cache can service a hit while a cache miss fill is waiting to complete. On a cache miss, the critical and following double words read from memory are simultaneously written to the instruction cache and forwarded to the instruction queue, thus minimizing stalls due to cache fill latency.

3.6.3 Allocation on Cache Misses

Instruction cache misses cause allocation into both the instruction cache and the L2 cache (assuming an L2 cache miss). Data cache misses cause allocation into the data cache only. They do not cause allocation into the L2 cache; the L2 cache is solely a victim cache for the data cache. The L2 cache allocates new entries for data accesses only when blocks are cast out of the data cache.

The castout (C), dirty (D), and modified (M) bits in the data cache tags are used to determine how a data cache replacement target is treated. If the replacement target is valid, then it is queued up as a castout if either the C or D bits are set. See Table 3-1 for the specific conditions for which the C and D bits are set and cleared.

When a block is queued up as a data cache castout and the L2 cache is enabled, the L2 cache allocates a new tag for the castout in the L2 cache if it misses and the C bit is set. If the C bit is cleared and the block misses in the L2 cache, the L2 cache does not allocate a tag. Instead, it passes the castout on to the system interface if the block is marked modified. If the data cache castout hits in the L2 cache, the castout data is written to the L2 cache regardless of the state of the C bit.

If the L2 cache is disabled, then the block replaced from the data cache is cast out to the system interface if the block is marked modified.

3.6.4 Load Miss Folding

The MPC7410's memory subsystem contains an eight entry reload buffer for L1 data cache reloads. The reload buffer consists of two main parts: an eight entry reload table (dRLT) which contains addresses and attributes, and an eight entry reload data buffer (dRLDB) which can store 32 bytes (a data cache block) per entry.

When a caching-allowed load or store misses in the data cache, an entry is allocated in the dRLT. If a subsequent load hits on a dRLT entry, it is placed in a four entry load fold queue (LFQ) with a tag pointing to the dRLT entry upon which it hit. When the proper bytes of data in the dRLDB become valid, then the load in the LFQ reads the data from the dRLDB and forwards it to the appropriate result bus. This is known as load miss folding.

Load miss folding effectively puts aside subsequent load misses to the same 32-byte data cache block to allow subsequent load and store access to the data cache.

Caching-inhibited loads are also allocated in the dRLT; however, subsequent loads are not allowed to fold into a dRLT entry allocated for a caching-inhibited load.

3.6.5 Store Miss Merging

When a caching-allowed store misses in the data cache, an entry is allocated in the dRLT and the store data is written into dRLDB. The remainder of the bytes not written by the store data are filled in when the cache block is eventually fetched from the L2 cache or the BIU. When all 32 bytes are valid, the cache block in the dRLDB is reloaded into the data cache.

If a subsequent store miss hits on a dRLT entry for a previous store miss, the subsequent store miss also writes its data into the dRLDB for that entry. The store can then drain from the completed store queue as it writes data to the dRLDB. The MPC7410 uses the coherency action performed by the first store miss for any subsequent stores to the same cache block in the reload buffer. When the coherency action for the original store miss that allocated the dRLT entry is complete and all 32 bytes of data are valid in the dRLDB, the cache block in the dRLDB is reloaded into the data cache. This behavior is known as store miss merging.

If a sufficient number of stores merge to the same dRLT entry such that all 32 bytes are written by store data, the reload buffer no longer needs to fill from the L2 cache or BIU. In this case, the cache block fill is treated as follows:

- If the original store that allocated the entry was marked memory-coherency-not-required, the cache block is immediately reloaded into the data cache without waiting for coherency action or data from the L2 cache or BIU.
- If the cache block fill request in the BIU for the reload buffer entry has not yet propagated to the bottom of the BIU's address queue, the transaction is completely dropped and does not appear on the address bus. In this case, store miss merging to non-global space enables the processor to silently allocate a new data cache block.

- If the cache block fill request in the BIU is at the bottom of the BIU's address queue but has not received a qualified bus grant for the read-with-intent-to-modify (RWITM) transaction, it performs an address-only kill broadcast instead. If the cache block fill request has already received a qualified bus grant, the transaction completes as a RWITM, but the data is discarded.

Note that two back-to-back AltiVec store misses can write a full 32-byte dRLT entry. For these back-to-back AltiVec stores, the MPC7410 nearly always performs kill coherency actions instead of RWITM transactions. Note that the chances of this happening decrease if other instructions are placed between the two stores or if a data dependency stalls the second store.

For large block copies to either global (memory-coherency-required) or non-global (memory-coherency-not-required) address space, the MPC7410 is more efficient if adjacent stores are used instead of **dcbz** or **dcba** instructions. This is due to the following three reasons:

- store hits to the data cache are fully pipelined whereas **dcbz**/**dcba** hits to the data cache can happen only once every four cycles best case
- the store miss merge mechanism allows the MPC7410 to issue kill transactions similar to **dcbz**/**dcba**
- **dcbz**/**dcba** instructions are usually used for prefetching; the real purpose of a copy is to perform real stores which the MPC7410 can perform just as efficiently without **dcbz**/**dcba** prefetches.

3.6.6 Store Hit to a Data Cache Block Marked Recent or Shared

Write-back stores that hit to a data cache block in the R or S state cannot be performed without first obtaining exclusive ownership of that block by a kill broadcast on the system bus.

When a write-back store hits on a shared or recent cache block, the target block is invalidated in the data cache. The current data from the target block is merged with the new store data and is copied into a reload buffer entry. A kill operation is propagated to the system bus. When the kill broadcast is successful, the target block is reloaded into the data cache in the MCD state.

Using the reload buffer for hit-on-shared/hit-on-recent simplifies snooping. If a snoop operation invalidates ownership of the target block before the kill operation is successful, then the reload buffer entry is changed to treat the entry like a normal store miss. In this case, the MPC7410 performs a RWITM operation on the address bus instead, and reloads the data cache in the MCD state.

3.6.7 Data Cache Block Push Operation

When a cache block in the MPC7410 is snooped and hit by another bus master and the data is modified, the cache block must be written to memory and made available to the snooping device. The cache block that is hit is said to be pushed out onto the system bus. The MPC7410 supports two kinds of snoop push operations—normal push operations and enveloped high-priority push operations, which are described in Section 9.4.4, “Using Data Bus Write Only (DBWO).”

3.6.8 Cache Block Replacement Selection

Both the instruction and data cache use a pseudo least-recently-used (PLRU) replacement algorithm when a new block needs to be placed in the cache. Note that data cache replacement selection is performed at reload time, not when a miss occurs. Instruction cache replacement selection occurs when an instruction cache miss is first recognized. This is fundamentally different from the data cache in that the replacement target is selected upon miss and not upon reload.

Each cache is organized as eight blocks (ways) per set by 128 sets. There is a valid bit for each way in the cache, L[0–7]. The replacement logic first checks to see if there are any invalid ways in the set and chooses the lowest-order, invalid block (L[0–7]) as the replacement target. When all eight ways in the set are valid, the PLRU algorithm is used to select the replacement target. There are seven PLRU bits, B[0–6] for each set in the cache.

A way is selected for replacement according to the PLRU bit encodings shown in Table 3-9.

Table 3-9. PLRU Replacement Way Selection

If the PLRU bits are:						Then the way selected for replacement is:
B0	0	B1	0	B3	0	L0
	0		0		1	L1
	0		1	B4	0	L2
	0		1		1	L3
	1	B2	0	B5	0	L4
	1		0		1	L5
	1		1	B6	0	L6
	1		1		1	L7

The PLRU algorithm is shown graphically in Figure 3-33.

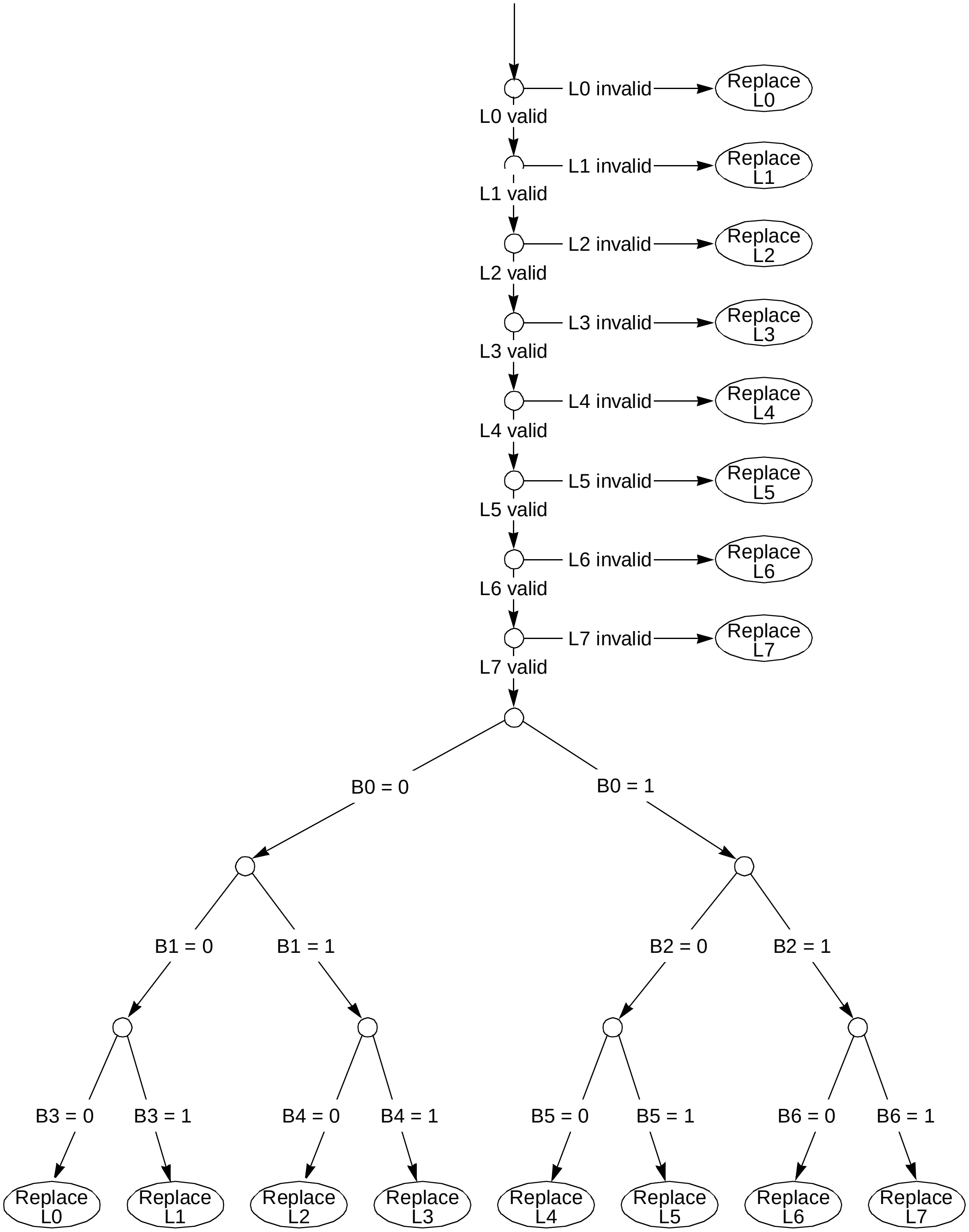


Figure 3-33. PLRU Replacement Algorithm

Data cache replacement selection can be modified by the data cache flush assist bit, HID0[DCFA]. When set, HID0[DCFA] forces the PLRU replacement algorithm to ignore any invalid entries and follow the replacement sequence defined by the PLRU bits. This can be used to simplify software flushing of the data cache. See Section 3.6.9, “L1 Cache Invalidation and Flushing,” for more information. HID0[DCFA] does not affect instruction cache replacement selection. If any of the valid bits (L[0–7]) for a given set in the instruction cache are invalid, the first invalid entry (from L0 to L7) is always chosen as the replacement way.

During power-up or hard reset, all the valid bits of the ways are cleared and the PLRU bits are cleared to point to way L0 of each set. Note that this is also the state of the data or instruction cache after setting their respective flash invalidate bits (HID0[DCFI] or HID0[ICFI]).

Each time a cache block is accessed, it is tagged as the most recently used way of the set (unless accessed by the AltiVec LRU instructions; refer to Section 7.1.2.1, “LRU Instructions”). For every hit in the cache or when a new block is reloaded, the PLRU bits for the set are updated using the rules specified in Table 3-10.

Table 3-10. PLRU Bit Update Rules

If the current access is to:	Then the PLRU bits in the set are changed to:						
	B0	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**
L0	1	1	x	1	x	x	x
L1	1	1	x	0	x	x	x
L2	1	0	x	x	1	x	x
L3	1	0	x	x	0	x	x
L4	0	x	1	x	x	1	x
L5	0	x	1	x	x	0	x
L6	0	x	0	x	x	x	1
L7	0	x	0	x	x	x	0

x = Does not change

Note that only three PLRU bits are updated for any given access.

3.6.8.1 AltiVec LRU Instruction Support

The data cache fully supports the AltiVec LRU instructions (**lvxl**, **stvxl**). If one of these instructions causes a hit in the data cache, then the PLRU bits are updated such that the way which hit is marked as least-recently-used by using the PLRU update rules shown in Table 3-11. If no other hit to the cache index occurs, this way is victimized upon the next data cache reload. Similarly, if an **lvxl** or **stvxl** instruction misses in the cache, the PLRU bits are updated as shown in Table 3-11 when that cache block reloads the data cache. Note that the instruction cache is not subject to any AltiVec LRU accesses.

Table 3-11. PLRU Bit Update Rules for AltiVec LRU Instructions

If the current AltiVec LRU access is to:	Then the PLRU bits in the set are changed to:						
	B0	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**
L0	0	0	x	0	x	x	x
L1	0	0	x	1	x	x	x
L2	0	1	x	x	0	x	x
L3	0	1	x	x	1	x	x
L4	1	x	0	x	x	0	x
L5	1	x	0	x	x	1	x
L6	1	x	1	x	x	x	0
L7	1	x	1	x	x	x	1

x = Does not change

Note that an AltiVec LRU access simply inverts the update value of the three PLRU bits when compared to the normal (MRU) update rules.

3.6.9 L1 Cache Invalidation and Flushing

The data cache can be invalidated by executing a series of **dcbi** instructions or by setting HID0[DCFI]. The instruction cache can be invalidated by executing a series of **icbi** instructions or by setting HID0[ICFI].

Any modified entries in the data cache can be copied back to memory (flushed) by using the hardware flush mechanism described in Section 3.5.2, “Data Cache Hardware Flush Parameter in MSSCR0.” Because the instruction cache never contains modified entries, no flushing mechanism is necessary.

While the hardware flush mechanism for the data cache is the preferred flush mechanism, software flush routines used for the MPC750 can also be used to flush the MPC7410 data cache. Note that future MPC7410 derivatives may not support the MPC750 software flush mechanism.

The software flush routines flush the data cache by using the **dcbf** instruction or by executing a series of 12 uniquely addressed load or **dcbz** instructions to each of the 128 sets. The address space should not be shared with any other process to prevent snoop hit invalidations during the flushing routine. Exceptions should be disabled during this time so that the PLRU algorithm does not get disturbed.

The data cache flush assist bit, HID0[DCFA], simplifies the software flushing process. When set, HID0[DCFA] forces the PLRU replacement algorithm to ignore the invalid entries and follow the replacement sequence defined by the PLRU bits. This reduces the series of uniquely addressed load or

dcbz instructions to eight per set. HID0[DCFA] should be set just prior to the beginning of the cache flush routine and cleared after the series of instructions is complete.

3.7 L2 Cache Interface

This section describes the MPC7410 microprocessor L2 cache interface, and its configuration and operation. It describes how the MPC7410 signals, defined in Chapter 8, "Signal Descriptions," interact to perform address and data transfers to and from the L2 cache.

3.7.1 L2 Cache Interface Overview

The MPC7410's L2 cache interface is implemented with an on-chip, two-way set associative tag memory with 8192 (8K) tags per way, and a dedicated interface with support for up to 2 Mbyte of external synchronous SRAMs.

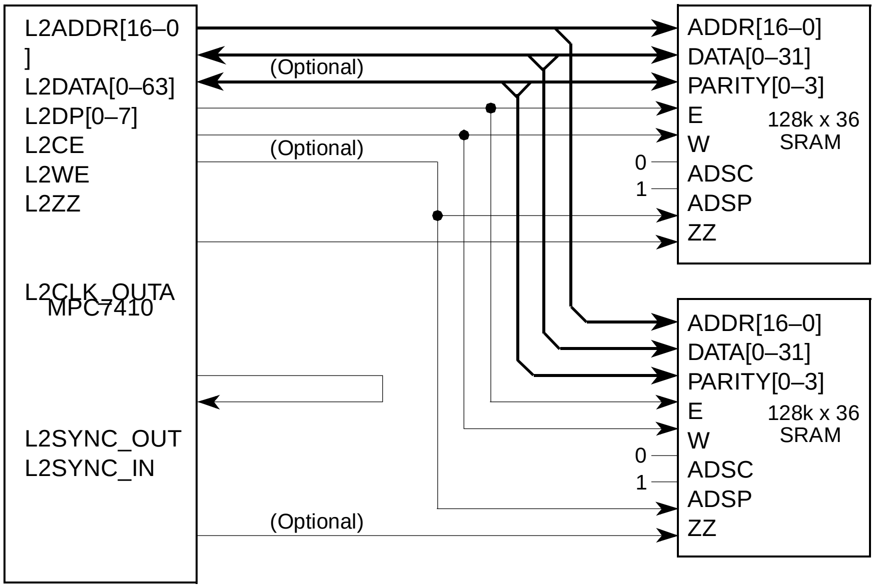
The tags are sectored to support either four, two, or one cache blocks per tag entry depending on the L2 cache size. Each sector (32-byte cache block) in the L2 cache has three status bits that are used to implement the MERSI cache coherency protocol (or the MESI and MEI subsets). The MPC7410's L2 cache may be configured to operate in write-back or write-through mode and maintains cache coherency through snooping.

The L2 interface can be configured to use half (256 Kbytes minimum) or all of the SRAM area as a direct-mapped, private memory space. Accesses to the private memory space do not propagate to the L2 cache nor are they visible to the external system bus.

The L2 cache control register (L2CR) allows control of L2 cache configuration and interface timing. The L2 private memory control register (L2PMCR) is provided for configuration of the private memory feature.

The L2 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.

Figure 3-34 shows the MPC7410 configured with a 1-Mbyte L2 cache.



Notes:

For a 2-Mbyte L2 cache, use address bits 17–0 (bit 0 is LSB).

For a 1-Mbyte L2 cache, use address bits 16–0 (bit 0 is LSB).

For a 512-Kbyte L2 cache, use address bits 15–0 (bit 0 is LSB).

For a 256-Kbyte L2 cache, use address bits 14–0 (bit 0 is LSB).

External clock routing should ensure that the rising edge of the L2 cache clock is coincident at the K input of all SRAMs and at the L2SYNC_IN input of the MPC7410. The clock A network can be used solely or the clock B network can also be used depending on loading, frequency, and number of SRAMs.

No pull-up resistors are normally required for the L2 cache interface.

The MPC7410 supports only one bank of SRAMs.

For high-speed operation, no more than two loads should be presented on each L2.

Figure 3-34. Typical 1-Mbyte L2 Cache Configuration

3.7.2 L2 Cache Organization

The L2 cache tags are configured for four sectors (128 bytes) for every tag entry when 2 Mbyte of external SRAM is used. The L2 cache tags are configured for two sectors (64-bytes) for every tag entry when 1 Mbyte of external SRAM is used. If the L2 cache is configured for 512 Kbytes or 256 Kbytes of external

SRAM, the tags are configured for one sector (32-bytes) per tag entry. Figure 3-35 shows the organization of the L2 cache tags.

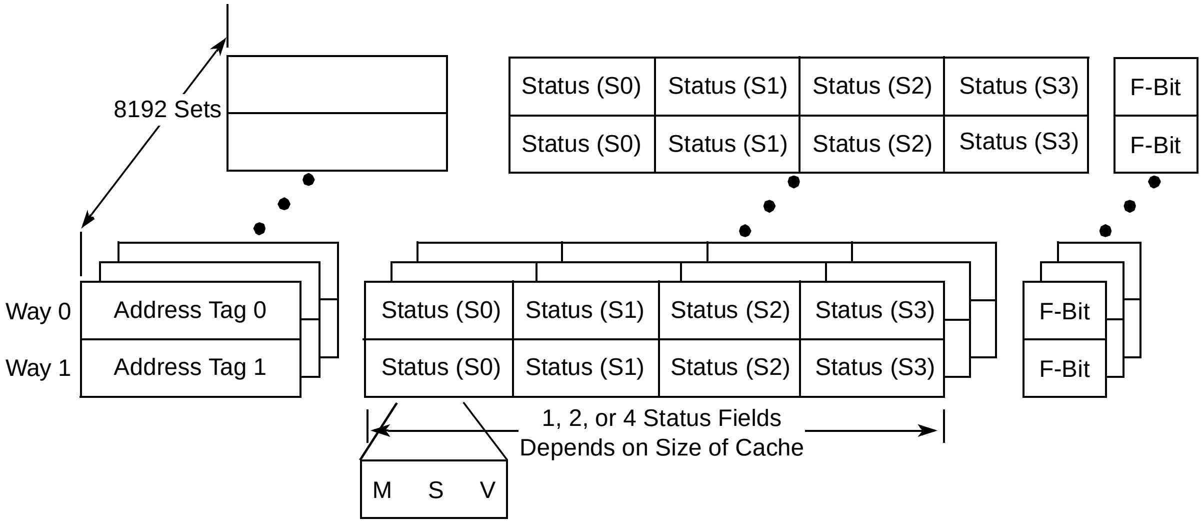


Figure 3-35. L2 Cache Controller Tag Organization

Physical address bits PA[13:24] provide the index to select a cache set. The tags consist of physical address bits PA[0:12]. Physical address bits A[25:31] locate a byte within the selected block.

3.7.2.1 L2 Cache Tag Status Bits

The L2 cache tag contains modified (M), shared (S), and valid (V) status bits for each of the two ways and four sectors. Table 3-12 describes the supported L2 cache states.

Table 3-12. Legal L2 Cache States

MSV value			MERSI state	Comments
M	S	V		
1	0	1	Modified	Cast out from data cache
0	0	1	Exclusive	Could be instruction or data
1	1	1	Recent	Could be instruction or data
0	1	1	Shared	Could be instruction or data
x	x	0	Invalid	Invalid line

The L2 cache tag also contains a FIFO replacement bit (F-bit) for each index. The F-bit is used for selecting a replacement target upon L2 cache reload. It is updated when a new tag is allocated in the L2 cache tag. See Section 3.7.6.2, “L2 Cache Replacement Selection,” for more information.

3.7.3 L2 Cache Control Register (L2CR)

The L2 cache control register (L2CR) allows control of L2 cache configuration, timing, and operation. The following sections describe the L2 cache control parameters in the L2CR.

The L2CR is a supervisor-level read/write, implementation-specific register that is accessed as SPR 1017. The contents of the L2CR are cleared during power-on reset. See Section 2.1.5.4.2, “L2 Cache Control Register (L2CR),” for additional information about the configuration of the L2CR.

3.7.3.1 Enabling and Disabling the L2 Cache

The L2 cache may be enabled or disabled by programming the L2CR[L2E] parameter. This parameter enables or disables the operation of the L2 cache (including snooping) starting with the next transaction that the L2 cache unit receives. When the L2 cache is disabled, the cache tag status bits are ignored and all accesses are propagated to the system bus. Note that if the L2 cache is enabled, the L1 data cache must also be enabled. Conversely, if the L1 data cache is disabled, the L2 cache must also be disabled.

Before enabling the L2 cache, the L2 clock must first be configured through the L2CR[L2CLK] bits, and a period of time must elapse for the L2 DLL to stabilize. See the MPC7410 hardware specifications for the DLL stabilization interval. Also before enabling the L2 cache, all other bits in the L2CR must be set appropriately, and the L2 cache may need to be globally invalidated. See Section 3.7.5, “L2 Cache Initialization,” for a description of the L2 cache initialization procedures.

Before the L2 cache is disabled it must be flushed to prevent coherency problems. The cache management instructions **dcbf**, **dcbst**, and **dcbi** do not affect the L1 data cache or L2 cache when they are disabled.

3.7.3.2 L2 Cache Parity Checking and Generation

The L2CR[L2PE] parameter enables or disables parity checking for the L2 data RAM interface. When L2PE is cleared, L2 parity checking is disabled. Note that The L2 interface always generates and drives parity on the L2DP[0:7] signals for writes to the SRAM array in 64-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b00). For 32-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b10), the L2 interface drives parity on the L2DP[0:3] signals and drives the L2DP[4:7] signals low.

3.7.3.3 L2 Cache Size

The L2CR[L2SIZ] bits configure the size of the L2 cache. They should be set according to the size of the L2 data bus and the organization of the L2 data RAMs that are present. Table 3-13 lists the data RAM organizations for the various L2 cache sizes. Table 3-13 also indicates typical SRAM sizes that might be used to construct such a cache.

Table 3-13. L2 Cache Sizes and Data RAM Organizations

L2 Cache Size	L2 Data Bus Size	L2 Data RAM Organization	Example SRAM Sizes That Might Be Used
256 Kbytes	64/72 bit	32K x 64/72	(2) 32K x 32/36
	32/36 bit	64K x 32/36	(1) 64K x 32/36
512 Kbytes	64/72 bit	64K x 64/72	(2) 64K x 32/36
	32/36 bit	128K x 32/36	(1) 128K x 32/36
1 Mbyte	64/72 bit	128K x 64/72	(2) 128K x 32/36
	32/36 bit	256K x 32/36	(2) 256K x 16/18

Table 3-13. L2 Cache Sizes and Data RAM Organizations (continued)

L2 Cache Size	L2 Data Bus Size	L2 Data RAM Organization	Example SRAM Sizes That Might Be Used
2 Mbytes	64/72 bit	256K x 64/72	(4) 256K x 16/18
	32/36 bit	512K x 32/36	(2) 512K x 16/18

Note: The MPC7410 supports only one bank of SRAMs. For very high speed operation, no more than two SRAMs should be used.

3.7.3.4 L2 Cache SRAM Types

The L2CR[L2RAM] bits configure the L2 RAM interface for the type of synchronous SRAMs that are used. The MPC7410 supports:

- Pipelined (register-register) burst SRAMs, which clock addresses in and clock data out
- Late-write SRAMs, which are required by the MPC7410 to be of the pipelined (register-register) configurations
- Newer generation pipeline burst SRAMs, referred to as PB3-type SRAMs

Note that the burst feature built into standard burst SRAMs and late-write SRAMs is not used by the MPC7410. The PB3-type SRAMs require the burst feature to be used, so the MPC7410 supports a 4-beat burst mode for PB3 SRAMs.

3.7.3.5 L2 Cache Write-Back/Write-Through Modes

The L2 cache normally operates in write-back mode. The L2CR[L2WT] parameter may be used to select write-through mode. In write-through mode, all writes to the L2 cache are also written to the system bus. For these writes, the L2 cache entry is always marked as exclusive rather than modified. L2WT must never be set after the L2 cache has been enabled as previously modified lines may get re-marked as exclusive during the course of normal operation.

3.7.3.6 L2 Cache Data-Only and Instruction-Only Operation

The L2CR[L2DO] parameter enables data-only operation in the L2 cache. For data-only operation, only transactions from the L1 data cache are allowed to be reloaded into the L2 cache. Instruction addresses already in the cache still hit for the L1 instruction cache. L2DO may be dynamically programmed as needed.

The L2CR[L2IO] parameter enables instruction-only operation in the L2 cache. For instruction-only operation, only transactions from the L1 instruction cache are allowed to be reloaded into the L2 cache. Data addresses already in the cache still hit for the L1 data cache. L2IO may be dynamically programmed as needed.

3.7.3.6.1 L2 Cache Locking Using L2DO and L2IO

The MPC7410's L2 cache can be locked by setting both the L2DO and L2IO bits of the L2CR. This prevents instruction cache misses from reloading the L2 cache and prevents data cache castouts from allocating entries in the L2 cache. Data cache castouts in the modified state are forwarded to the system interface. Note that locking the L2 cache using this mechanism is completely independent of L1 data or instruction cache locking.

3.7.3.7 L2 Cache Global Invalidation

The MPC7410 supports global (not flash) invalidation of the L2 cache through the L2CR[L2I] parameter. Setting L2I causes a global invalidation of the L2 cache. A global invalidation is performed by automatically sequencing through the L2 cache tags and clearing all bits of the tag (tag data bits, tag status bits, and FIFO bit). The global invalidation function must be performed only while the L2 cache is disabled. L2I must never be set while the L2 cache is enabled. During the invalidation, all memory activity from the L1 data and instruction caches are blocked from accessing the L2 until the invalidation is complete.

The L1 caches are invalidated automatically upon power-up (hard reset), but the L2 cache tags must be explicitly invalidated by software setting the L2I bit.

L2CR[L2IP] is a read-only bit that indicates whether an L2 global invalidate is in progress. It should be monitored after an L2 global invalidate has been initiated to determine when the global L2 invalidation has completed.

The sequence for performing a global invalidation of the L2 cache is as follows:

1. Prefetch the code that monitors L2CR[L2IP] (step 5) into the L1 instruction cache. The L2IP monitor code must be resident in the L1 instruction cache before the L2CR[L2I] bit is set (step 4). Otherwise the global invalidate operation will prevent the fetching of the L2IP monitor code from memory until after the invalidate has completed and the L2IP monitor code will never see the L2IP bit set.
2. Execute a **dssall** instruction to cancel any pending data stream touch instructions.
3. Execute a **sync** instruction to finish any pending store operations in the load/store unit, disable the L2 cache by clearing L2CR[L2E], and execute an additional **sync** instruction after disabling the L2 cache to ensure that any pending operations in the L2 cache unit have completed.
4. Initiate the global invalidation operation by setting the L2CR[L2I] bit.
5. Monitor the L2CR[L2IP] bit to determine when the global invalidation operation is completed (indicated by the clearing of L2CR[L2IP]). The global invalidation requires approximately 16K core clock cycles to complete.
6. After detecting the clearing of L2CR[L2IP], clear L2CR[L2I] and re-enable the L2 cache for normal operation by setting L2CR[L2E].

3.7.3.8 L2 Cache Flushing

In the MPC7410, the L2 cache is a victim cache for the L1 data cache. As such, the L2 cache flush routines used for MPC750-based systems will not work on the MPC7410. The MPC7410 provides a hardware flush mechanism through L2CR[HWF]. This hardware flush method is the recommended method for flushing the L2 cache. Although the hardware flush mechanism is the preferred method of flushing the cache, if for some reason a software flush is desired, the MPC7410 provides a software flush assist bit L2CR[L2FA] to facilitate software flushing of the L2 cache. The following sections describe flushing the L2 cache using the hardware and software methods.

3.7.3.8.1 L2 Cache Hardware Flush

The hardware flush mechanism is controlled by L2CR[L2HWF]. When the processor detects a state transition from 0 to 1 in L2HWF, the MPC7410 initiates a hardware flush of the L2 cache.

The flush is performed by starting with low cache indices and increments through way 0 of the cache one index at a time until the maximum index value is obtained. Then, the index is reset to zero and the same process is repeated for way 1 of the L2 cache. For each index and way of the cache, the processor generates a castout operation to the system bus for all modified cache blocks. At the end of the hardware flush, all lines in the L2 cache tags are in the invalid state.

During the flush, all memory activity from the L1 instruction and L1 data cache are blocked from accessing the L2 until the flush is complete. Snoops, however, are fully serviced by the L2 cache during the flush.

When the L2 cache tags have been fully flushed of all valid entries, the L2CR[L2HWF] bit is cleared by hardware. Note that when L2HWF is cleared, it does not guarantee that all lines from the L2 have been written completely to the system interface. L2 copybacks may still be queued up in the bus interface unit. A final **sync** instruction is required to guarantee that all data from the L2 cache has been written to the system address bus.

The recommended sequence to flush the L2 cache follows:

1. disable interrupts
2. **dssall**
3. **sync**
4. set L2CR[L2HWF] = 1
5. **sync**

The L2 cache hardware flush mechanism is not present in earlier microprocessor implementations. Using L2CR[L2HWF] is the preferred mechanism for flushing the L2 cache on the MPC7410.

3.7.3.8.2 L2 Cache Software Flush

There are a variety of methods to flush the L2 cache using load, **dcbz**, **dcbf**, or AltiVec **stvxl** instructions. The L2 cache flush assist bit, L2CR[L2FA], simplifies the software flushing process. In normal (non-flushing) operations, L2FA is cleared and all lines are cast out from the L1 data cache that have a status of CDMRSV = 01xxx1 (that is, the C bit is negated), does not allocate in the L2 cache if they miss. However, when set, L2FA forces every castout from the L1 data cache to allocate an entry in the L2 cache if that castout misses in the L2 regardless of the state of the C bit.

L2FA should be set just prior to the beginning of the cache flush routine and cleared after the series of instructions is complete. The address space should not be shared with any other process to prevent snoop hit invalidations during the flushing routine. Exceptions should be disabled during this time so that the FIFO replacement logic is not disturbed.

The following procedure is an efficient L2 cache software flush algorithm using **stvxl**:

1. Set HID0[DCFA]
2. Set L2CR[L2FA] and clear L2CR[L2IO]
3. Set L2CR[L2DO] (to prevent instruction reloads of the L2)

4. Disable all interrupts (to avoid disturbing cache replacement pointers)
5. Execute three uniquely addressed **stvxl** instructions to each 32-byte block of the L2 cache. The three stores must be to the same L2 index (that is, bits 12–26 of the physical address must be equal). The following pseudo-C code provides an example of how to do this. Note that this example assumes data translation is disabled (MSR[DR] = 0):

```
r1=0x00000000;/* r1, r2, and r3 can be any values as long */
r2=0x10000000;/* as bits 12-26 are the same for all three   */
r3=0x20000000;/* and bits 0-11 are different between all three */
r4=0x0;
r5=0x10;
for (i=0; i<L2_SIZE_IN_BYTES / 32; i++) {
        stvxl   r0, r1, r4; stvxl r0, r1, r5;
        stvxl   r0, r2, r4; stvxl r0, r2, r5;
        stvxl   r0, r3, r4; stvxl r0, r3, r5;
        r4 += 0x20; r5 +=0x20;}
```

The second store to each cache block (using r5) is for performance reasons. The MPC7410 merges the entire 32-byte cache block for each **stvxl** pair. If the stores are mapped global (M = 1), then the stores perform address-only kill transactions on the bus because they merge to the full 32-byte cache block. If the stores are mapped non-global (M = 0), then the stores merge to 32 bytes and silently allocate in the L1 data cache. See Section 3.6.5, “Store Miss Merging,” for more information on store miss merging, Note that this algorithm does not require knowledge of how the L2 cache is sectored for each size configuration and works for all L2 sizes.

3.7.3.9 L2 Cache Clock and Timing Controls

The L2CR[L2CLK] parameter specifies the operating frequency for the L2 data RAM interface. This is expressed as a clock divider ratio relative to the MPC7410’s core clock frequency. When cleared to all 0s, the on-chip DLL for the L2 interface is disabled (and held in reset), and the L2 clock outputs are turned off. When set to a non-zero value, the on-chip DLL is enabled, and the L2 clocks are generated. After setting the L2 clock ratio, a period of time must elapse for the DLL to stabilize before enabling the L2 interface. See the MPC7410 hardware specifications for more information.

The L2CR[L2OH] parameter determines the output hold time of the address, data, and control signals driven by the MPC7410 to the L2 data RAMs. L2OH should generally be set according to the input hold time requirements of the SRAMs in the system. Typically burst RAMs require an input hold time of 0.5 ns, and late-write RAMs require an input hold time of 1.0 ns. See the MPC7410 hardware specifications for more information.

The L2CR[L2SL] parameter is used to slow down the L2 bus interface by increasing the delay through the DLL. Setting L2SL increases the delay of each tap of the DLL delay line. It is intended to slow down the L2 bus interface to accommodate slower L2 bus frequencies. L2SL should generally be set if the L2 RAM interface is being operated at lower frequencies. See the MPC7410 hardware specifications for more information.

The L2CR[L2DF] parameter controls the behavior of the L2 clock output signals. Setting L2DF configures the two L2 clock outputs, L2CLK_OUTA, and L2CLK_OUTB, to operate as a differential clock pair (L2CLK_OUTA/L2CLK_OUTB). In this mode, the B clock is driven as the logical complement of the A clock. This mode is provided to support late-write SRAMs, many of which require a differential clock.

The L2CR[L2BYP] parameter is intended for use when the PLL is being bypassed, and for engineering evaluation. The DLL requires the following three input clocks:

- An internal square wave clock from the PLL to phase adjust and export
- An internal non-square wave clock for the internal phase reference
- A feedback clock (L2SYNC_IN) for the external phase reference

When L2BYP is set, the MPC7410 uses the non-square wave clock (#2) for both phase adjust and phase reference (#1 and #2) thus bypassing the square wave clock from the PLL. Note that the non-square wave clock (#2) is the actual clock used by the MPC7410's L2 interface circuitry. If the PLL is being bypassed, the DLL must operate in 1:1 mode, and SYSCLK must be fast enough for the DLL to support.

3.7.3.10 L2 Cache Power Management and Test Controls

The L2CR[L2CTL] parameter enables/disables automatic operation of the L2 low-power mode signal, L2ZZ, for cache RAMs that support the ZZ function. When L2CTL is set, the MPC7410 automatically asserts L2ZZ when entering nap or sleep mode, and automatically negates L2ZZ when exiting nap or sleep. L2CTL should not be set when the MPC7410 is in nap mode and dynamic snooping is being performed through negation of $\overline{\text{QACK}}$. The relatively long recovery time from ZZ negation that many SRAM vendors require may only allow use of this function for deep-sleep operation.

The L2CR[L2CLKSTP] parameter controls automatic stopping of the L2 clock output signals for cache RAMs that support this function. When L2CLKSTP is set, the L2 clock output signals automatically stop when the MPC7410 enters nap or sleep mode, and automatically restart when the MPC7410 exits nap or sleep.

The L2CR[L2TS] parameter is provided to support L2 cache testing. See Section 3.7.9, "L2 Cache Testing," for more information.

The L2CR[L2DRO] parameter controls the behavior of the MPC7410 when it encounters a potential (or actual) DLL rollover. A potential rollover condition occurs when the DLL selects the last tap of the delay line and risks rolling over to the first tap while trying to keep in sync. Such a condition is improper operation for the DLL, and while this condition is not expected, L2DRO allows detection for added security. Setting L2DRO causes a checkstop when a potential (or actual) rollover condition occurs. L2DRO may be set when the DLL is first enabled (set with the L2CLK bits) to detect rollover during initial synchronization. It may also be set when the L2 cache is enabled (with L2E bit) after the DLL has achieved initial lock.

3.7.4 L2 Private Memory Control Register—MPC7410 Only

The L2 private memory control register (L2PMCR) allows control of the private memory feature of the L2 cache interface of the MPC7410. Note that the MPC7400 does not support the private memory feature and does not implement the L2PMCR. The L2PMCR is a supervisor-level read/write, implementation-specific register that is accessed as SPR 1016. The contents of the L2PMCR are cleared during power-on reset. See Section 2.1.5.4.1, "L2 Private Memory Control Register (L2PMCR)—MPC7410 Only," for additional information about programming the L2CR.

The L2PMCR includes the following parameters:

- PMEN—enables or disables the private memory function.
- PMBA—defines the starting address of the private memory space, aligned to the appropriate block size. The number of bits that are actually used to determine a hit is dependent on L2PMCR[PMSIZ]. If the upper bits of a physical address of a load, store or cache operation match the PMBA, the data is read or written from the external SRAMs regardless of the state of the WIMG bits. Note that transactions that hit in the private memory space are not visible on the external system bus.
- DBSIZ—configures the size (32- or 64-bits) of the external SRAM data bus. Note that the MPC7400 does not support a 32-bit L2 data bus. In 32-bit L2 data bus mode, the MPC7410 uses the high-order L2 data signals (L2DATA[0:31]) for L2 data; the low-order data signals (L2DATA[32:63]) are not sampled for reads and are driven low for writes. Note that PB3-type SRAMs cannot use 32-bit L2 data bus mode because the MPC7410 restricts accesses to PB3 SRAMs to 4-beat bursts.
- PMSIZ—configures the size of the private memory space. It is possible to simultaneously use one half of the available external data SRAM as private memory space and the other half as L2 cache. Table 3-14 describes the allowed combinations for L2 cache and private memory.

Table 3-14. L2 Cache/Private Memory Configurations

Total SRAM Space	All L2 Cache	Half L2 Cache and Half Private Memory	All Private Memory

256 Kbytes	L2CR
	L2E =0b1
	L2SIZ = 0b01 (256 Kbytes)
	L2PMCR
	PMEN = 0b0
	PMSIZ = N/A
		Not Supported	L2CR
			L2E = 0b0
			L2SIZ = N/A
			L2PMCR
			PMEN = 0b1
			PMSIZ = 0b01 (256 Kbytes)

512 Kbytes	L2CR
	L2E = 0b1
	L2SIZ = 0b10 (512 Kbytes)
	L2PMCR
	PMEN = 0b0
	PMSIZ = N/A
		L2CR
		L2E = 0b1
		L2SIZ = 0b01 (256 Kbytes)
		L2PMCR
		PMEN = 0b1
		PMSIZ = 0b01 (256 Kbytes)
			L2CR
			L2E = 0b0
			L2SIZ = N/A
			L2PMCR
			PMEN = 0b1
			PMSIZ = 0b10 (512 Kbytes)

1 Mbyte	L2CR
	L2E = 0b1
	L2SIZ = 0b11 (1 Mbyte)
	L2PMCR
	PMEN = 0b0
	PMSIZ = N/A
		L2CR
		L2E = 0b1
		L2SIZ = 0b10 (512 Kbytes)
		L2PMCR
		PMEN = 0b1
		PMSIZ =0b10 (512 Kbytes)
			L2CR
			L2E = 0b0
			L2SIZ =N/A
			L2PMCR
			PMEN = 0b1
			PMSIZ = 0b11 (1 Mbyte)

2 Mbytes	L2CR
	L2E = 0b1
	L2SIZ = 0b00 (2 Mbytes)
	L2PMCR
	PMEN = 0b0
	PMSIZ = N/A
		L2CR
		L2E = 0b1
		L2SIZ = 0b11 (1 Mbyte)
		L2PMCR
		PMEN = 0b1
		PMSIZ = 0b11 (1 Mbyte)
			L2CR
			L2E = 0b0
			L2SIZ = N/A
			L2PMCR
			PMEN = 0b1
			PMSIZ = 0b00 (2 Mbytes)

Note that any combination not shown in Table 3-14 is not allowed.

3.7.5 L2 Cache Initialization

Following a power-on or hard reset, the L2 cache and the L2 cache DLL are disabled initially. Before enabling the L2 cache, the L2 cache DLL must first be configured through the L2CR register, and the DLL must be allowed 640 L2 cache clock periods to achieve phase lock. Before enabling the L2 cache, other configuration parameters must be set in the L2CR, and the L2 cache tags must be globally invalidated. The L2 cache should be initialized during system start-up.

The sequence for initializing the L2 cache is as follows:

1. Power-on reset (automatically performed by the assertion of $\overline{\text{HRESET}}$).
2. Disable L2 cache by clearing L2CR[L2E].
3. Set the L2CR[L2CLK] bits to the desired clock divider setting. Setting a nonzero value automatically enables the DLL. All other L2 cache configuration bits should be set to properly configure the L2 cache interface for the SRAM type, size, and interface timing required.
4. Wait for the L2 cache DLL to achieve phase lock. This can be timed by setting the decrementer for a time period equal to 640 L2 cache clocks, or by performing an L2 cache global invalidate.
5. Perform an L2 cache global invalidate. The global invalidate could be performed before enabling the DLL, or in parallel with waiting for the DLL to stabilize. Refer to Section 3.7.3.7, "L2 Cache Global Invalidation," for more information about L2 cache global invalidation. Note that a global invalidate always takes much longer than it takes for the DLL to stabilize.
6. After the DLL stabilizes, an L2 cache global invalidate has been performed, and the other L2 cache configuration bits have been set, enable the L2 cache for normal operation by setting the L2CR[L2E] bit to 1.

3.7.6 L2 Cache Operation

The MPC7410's L2 cache is a combined instruction and data cache that receives memory requests from both L1 instruction and data caches independently. The L1 requests are generally the result of instruction fetch misses, data load or store misses, L1 data cache castouts, write-through operations, or cache management instructions. Each L1 request generates an address lookup in the L2 cache tags. If a hit occurs, the instructions or data are forwarded to the appropriate L1 cache. A miss in the L2 cache tags causes the L1 request to be forwarded to the system bus interface. The L2 cache also services snoop requests from the system bus.

Generally, the L2 cache operates according to the following rules:

- In case of multiple pending requests to the L2 cache, snoop requests have the highest priority. The next priority is a data cache reload, unless there is an address conflict with an L1 data cache castout. In this case, the L1 castout will have higher priority. This insures that reads and writes to the same cache block are kept in order. The lowest priorities are instruction fetches from the L1 instruction cache and L2 instruction reloads.
- All requests to the L2 cache that are marked caching-inhibited bypass the L2 cache (even if they would have normally hit), and do not cause any L2 tag state changes.
- Requests to the L2 cache that are marked caching-allowed (even if the respective L1 cache is locked) are serviced by the L2 cache. Caching-allowed burst requests are serviced in their entirety.

Caching-allowed single-beat requests are allowed to hit and update in case of a store hit, but do not cause allocation or deallocation. Note that these comments apply only if the cache disabling conditions of Section 3.7.3.1, “Enabling and Disabling the L2 Cache,” are met.

- Burst read and single-beat read requests from the L1 instruction or data caches that hit in the L2 cache are forwarded data from the L2 SRAMs.
- Burst read requests from the L1 instruction or data caches that miss in the L2 cache will initiate a burst read operation from the system interface for the cache block that missed. The cache block that is received from the bus is forwarded to the appropriate L1 cache.
- Normal burst writes from the L1 data cache due to castouts (also referred to as replacement copybacks) are written to the L2 cache with the same state (MERSI) information as they had in the L1. If the L2 is configured as write-through (L2WT = 1), they are marked exclusive instead, and are also forwarded to the system interface. If the L1 castout requires a new tag entry to be allocated in the L2 cache and the current tag is modified, any modified sectors of the tag to be replaced are castout from the L2 cache to the system interface, unless the C bit of the L1 cast out is clear. If the C bit is clear, and the L1 castout misses in the L2, it does not allocate a new entry and is forwarded to the system interface. If a new tag is allocated, the F-bit is updated to point to the other cache way. Note that setting the L2IO bit of the L2CR forces the C bit of all L1 castouts to be cleared. In this case, L1 castouts will never allocate in the L2.
- Normal burst writes to the L2, on behalf of instruction cache misses that cause L2 allocates, are written to the L2 with the state (RSI) information obtained from the system interface. If this write ever hits in the L2 (due to data and instructions occupying the same block), then it is discarded.
- Normal single-beat writes (not **stwcx.**) that are marked write-through (by address translation or because the L1 cache is locked) are written to the L2 cache if they hit, and they are also written to the system interface independent of L2 hit/miss status. In case of a hit to a line in the L2 not marked modified, the status (MERSI) information and F-bit remain unchanged. In case of a hit to a line in the L2 that is marked modified, the entire line is pushed to memory and the state is changed to exclusive. The F-bit remains unchanged.
- Caching-allowed **stwcx.** operations are handled by the L1 data cache similarly to normal caching-allowed stores. The L2 cache does not treat **stwcx.** differently than a normal caching-allowed store. Caching-inhibited **stwcx.** operations do not access the L2 tags and are forwarded to the system interface.
- The **dcbz** instruction does not affect the L2 cache state. The **dcbz** instruction is handled entirely by the L1.
- On the MPC7410, **dcba** differs from **dcbz** only in its exception generation. As such, it is identical to **dcbz** from an L2 perspective. The **dcba** instruction does not affect the L2 cache state.
- A **dcbf** instruction is issued to the L2 cache after being processed by the L1 data cache. If a **dcbf** hits in the L2 cache, it invalidates the block. If the **dcbf** requires a cache block push from the L1 data cache, the push is forwarded to the system interface. If the **dcbf** does not require a cache block push from the L1 data cache, and hits on a block marked modified in the L2 cache, the L2 pushes the data to the system interface. In either case, if the cache block existed in the L2, it is marked invalid. If the **dcbf** is marked global, it is forwarded to the system interface.
- A **dcbst** instruction is issued to the L2 cache after being processed by the L1 data cache. If the **dcbst** requires a cache block push in the L1 data cache, this data is written to the L2, the cache

block is marked exclusive, and the push is forwarded to the system interface. If the **dcbst** does not require a cache block push from the L1 data cache, and the cache block is modified in the L2 cache, the L2 pushes the data to the system interface and marks the cache block exclusive. If the **dcbst** misses in the L2 cache and is marked global, it is forwarded to the system interface.

- A **dcbi** instruction is always issued to the L2 cache, and causes the cache block to be invalidated in the L2 in case of a hit. A **dcbi** instruction is also issued to the system interface if they are marked global.
- The **icbi** instruction never affects the L2 cache. All **icbi** instructions are passed to the system interface.
- **sync**, **eieio**, **eciwx**, **ecowx**, **tlbi**, and **tlbsync** instructions bypass the L2 cache, and are forwarded to the system interface for further processing.

3.7.6.1 L2 Cache Allocation on Cache Misses

The L2 cache is a victim cache for the L1 data cache. The L2 cache allocates new entries for data accesses only when blocks are cast out of the L1 data cache. When a block is queued up as a data cache castout and the L2 cache is enabled, the L2 cache allocates a new tag for the castout in the L2 cache if it misses and the C bit is set. If the C bit is cleared and the block misses in the L2 cache, the L2 cache does not allocate a tag. Instead, it passes the castout to the system interface if the cache block is marked modified. If the data cache castout hits in the L2 cache, the castout data is written to the L2 cache regardless of the state of the C bit.

If the L2 cache is disabled, then the block replaced from the L1 data cache is cast out to the system interface if the cache block is marked modified.

3.7.6.2 L2 Cache Replacement Selection

The L2 cache uses a least-recently used (LRU) replacement algorithm. L2 cache victims are selected based on the FIFO replacement bit (F-bit) in the cache tags. When an L1 data cache castout or L1 instruction cache reload allocates a new tag in the L2 cache, the F bit is updated to point to the other cache way. L2 cache victim selection is performed at reload time, not at demand-miss time.

3.7.6.3 Store Hit to a Shared or Recent L2 Cache Block

If a write-back store misses in the L1 data cache but hits on an L2 cache block in the shared or recent state, the L2 cache provides the cache block to the reload data buffer. A kill operation is then propagated to the system bus. The reload data buffer treats the entry as a hit-on-shared/hit-on-recent and waits for the bus to complete the kill broadcast before reloading the data cache.

As in the data cache hit-on-shared/hit-on-recent case, if a snoop operation invalidates ownership of the target block before the kill operation is successful, the reload buffer entry changes to treat the entry like a normal store miss. In this case, the MPC7410 performs a RWITM operation on the address bus instead and reloads the L1 data cache in the modified state.

3.7.7 Private Memory Operation—MPC7410-Only

The L2 interface for the MPC7410 also supports operation as a private memory. This feature allows the MPC7410 to have access to a low latency, high bandwidth private memory space. Note that the MPC7400 does not implement the private memory feature. The private memory space is not snooped and therefore is not coherent with other processors in a system. The private memory space can contain instructions and data and its contents can be cached in the L1 instruction and data caches provided that accesses are marked caching-allowed. Note that instructions in the L2 private memory space should not be marked as caching-inhibited.

Private memory receives requests from both the L1 instruction cache and the L1 data cache independently. The L1 requests are generally the result of instruction misses, data load or store misses, L1 data cache castouts, write-through operations, or cache management instructions. The L1 requests are looked-up in the L2 tags and compared with the proper bits in L2PMCR[PMBA]. If a match with PMBA is determined, the result of the L2 tag lookup is ignored and the request is forwarded to the external SRAM interface.

The private memory space can be initialized by a sequence of program load instructions from system memory and program store instructions to the private memory space.

The private memory space does not have coherency state information. When the L1 data cache is reloaded for a caching-allowed load or store, the state of the block is either exclusive (for a load) or modified (for a store).

All transactions that read or write data except **eciwx** and **ecowx** are allowed to hit in the private memory space, regardless of address translation (WIMG memory/cache access attributes). The **icbi**, **sync**, **tlbie**, **tlbsync**, **eieio**, **eciwx**, and **ecowx** instructions never hit in the private memory space and are forwarded to the system interface. Any **dcbi** instructions that hit in the private memory space are discarded (after invalidating the L1 data cache).

Performance monitor events related to the L2 cache may not produce expected results when private memory is enabled. Specifically, hits to the private memory space are treated as L2 cache misses by the performance monitor. There are no new performance monitor events that specifically support the private memory feature.

Generally, the private memory operates according to the following:

- Arbitration is shared with the L2 cache and thus uses the same priorities.
- Requests to the L2 interface that are marked caching-allowed by address translation (even if the respective L1 cache is locked) are serviced by the L2 interface if they hit in the private memory space.
- Burst read and single-beat read requests from the L1 instruction or data caches that hit in the private memory space are forwarded data from the L2 SRAMs.
- Burst read requests from the L1 instruction or data caches that miss in both the private memory space and the L2 cache initiate a burst read operation on the system interface for the cache block that missed. The cache block that is received from the system bus is forwarded to the appropriate L1 cache. L1 instruction cache misses are also allocated into the L2. If the L2 allocate requires a new tag entry and the current tag is dirty (M), any dirty sectors of the tag to be replaced are castout from the L2 cache to the system interface, and the FIFO replacement bit (F-bit) is updated to point

to the other cache way. If the L2 cache is disabled (L2CR[L2E] = 0) or set for data only mode (L2CR[L2DO] = 1), then L1 instruction cache misses are not allocated into the L2.

- Normal burst writes from the L1 data cache due to castouts (also referred to as replacement copybacks) that hit in the private memory space are written to the external SRAMs regardless of the L1 status bits (CDMRSV), or the L2CR[L2IO] parameter. Burst writes that miss in the private memory space are allocated in the L2 cache (if enabled).
- Caching-allowed **stwcx.** operations are handled by the L1 data cache similarly to normal caching-allowed stores. The L2 interface does not treat **stwcx.** differently than a normal caching-allowed store. Caching-inhibited **stwcx.** operations that hit in the private memory space write the appropriate data to the L2 SRAMs and are not forwarded to the system interface.
- **dcbz** operations that hit in the private memory space do not affect the data in the external SRAMs. They are handled entirely by the L1 and are not forwarded to the system interface.
- **dcbf** operations are issued to the L2 interface after being processed by the L1 data cache. If a cache block push due to a **dcbf** that hits modified data in the L1 data cache hits in the private memory space, the cache block is written to the L2 SRAMs. **dcbf** operations that hit in the private memory space are never forwarded to the system interface.
- **dcbst** instructions are issued to the L2 cache after being processed by the L1 data cache. If a cache block push due to a **dcbst** that hits modified data in the L1 data cache hits in the private memory space, the cache block is written to the L2 SRAMs. **dcbst** operations that hit in the private memory space are never forwarded to the system interface.
- **dcbi** instructions that hit in the private memory space are discarded and are never forwarded to the system interface.
- **icbi** instructions never affect the L2 interface. They are passed to the system interface for further processing.
- **sync**, **eieio**, **eciwx**, **ecowx**, **tlbie**, and **tlbsync** instructions pass though the L2 interface and are forwarded to the system interface for further processing.

3.7.8 L2 Cache Clock Configuration

The MPC7410 provides a programmable clock for the L2 cache external synchronous data RAM. The clock frequency for the external SRAM is provided by dividing the MPC7410's internal clock by ratios of 1, 1.5, 2, 2.5, 3, 3.5, or 4 programmed through the L2CR[CLK] bit. The L2 cache clock is phase-adjusted to synchronize the clocking of the latches in the MPC7410's L2 cache interface with the clocking of the external SRAM by means of an on-chip delay-locked loop (DLL).

The ratio selected for the L2 cache clock is dependent on the frequency supported by the external SRAMs, the MPC7410's internal operation frequency, and the range of phase adjustment supported by the L2 cache DLL. Refer to the MPC7410 hardware specifications for additional information about L2 cache clock configuration.

3.7.9 L2 Cache Testing

In the course of system power-up, testing may be required to verify proper operation of the L2 cache tags, external SRAMs, and overall L2 cache system. This section describes features and methods for testing the L2 cache.

L2CR[L2DO] and L2CR[L2TS] support the testing of the L2 cache. L2CR[L2DO] prevents instructions from being cached in the L2 cache. This allows the L1 instruction cache to remain enabled during the testing process without having L1 instruction cache misses affect the contents of the L2 cache and allows all L2 cache activity to be controlled by program-specified load and store operations.

L2CR[L2TS] is used with the **dcbf** and **dcbst** instructions to push data into the L2 cache. When L2TS is set, **dcbf** pushes from the L1 data cache are allocated in the L2 cache (rather than stored to the system bus as with normal **dcbf** operations) and all **dcbz** operations are treated as non-global (to suppress address broadcasts). In addition, write-through stores are not forwarded to the system interface. Write-through stores that hit in the L2 cache update the cache data RAMs. L2TS allows general testing of the L2 cache data RAMs and tags by allowing a **dcbz**/**dcbf** loop to initialize the L2 cache with address and data information, and then allowing various read/write operations to test the L2 cache data RAMs and/or tags. Note that due to the influence of L2TS on the replacement algorithm, it is necessary to initialize an address range that is twice (2x) the physical L2 cache size and perform testing on the second half of that address range.

3.7.9.1 Testing Overall L2 Cache Operation

One method for testing overall L2 cache operation is to enable the caches for normal operation and run a comprehensive program designed to exercise all the caches, including L2 reload and castout activity. The performance monitors may be used to monitor hits, misses, and castouts of cacheable operations. Note that performance monitor events related to the L2 cache may not produce expected results when private memory is enabled. Therefore, private memory should be disabled during the following L2 test procedures.

3.7.9.2 Testing L2 Cache External SRAMs

The L2 cache external SRAMs may be tested using the following procedure:

1. Disable address translation (MSR[DR] = 0) to invoke the default WIMG setting of 0b0011.
2. Set L2CR[L2DO] and L2CR[L2TS], and perform a global invalidation of the L1 data cache and the L2 cache. The L1 instruction cache can remain enabled to improve execution efficiency.
3. Enable the L2 cache and the L1 data cache.
4. Execute a series of **dcbz** and **dcbf** instructions to initialize the cache with a sequential range of addresses and with cache data consisting of zeroes. The range of addresses must be twice the physical L2 cache size. Although the L2 cache is in data-only mode at this point, instruction accesses may still hit in the L2 cache, so ensure that the sequential range of addresses selected does not overlap with any existing instruction address space.
5. Invalidate and lock the L1 data cache.

6. Perform a series of store and load operations using a variety of non-zero bit patterns to test for stuck bits and pattern sensitivities in the L2 cache SRAMs. These loads and stores should be in the second half of the range of addresses used to initialize the caches in step 4 so that each access hits in the L2 cache.

3.7.9.3 Testing L2 Cache Tags

The L2 cache internal tags may be tested using the following procedure:

1. Disable address translation (MSR[DR] = 0) to invoke the default WIMG setting of 0b0011.
2. Set L2CR[L2DO] and L2CR[L2TS], and perform a global invalidation of the L1 data cache and the L2 cache. The L1 instruction cache can remain enabled to improve execution efficiency.
3. Enable the L2 cache and the L1 data cache.
4. Execute a series of **dcbz** and **dcbf** instructions to initialize the cache with a sequential range of addresses and with cache data consisting of zeroes. The range of addresses must be twice the physical L2 cache size. Although the L2 cache is in data-only mode at this point, instruction accesses may still hit in the L2 cache, so ensure that the sequential range of addresses selected does not overlap with any existing instruction address space.
5. Invalidate and lock the L1 data cache.
6. Perform a series of non-zero stores to a range of addresses not currently in the L2 cache. Each of these stores should miss.
7. Initialize the performance monitor counters to zero, and set the MMCR registers to count the number of L2 cache hits.
8. Perform a series of reads from the second half of the original range of addresses located in the cache and verify that the data read was not affected by the stores performed in step 6. For accurate reporting of the number of hits, only one load per cache block should be performed.
9. Disable the performance monitor counters and verify that the number of hits matches the accesses performed by the test program. All accesses to the second half of the original region should hit.

Note that when running these cache tests, the performance monitor counters can only be used to count load hits/misses in the L2 cache. Hits or misses that result from stores cannot be counted. This is due to the L1 data cache being locked during the test procedure, which means that data store operations are treated as write-through. Loads are treated as cacheable when the L1 data cache is locked, and can therefore be counted by the performance monitors.

3.7.10 L2 Cache SRAM Timing Examples

This section describes the signal timing for the three types of SRAM (pipelined burst SRAM, late-write SRAM, and PB3 SRAM) supported by the MPC7410's L2 cache interface. The timing diagrams illustrate the best case logical (ideal, non AC-timing accurate) interface operations. For proper interface operation, the designer must select SRAMs that support the signal sequencing illustrated in the timing diagrams.

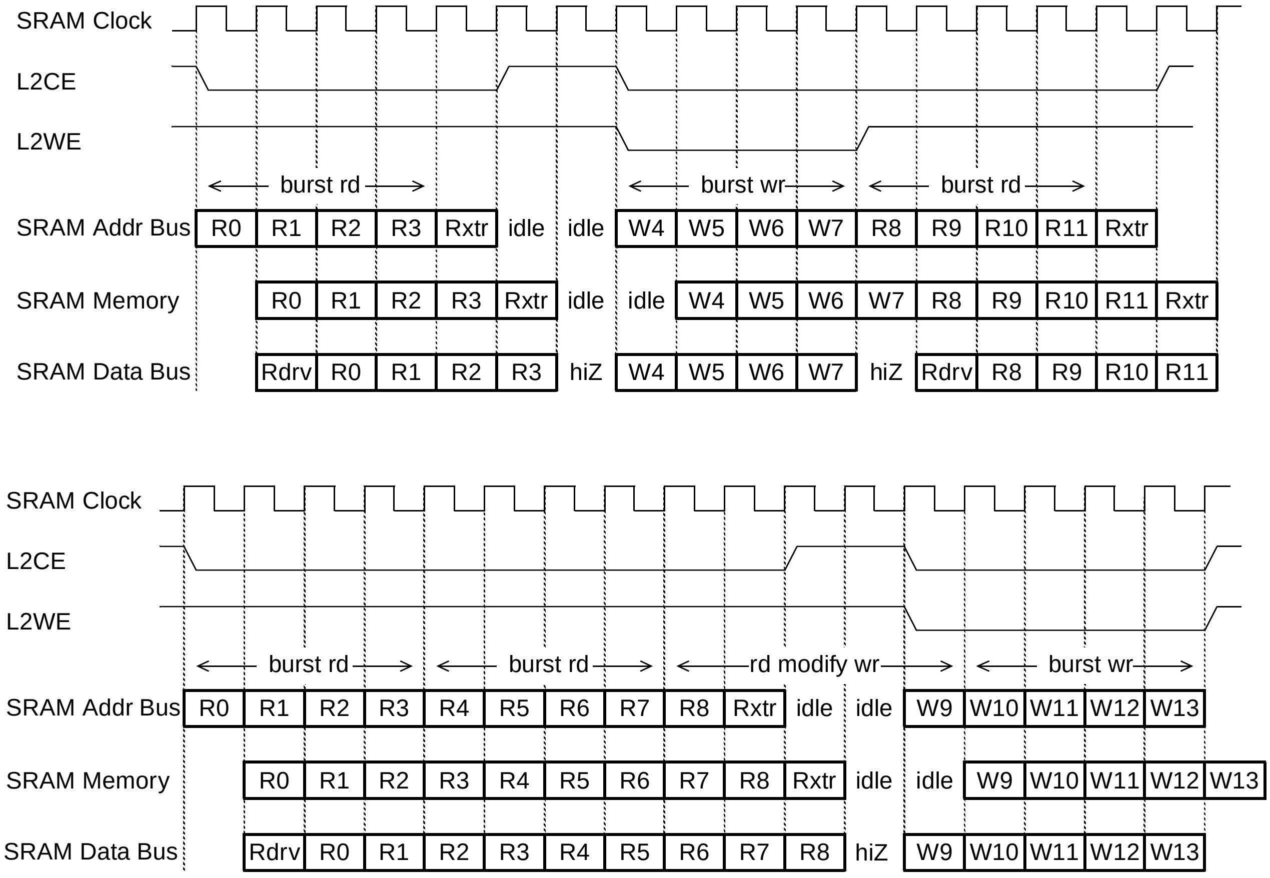
The SRAM selected for a system design is usually a function of desired system performance, L2 cache bus frequency, and SRAM unit cost. The following sections describe the operation of the three SRAM types supported by the MPC7410, and the design trade-offs associated with each.

3.7.10.1 Pipelined Burst SRAM

Pipelined burst SRAMs are sometimes referred to as PB2 (pipelined burst, 2nd generation) SRAMs to distinguish them from PB3 SRAMs. Pipelined burst SRAMs operate by clocking read data from the memory array into a buffer before driving the data onto the data bus. This causes an extra clock cycle of latency for initial read accesses, but the L2 cache bus frequencies supported can be higher. Note that the MPC7410's L2 cache interface requires the use of single-cycle deselect pipelined burst SRAM for proper operation.

Note that during burst transfers into and out of the SRAM array, the MPC7410 generates an address for each data beat. That is, the MPC7410 does not use the burst feature (one address, many data beats) of the pipelined burst SRAMs.

Figure 3-36 shows memory access timings when the L2 cache interface is configured for pipelined burst SRAM and a 64-bit data bus. The timing for a 32-bit L2 data bus is identical except that there are eight data beats for a transaction in 32-bit L2 data bus mode compared to the four data beats shown in Figure 3-36 for the 64-bit L2 data bus. The control signal behaviors and general sequencing are unchanged between 32- and 64-bit L2 data bus modes.



SRAM Clock
L2CE
L2WE
burst rd
burst wr
burst rd
SRAM Addr Bus
R0 R1 R2 R3 Rxtr idle idle W4 W5 W6 W7 R8 R9 R10 R11 Rxtr
SRAM Memory
R0 R1 R2 R3 Rxtr idle idle W4 W5 W6 W7 R8 R9 R10 R11 Rxtr
SRAM Data Bus
Rdrv R0 R1 R2 R3 hiZ W4 W5 W6 W7 hiZ Rdrv R8 R9 R10 R11
SRAM Clock
L2CE
L2WE
burst rd
burst rd
rd modify wr
burst wr
SRAM Addr Bus
R0 R1 R2 R3 R4 R5 R6 R7 R8 Rxtr idle idle W9 W10 W11 W12 W13
SRAM Memory
R0 R1 R2 R3 R4 R5 R6 R7 R8 Rxtr idle idle W9 W10 W11 W12 W13
SRAM Data Bus
Rdrv R0 R1 R2 R3 R4 R5 R6 R7 R8 hiZ W9 W10 W11 W12 W13

Notes: Rdrv indicates where some burst RAMs may begin driving the data bus.
Rxtr indicates where an extra read cycle is signaled to keep the burst RAM driving the data bus for the last read. The MPC7410 does not support aborted reads

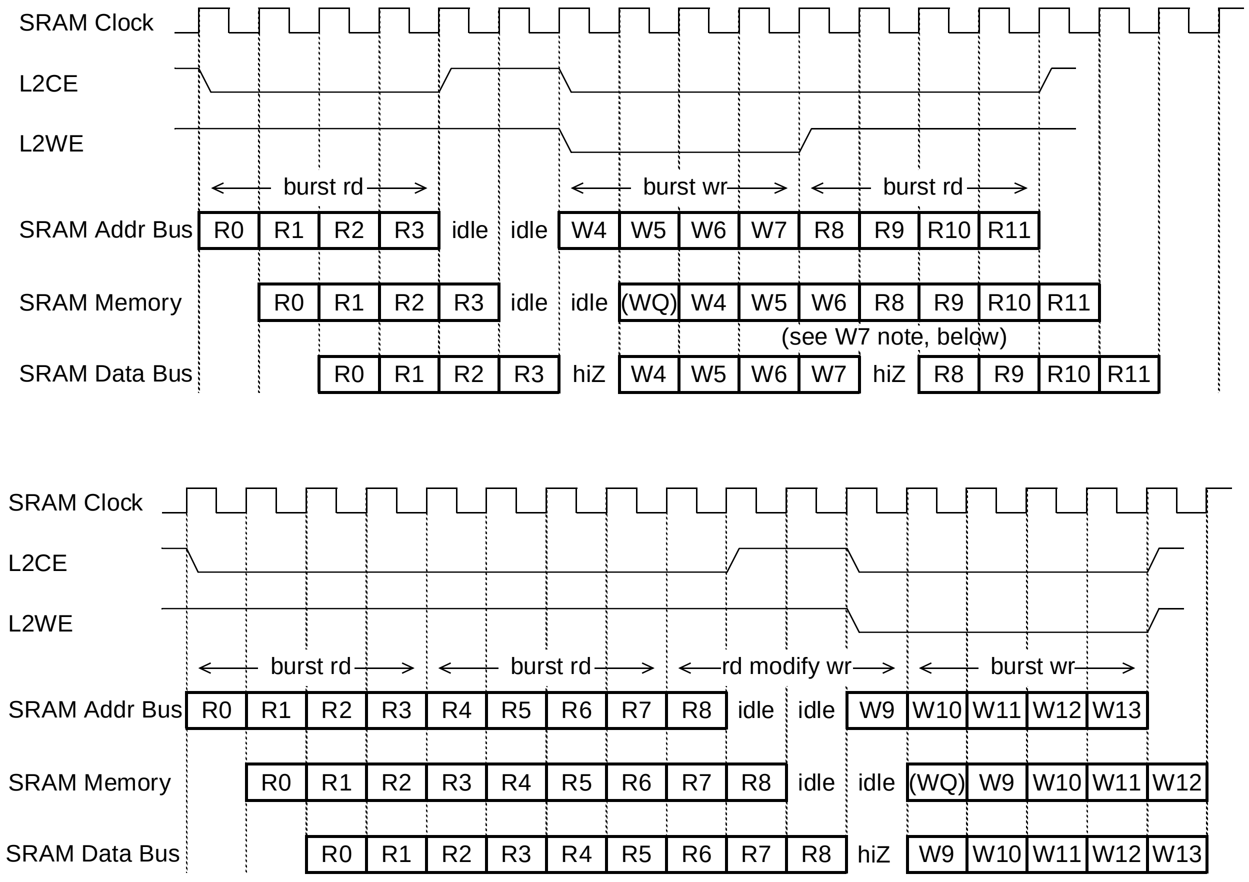
Figure 3-36. Pipeline Burst SRAM Timing

3.7.10.2 Late-Write SRAM

Late-write SRAMs offer improved performance when compared to pipelined burst SRAMs by not requiring an extra read cycle during read operations, and requiring one cycle less when transitioning from a read to a write operation. Late-write SRAMs implement an internal write queue, allowing write data to be provided one cycle after the write operation is signaled on the address and control buses. In this manner, write operations are queued on the address and data bus in the same manner as read operations, allowing transitions between read and write operations to occur more efficiently.

Note that during burst transfers into and out of the SRAM array, the MPC7410 generates an address for each data beat. That is, the MPC7410 does not use the burst feature (one address, many data beats) of the late-write SRAMs.

Figure 3-37 shows memory access timings when the L2 cache interface is configured for late-write SRAM and a 64-bit data bus. The timing for a 32-bit L2 data bus is identical except that there are eight data beats for a transaction in 32-bit L2 data bus mode compared to the four data beats shown in Figure 3-37 for the 64-bit L2 data bus. The control signal behaviors and general sequencing are unchanged between 32- and 64-bit L2 data bus modes.



Note: WQ is the last previous write that was queued in the late-write RAM.
Note also: W7 is queued in the late-write device and will not appear in SRAM Memory until the next write.

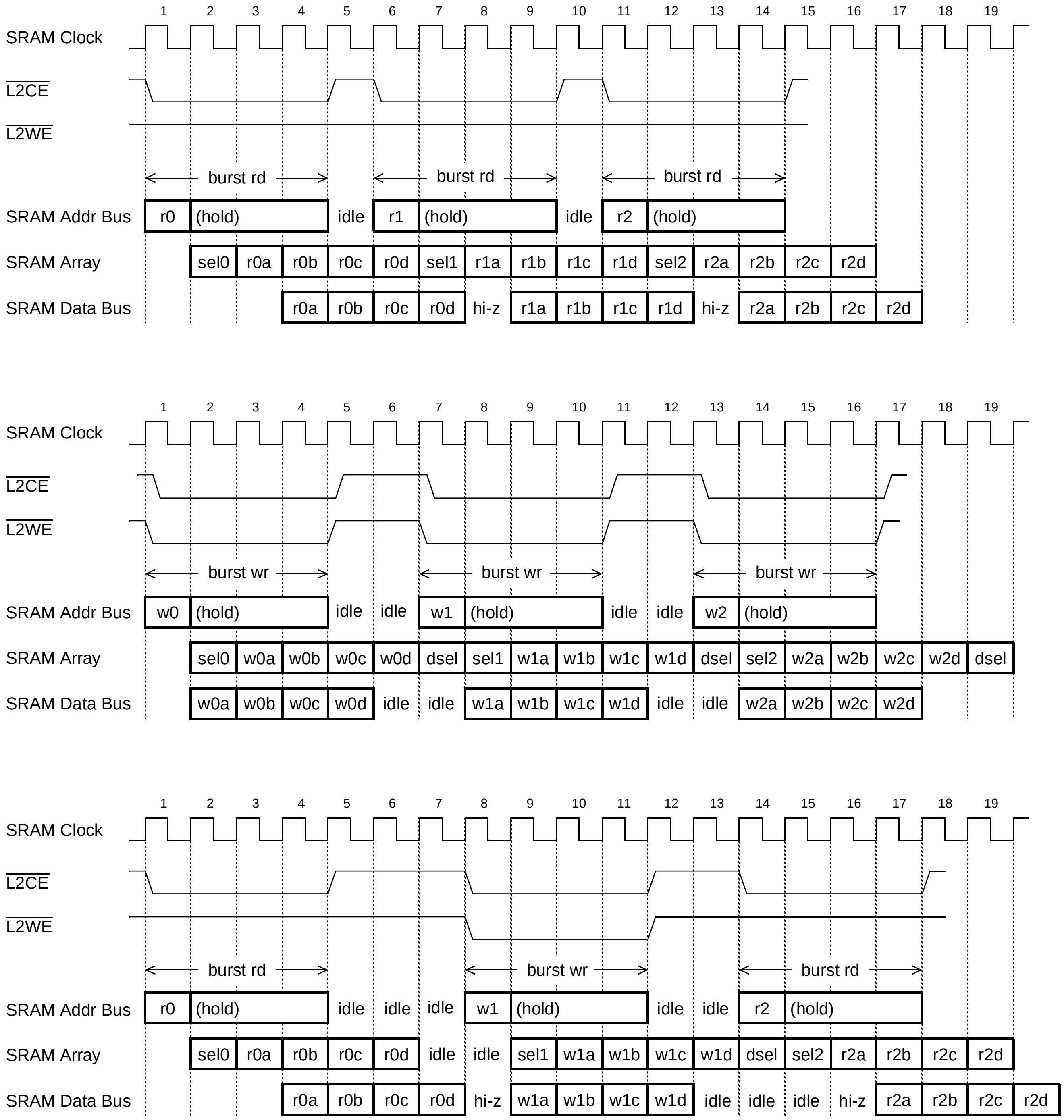
Figure 3-37. Late-Write SRAM Timing

3.7.10.3 PB3 SRAM

PB3 (pipelined burst, third generation) SRAMs are a later generation of SRAM than either pipelined burst SRAM (PB2) or late-write SRAM. PB3 SRAMs mimic the efficiencies of the late-write SRAMs, but operate more like traditional PB2 SRAMs (that is, they have no internal write queue). PB3 SRAMs stage the initial internal array access over two clock cycles, thereby requiring an additional wait state for the first read data beat.

Note that for PB3 SRAMs, the MPC7410 generates a single address for burst transfers of four data beats (32-bytes) into and out of the SRAM array. That is, the MPC7410 does use the burst feature (one address, many data beats) of the PB3 SRAMs. However, the MPC7410 does not support an eight-beat transfer to PB3 SRAMs, and therefore cannot support PB3 SRAMs in 32-bit L2 data bus mode.

Figure 3-38 shows memory access timings when the L2 cache interface is configured for PB3 SRAM.


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
SRAM Clock
$\overline{L2CE}$
$\overline{L2WE}$
burst rd
burst rd
burst rd
SRAM Addr Bus
r0 (hold) idle r1 (hold) idle r2 (hold)
SRAM Array
sel0 r0a r0b r0c r0d sel1 r1a r1b r1c r1d sel2 r2a r2b r2c r2d
SRAM Data Bus
r0a r0b r0c r0d hi-z r1a r1b r1c r1d hi-z r2a r2b r2c r2d
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
SRAM Clock
$\overline{L2CE}$
$\overline{L2WE}$
burst wr
burst wr
burst wr
SRAM Addr Bus
w0 (hold) idle idle w1 (hold) idle idle w2 (hold)
SRAM Array
sel0 w0a w0b w0c w0d dsel sel1 w1a w1b w1c w1d dsel sel2 w2a w2b w2c w2d dsel
SRAM Data Bus
w0a w0b w0c w0d idle idle w1a w1b w1c w1d idle idle w2a w2b w2c w2d
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
SRAM Clock
$\overline{L2CE}$
$\overline{L2WE}$
burst rd
burst wr
burst rd
SRAM Addr Bus
r0 (hold) idle idle idle w1 (hold) idle idle r2 (hold)
SRAM Array
sel0 r0a r0b r0c r0d idle idle sel1 w1a w1b w1c w1d dsel sel2 r2a r2b r2c r2d
SRAM Data Bus
r0a r0b r0c r0d hi-z w1a w1b w1c w1d idle idle idle hi-z r2a r2b r2c r2d

Note: For PB3, L2ZZ is reused as $\overline{L2ADS}$ and asserts during the first clock only of each $\overline{L2CE}$ assertion.
For PB3, internal array access requires 1 cycle to row select, 1 cycle for each column select of burst (a-d), 1 cycle deselect if write.

Figure 3-38. PB3 SRAM Timing

3.8 System Bus Interface Unit

The bus interface unit buffers bus requests from the L1 instruction cache, the L1 data cache, and the L2 cache, and executes the requests per the system bus protocol. It includes address register queues, prioritizing logic, and bus control logic. The bus interface unit includes a six-entry data transaction queue to support pipelining of multiple transactions. The bus interface also captures snoop addresses for snooping in the caches, the address register queues, and the reservation address. For additional information about the MPC7410 bus interface and the bus protocols, refer to Chapter 9, “System Interface Operation.”

3.9 Caches and System Bus Transactions

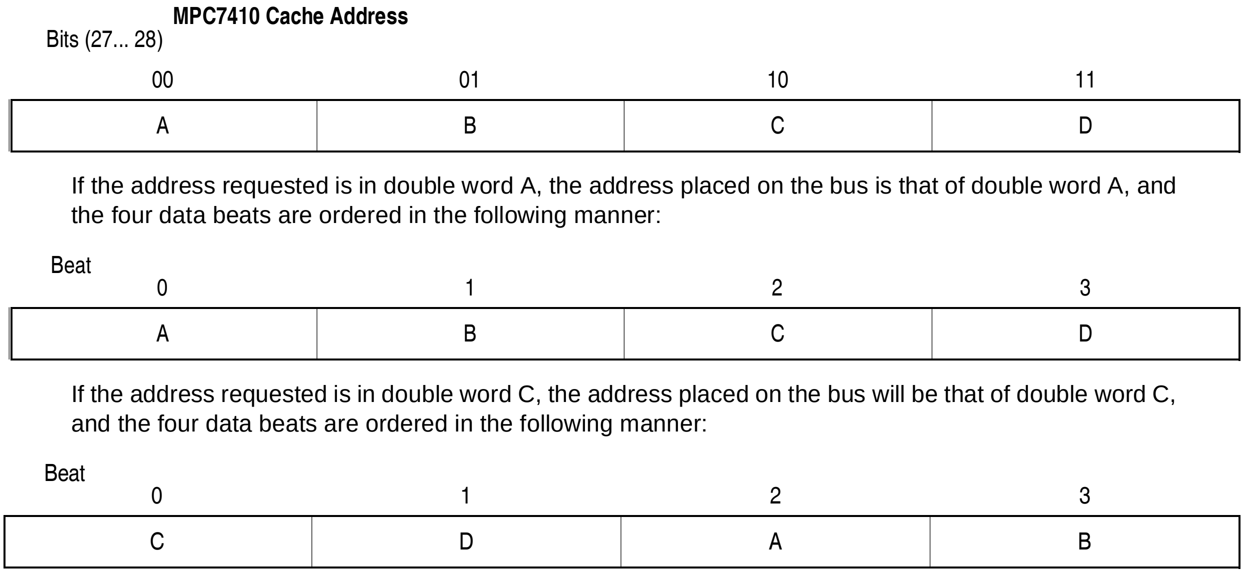
The MPC7410 transfers data to and from the cache in single-beat transactions of up to eight bytes, in two-beat burst transfers of 16 bytes for caching-inhibited (WIMG = x1xx) or caching-allowed, write-through (WIMG = 10xx) AltiVec loads and stores (in MPX bus mode), or in four-beat transactions of 32 bytes for cache block fills. The MPC7410 transfer burst ($\overline{\text{TBST}}$) output signal indicates to the system whether the current transaction is a single-beat transaction or burst (two- or four-beat) transfer.

Single-beat bus transactions can transfer from one to eight bytes to or from the MPC7410, and can be misaligned. Single-beat transactions can be caused by caching-allowed, write-through accesses (WIMG = 10xx), caching-inhibited accesses (WIMG = x1xx), accesses when the cache is disabled (HID0[DCE] is cleared), or accesses when the cache is locked (HID0[DLOCK] is set).

In MPX bus mode, two-beat burst transactions are caused by quad-word (128-bit) AltiVec loads and stores that are marked write-through or caching-inhibited. These two-beat burst transactions are always aligned to a quad-word boundary. In 60x bus mode, quad-word AltiVec loads and stores are split into two separate 8-byte, single-beat transactions on the system bus.

Cache block burst transactions on the MPC7410 always transfer 32-bytes of data in four beats of 8-bytes each, and are aligned to a double-word boundary. Burst transactions have an assumed address order. For caching-allowed read operations, instruction fetches, or caching-allowed, non-write-through write operations that miss in the cache, the MPC7410 presents the double-word-aligned address associated with the load/store instruction or instruction fetch that initiated the transaction.

As shown in Figure 3-39, the first double word contains the address of the load/store or instruction fetch that missed the cache. This minimizes latency by allowing the critical code or data to be forwarded to the processor before the rest of the block is filled. For all other burst operations, however, the entire block is transferred in order (cache-block aligned). Critical-double-word-first fetching on a cache miss applies to both the data and instruction cache.



MPC7410 Cache Address

Bits (27... 28)	00	01	10	11
	A	B	C	D

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

Beat	0	1	2	3
	A	B	C	D

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

Beat	0	1	2	3
	C	D	A	B

Figure 3-39. Double-Word Address Ordering—Critical Double Word First

3.9.1 Bus Operations Caused by Cache Control Instructions

The cache control, TLB management, and synchronization instructions supported by the MPC7410 may affect or be affected by the operation of the system bus. The operation of the instructions may also indirectly cause bus transactions to be performed, or their completion may be linked to the bus.

When memory coherency is required (WIMG = xx1x), the **dcbst**, **dcbf**, and **dcbi** instructions cause a broadcast on the system bus to maintain coherency. The **icbi** instruction is always broadcast, regardless of the state of the memory-coherency-required attribute. For detailed information on the cache control instructions, refer to Chapter 2, "Programming Model," in this book and Chapter 8, "Instruction Set," in *The Programming Environments Manual*.

Table 3-15 provides an overview of the bus operations initiated by cache control instructions. Note that Table 3-15 assumes that the WIM bits are set to 001; that is, the cache is operating in write-back mode, caching is allowed, and memory coherency is enforced.

Table 3-15. Bus Operations Caused by Cache Control Instructions (WIM = 001)

Instruction	Current Cache State	Next Cache State	Bus Operation	Comment
sync	Don't care	No change	**sync**	Waits for memory queues to complete bus activity
tlbie	Don't care	No change	**tlbie**	Address-only bus operation
tlbsync	Don't care	No change	**tlbsync**	Address-only bus operation
eieio	Don't care	No change	**eieio**	Address-only bus operation
dcbt	M, E, R, S	No change	None	—

Table 3-15. Bus Operations Caused by Cache Control Instructions (WIM = 001) (continued)

Instruction	Current Cache State	Next Cache State	Bus Operation	Comment
dcbt	I	E, R, or S	Read	Fetched cache block is stored in the cache
dcbtst	M, E, R, S	No change	None	—
dcbtst	I	E	RWITM (60x bus mode) RCLAIM (MPX bus mode)	Fetched cache block is stored in the cache
dcbz	M, E	M	None	Writes over modified data
dcbz	R, S, I	M	Kill	—
dcbst	M	E	Write with kill	Block is pushed
dcbst	E, R, S, I	No change	Clean	Address-only bus operation
dcbf	M	I	Write with kill	Block is pushed
dcbf	E, R, S, I	I	Flush	Address-only bus operation
dcba	M, E	M	None	Writes over modified data
dcba	R, S, I	M	Kill	—
dcbi	Don't care	I	Kill	Address-only bus operation
icbi	Don't care	I	**icbi**	—

For additional details about the specific bus operations performed by the MPC7410, see Chapter 9, "System Interface Operation."

3.9.2 Transfer Attributes

In addition to the address and transfer type signals, the MPC7410 supports the transfer attribute signals $\overline{\text{TBST}}$, TSIZ[0:2], $\overline{\text{WT}}$, $\overline{\text{CI}}$, and $\overline{\text{GBL}}$. The TBST and TSIZ[0:2] signals indicate the data transfer size for the bus transaction.

The $\overline{\text{WT}}$ signal reflects the write-through/write-back status (the complement of the W bit) for the transaction as determined by the MMU address translation during write operations. $\overline{\text{WT}}$ is also asserted for burst writes due to **dcbf** (flush) and **dcbst** (clean) instructions, snoop pushes, and **eciwx** transactions; $\overline{\text{WT}}$ is negated for **ecowx** transactions.

The $\overline{\text{CI}}$ signal reflects the caching-inhibited/caching-allowed status (the complement of the I bit) of the transaction as determined by the MMU address translation even if the L1 caches are locked. The $\overline{\text{CI}}$ signal is asserted for data loads or stores if the L1 data cache is disabled. The $\overline{\text{CI}}$ signal is also always asserted for **eciwx**/**ecowx** bus transactions independent of the address translation.

The $\overline{\text{GBL}}$ signal reflects the memory coherency requirements (the complement of the M bit) of the transaction as determined by the MMU address translation. Address bus masters assert $\overline{\text{GBL}}$ to indicate that the current transaction is a global access (that is, an access to memory shared by more than one device). Because cache block castouts and snoop pushes do not require snooping, the $\overline{\text{GBL}}$ signal is not asserted for these operations. Note that $\overline{\text{GBL}}$ is asserted for all data read or write operations when using real addressing mode (that is, address translation is disabled).

Table 3-16 summarizes the address and transfer attribute information presented on the bus by the MPC7410 for various master or snoop-related transactions.

Table 3-16. Address/Transfer Attributes Generated by the MPC7410

Bus Transaction	A[0:31]	TT[0–4]	$\overline{\text{TBST}}$	TSIZ[0:2]	$\overline{\text{WT}}$	$\overline{\text{CI}}$	$\overline{\text{GBL}}$
Instruction Fetch Operations							
Burst (caching-allowed)	PA[0:28] || 0b000	0 1 0 1 0	0	0 1 0	¬ W	1	¬ M
Single-beat read (caching-inhibited or cache disabled)	PA[0:28] || 0b000	0 1 0 1 0	1	0 0 0	¬ W	0	¬ M
Data Cache Operations							
Cache block fill (due to load miss)	PA[0:28] || 0b000	F 1 0 1 0	0	0 1 0	¬ W	1	¬ M
Cache block fill (due to store miss)	PA[0:28] || 0b000	A 1 1 1 0	0	0 1 0	1	1	¬ M
Store hit on shared/store miss merge	PA[0:26] || 0b00000	0 1 1 0 0	0	0 1 0	¬ W	1	¬ M
Castout (normal replacement)	CA[0:26] || 0b00000	0 0 1 1 0	0	0 1 0	1	1	1
Cache block clean due to **dcbst** hit to modified	PA[0:26] || 0b00000	0 0 1 1 0	0	0 1 0	0	1	1
Cache block flush due to **dcbf** hit to modified	PA[0:26] || 0b00000	0 0 1 1 0	0	0 1 0	0	1	1
Snoop copyback	CA[0:26] || 0b00000	0 0 1 1 0	0	0 1 0	0	1	1
dcbt, **dst**, **dstt**	PA[0:26] || 0b00000	0 1 0 1 0	0	0 1 0	¬ W	1	¬ M
dcbtst, **dstst**, **dststt** (60x bus mode)	PA[0:26] || 0b00000	0 1 1 1 0	0	0 1 0	¬ W	1	¬ M
dcbtst, **dstst**, **dststt** (MPX bus mode)	PA[0:26] || 0b00000	0 1 1 1 1	0	0 1 0	¬ W	1	¬ M
Data Cache Bypass Operations							
Single-beat read (caching-inhibited or cache disabled)	PA[0:31]	F 1 0 1 0	1	S S S	¬ W	¬ I	¬ M
AltiVec load (caching-inhibited, write-through, or cache disabled) in MPX bus mode	PA[0:28] || 0b000	F 1 0 1 0	0	0 0 1	¬ W	¬ I	¬ M
Single-beat write (caching-inhibited, write-through, or cache disabled)	PA[0:31]	0 0 0 1 0	1	S S S	¬ W	¬ I	¬ M

Table 3-16. Address/Transfer Attributes Generated by the MPC7410 (continued)

Bus Transaction	A[0:31]	TT[0–4]	$\overline{\text{TBST}}$	TSIZ[0:2]	$\overline{\text{WT}}$	$\overline{\text{CI}}$	$\overline{\text{GBL}}$
AltiVec store (caching-inhibited, write-through, or cache disabled) in MPX bus mode	PA[0–28] || 0b000	0 0 0 1 0	0	0 0 1	¬ W	¬ I	¬ M
stwcx. (caching-inhibited)	PA[0–29] || 0b00	1 0 0 1 0	1	1 0 0	¬ W	0	¬ M
Special Instructions							
icbi (addr-only)	PA[0–26] || 0b00000	0 1 1 0 1	0	0 1 0	¬ W	¬ I	¬ M
dcba (addr-only)	PA[0–26] || 0b00000	0 1 1 0 0	0	0 1 0	1	1	0
dcbz (addr-only)	PA[0–26] || 0b00000	0 1 1 0 0	0	0 1 0	1	1	0
dcbi (addr-only)	PA[0–26] || 0b00000	0 1 1 0 0	0	0 1 0	¬ W	¬ I	¬ M
dcbf (addr-only)	PA[0–26] || 0b00000	0 0 1 0 0	0	0 1 0	¬ W	¬ I	¬ M
dcbst (addr-only)	PA[0–26] || 0b00000	0 0 0 0 0	0	0 1 0	¬ W	¬ I	¬ M
sync (addr-only)	0x0000_0000	0 1 0 0 0	0	0 1 0	1	1	0
tlbsync (addr-only)	0x0000_0000	0 1 0 0 1	0	0 1 0	1	1	0
tlbie (addr-only)	EA[0–31]	1 1 0 0 0	0	0 1 0	1	1	0
eieio (addr-only)	0x0000_0000	1 0 0 0 0	0	0 1 0	1	1	0
eciwx	PA[0–29] || 0b00	1 1 1 0 0	EAR[28–31]		0	0	1
ecowx	PA[0–29] || 0b00	1 0 1 0 0	EAR[28–31]		1	0	1

Notes: PA = Physical address, CA = Cache address, EA = Effective address.
W,I,M = WIM state from address translation; ¬ = complement; 0 or 1 = WIM state implied by transaction type in table.
F = Instruction fetch transfer type mode; high if HID0[IFTT] = 0b1, high if **lwarx**, low otherwise.
A = Atomic; high if **stwcx.**, low otherwise
S = Transfer size
Special instructions listed may not generate bus transactions depending on cache state.
TT[0–4] = 0b01011 (RWNITC) is snooped by the MPC7410, but is not generated by the MPC7410.
TT[0–4] = 0b00001 (**lwarx** reservation set) is neither snooped nor generated by the MPC7410.

3.9.3 Snooping

The MPC7410 maintains data cache coherency in hardware by coordinating activity between the data cache, the memory subsystem, the L2 cache, and the bus interface unit. The MPC7410 has a copyback cache that relies on bus snooping to maintain cache coherency with other caches in the system. For the MPC7410, the coherency size of the bus is 32 bytes, the size of a cache block. This means that any bus transactions that cross an aligned 32-byte boundary must present a new address onto the bus at that boundary for proper snoop operation by the MPC7410, or they must operate noncoherently with respect to the MPC7410.

As bus operations are performed on the bus by other bus masters and itself, the MPC7410 bus snooping logic monitors the addresses and transfer attributes that are referenced. The MPC7410 must see all system coherency snoops to function properly in a symmetric multiprocessing (SMP) environment. The MPC7410 cannot support external devices that filter out snoop traffic on the bus (for example, an external, in-line cache).

The MPC7410 snoops bus transactions during the cycle that $\overline{\text{TS}}$ is asserted for all global transactions ($\overline{\text{GBL}}$ asserted).

The state of $\overline{\text{ABB}}$ is not sampled to determine a qualified snoop condition. Every assertion of $\overline{\text{TS}}$ detected by the MPC7410 (whether snooped or not) must be followed by an accompanying assertion of $\overline{\text{AACK}}$.

There are several bus transaction types defined for the system bus. As shown in Table 3-17, the MPC7410 snoops many, but not all, system transactions. The transactions in Table 3-17 correspond to the transfer type signals TT[0:4], which are described in Section 8.2.4.2, "Transfer Type (TT[0:4])."

Table 3-17. Snooped Bus Transaction Summary

Transaction	TT[0–4]	Snooped by MPC7410
Clean	00000	Yes
Flush	00100	Yes
sync	01000	Yes
Kill	01100	Yes
eieio	10000	No
External control word write	10100	No
TLB invalidate (**tlbie**)	11000	Yes
External control word read	11100	No
lwarx reservation set	00001	No
Reserved	00101	No
tlbsync	01001	Yes
icbi	01101	Yes
Reserved	1XX01	No
Write-with-flush	00010	Yes
Write-with-kill	00110	Yes
Read (or instruction fetch if HID0[IFTT] = 0b1)	01010	Yes
Read-with-intent-to-modify (RWITM)	01110	Yes
Write-with-flush-atomic	10010	Yes
Reserved	10110	No
Read-atomic (or data read if HID0[IFTT] = 0b1)	11010	Yes
Read-with-intent-to-modify-atomic	11110	Yes

Table 3-17. Snooped Bus Transaction Summary (continued)

Transaction	TT[0–4]	Snooped by MPC7410
Reserved	00011	No
Reserved	00111	No
Read-with-no-intent-to-cache (RWNITC)	01011	Yes
Read-claim (RCLAIM) (MPX bus mode only)	01111	Yes
Reserved	1XX11	No

Once a qualified snoop condition is detected on the bus, the snooped address associated with $\overline{\text{TS}}$ is compared against the data cache tags, reload buffer table entries, memory queues, reservation address, and/or other storage elements as appropriate. The L1 data cache tags and L2 cache tags are snooped for standard data cache coherency support. No snooping is done in the instruction cache for coherency.

The memory queues are snooped for pipeline collisions and memory coherency collisions. A pipeline collision is detected when another bus master addresses any portion of a line that this MPC7410's reload data buffer is currently in the process of loading (dRLDB loading from L2 cache, or dRLDB/L2 cache loading from memory). A memory coherency collision occurs when another bus master addresses any portion of a line that the MPC7410 has currently queued to write to memory from the data cache (castout or copyback), but has not yet been granted bus access to perform.

If the snooped address does not hit in the cache, snooping finishes with no action taken. If, however, the address hits in the cache, the MPC7410 reacts according to the coherency protocol diagrams shown in Section 3.4.3, "Coherency Protocols." Note that the MPC7410 snoops its own transactions and may assert $\overline{\text{ARTRY}}$ for **tlbie**, **tlbsync**, **icbi**, and **sync** broadcasts that result in pipeline collisions.

Chapter 4
Exceptions

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

The exception mechanism allows processors built on Power Architecture technology 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 architecture requires that exceptions be taken in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are handled strictly in order with respect to the instruction stream. When an instruction-caused exception is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute state, are required to complete before the exception is taken. In addition, if a single instruction encounters multiple exception conditions, those exceptions are taken and handled sequentially. Likewise, exceptions that are asynchronous and precise are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results.

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

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

In this book, the following terms are used to describe the stages of exception processing:

Recognition — Exception recognition occurs when the condition that can cause an exception is identified by the processor.

Taken	An exception is said to be taken when control of instruction execution is passed to the exception handler; that is, the context is saved and the instruction at the appropriate vector offset is fetched and the exception handler routine begins executing in supervisor mode.
Handling	Exception handling is performed by the software at the appropriate vector offset. Exception handling is begun in supervisor mode.

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

AltiVec Technology and the Exception Model

Only the three following exceptions may result from execution of an AltiVec instruction:

- An AltiVec unavailable exception occurs with an attempt to execute any non-stream AltiVec instruction with MSR[VEC] = 0. After this exception occurs, execution resumes at offset 0x00F20 from the physical base address indicated by MSR[IP]. This exception does not occur for data streaming instructions (**dst**[**t**], **dstst**[**t**] **dss**, and **dssall**). Also note that the VRSAVE register is not protected by this exception; this is consistent with the *AltiVec Programming Environments Manual*.
- A DSI exception occurs for an AltiVec load or store only if the load or store operation encounters a page fault (does not find a valid PTE during a table search operation) or a protection violation. Also a DSI exception occurs if an AltiVec load or store attempts to access a SR[T] = 1 (direct-store) memory location.
- An AltiVec assist exception may occur if an AltiVec floating-point instruction detects denormalized data as an input or output in Java mode. After this exception occurs, execution resumes at offset 0x01600 from the physical base address indicated by MSR[IP].

4.1 Exceptions

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

The types of exceptions are shown in Table 4-1. Note that all exceptions except for the performance monitor, AltiVec unavailable, instruction address breakpoint, system management, AltiVec assist, and thermal management (MPC7400 only) exceptions are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

Table 4-1. 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, thermal management exception
Synchronous	Precise	Instruction-caused exceptions

The exception classifications are discussed in greater detail in Section 4.2, “Exception Recognition and Priorities.” For a better understanding of how the MPC7410 implements precise exceptions, see Chapter 6, “Instruction Timing.” Exceptions implemented in the MPC7410, and conditions that cause them, are listed in Table 4-2, which also notes when an exception is implementation-specific to the MPC7410.

Table 4-2. Exceptions and Conditions

Exception Type	Vector Offset	Causing Conditions
Reserved	0x00000	—
System Reset	0x00100	Assertion of either $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$ or at power-on reset
Machine Check	0x00200	Assertion of $\overline{\text{TEA}}$ during a data bus transaction, assertion of $\overline{\text{MCP}}$, an address bus parity error on MPX bus, a data bus parity error on MPX bus, an L1 instruction cache error, and L1 data cache error, an L2 data parity error, or an L2 cache tag parity error. MSR[ME] must be set.
DSI	0x00300	As specified in the 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 write-through memory/cache access attributes.
ISI	0x00400	As specified in the 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 architecture
Floating-point Unavailable	0x00800	As specified in the architecture
Decrementer	0x00900	As defined by the architecture, when the msb of the DEC register changes from 0 to 1 and MSR[EE] = 1.
Reserved	0x00A00–00BFF	—

Table 4-2. Exceptions and Conditions (continued)

Exception Type	Vector Offset	Causing Conditions
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 MPC7410differs from the OEA by not taking this exception on an **isync**.
Reserved	0x00E00	The MPC7410 does not generate an exception to this vector. Other processors built on Power Architecture technology 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 (MPC7410-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 (**dstx**, **dss,** or **dssall**) (MPC7410-specific).
Instruction Address Breakpoint	0x01300	IABR[0–29] matches EA[0–29] of the next instruction to complete and IABR[BE] = 1 (MPC7410-specific).
System Management Interrupt	0x01400	MSR[EE] = 1 and $\overline{\text{SMI}}$ is asserted (MPC7410-specific).
Reserved	0x01500–015FF	—
Altivec Assist	0x01600	This MPC7410-specific exception supports denormalization detection in Java mode as specified in the *AltiVec Technology Programming Environments Manual*.
Thermal Management	0x01700	MPC7400 only. Generated when the thermal management assist unit detects that the temperature has exceeded the programmed threshold.
Reserved	0x01800–02FFF	—

4.2 Exception Recognition and Priorities

Exceptions are roughly prioritized by exception class, as follows:

1. Nonmaskable, asynchronous exceptions such as system reset and machine check exceptions, have priority over all other exceptions although the machine check exception condition can be disabled so the condition causes the processor to go directly into the checkstop state. These exceptions cannot be delayed and do not wait for completion of any precise exception handling.
2. Synchronous, precise exceptions are caused by instructions and are taken in strict program order.
3. Imprecise exceptions (imprecise mode floating-point enabled exceptions) are caused by instructions and they are delayed until higher priority exceptions are taken. Note that the MPC7410 does not implement an exception of this type.
4. Maskable asynchronous exceptions (external interrupt, decrementer, system management interrupt, thermal management, and performance monitor exceptions) are delayed until higher priority exceptions are taken.

The following list of categories describes how the MPC7410 handles exception conditions up to the point that the exception is taken. Note that a recoverable state is reached if the completed store queue is empty and any instruction that is next in program order, and has been signaled to complete, has completed. If MSR[RI] = 0, the MPC7410 is in a nonrecoverable state. Also, instruction completion is defined as updating all architectural registers associated with that instruction, and then removing that instruction from the completion buffer. When all the pending store instructions are completed, the completed store queue is empty.

- Exceptions caused by asynchronous events (interrupts). These exceptions are further distinguished by whether they are maskable and recoverable.
  - Asynchronous, nonmaskable, nonrecoverable

    System reset for assertion of $\overline{\text{HRESET}}$—Has highest priority and is taken immediately regardless of other pending exceptions or recoverability (includes power-on reset).
  - Asynchronous, maskable, nonrecoverable

    Machine check exception—Has priority over any other pending exception except system reset for assertion of $\overline{\text{HRESET}}$ (or power-on reset). Taken immediately regardless of recoverability.
  - Asynchronous, nonmaskable, recoverable

    System reset for $\overline{\text{SRESET}}$—Has priority over any other pending exception except system reset for $\overline{\text{HRESET}}$ (or power-on reset), or machine check. Taken immediately when a recoverable state is reached.

  - Asynchronous, maskable, recoverable

    System management interrupt, performance monitor, thermal management, external interrupt, and decrementer exceptions—Before handling this type of exception, the next instruction in program order must complete. If that instruction causes another type of exception, that exception is taken and the asynchronous, maskable recoverable exception remains pending until the instruction completes. Further instruction completion is halted. The asynchronous, maskable recoverable exception is taken when a recoverable state is reached.

- Instruction-related exceptions. These exceptions are further organized into the point in instruction processing at which they generate an exception.
  - Instruction fetch
    - ISI exceptions—Once this type of exception is detected, dispatching stops and the current instruction stream is allowed to drain out of the machine. If completing any of the instructions in this stream causes an exception, that exception is taken and the instruction fetch exception is discarded, but may be encountered again when instruction processing resumes. Otherwise, once all pending instructions have executed and a recoverable state is reached, the ISI exception is taken.
  - Instruction dispatch/execution
    - Program, DSI, alignment, floating-point unavailable, system call, instruction address breakpoint, and data address breakpoint. This type of exception is determined during dispatch or execution of an instruction. The exception remains pending until all instructions before the exception-causing instruction in program order complete. The exception is then taken without completing the exception-causing instruction. If completing these previous instructions causes an exception, that exception takes priority over the pending instruction dispatch/execution exception, which is discarded, but may be encountered again when instruction processing resumes.
  - Post-instruction execution
    - Trace—Trace exceptions are generated following execution and completion of an instruction while trace mode is enabled. If executing the instruction produces conditions for another type of exception, that exception is taken and the post-instruction exception is ignored for that instruction.

Note that these exception classifications correspond to how exceptions are prioritized, as described in Table 4-3.

Table 4-3. MPC7410 Exception Priorities

Priority	Exception	Cause
Asynchronous Exceptions (Interrupts)		
0	System Reset	Power-on reset, assertion of $\overline{\text{HRESET}}$ and $\overline{\text{TRST}}$ (hard reset)
1	Machine Check	Any enabled machine check condition (assertion of $\overline{\text{TEA}}$ or $\overline{\text{MCP}}$, address or data parity error, data cache error, instruction cache error, L2 data parity error, L2 tag error)
2	System Reset	Assertion of $\overline{\text{SRESET}}$ (soft reset)

Table 4-3. MPC7410 Exception Priorities (continued)

Priority	Exception	Cause
3	System Management Interrupt	Assertion of $\overline{\text{SMI}}$
4	External Interrupt	Assertion of $\overline{\text{INT}}$
5	Performance Monitor	Any programmer-specified performance monitor condition
6	Decrementer	Decrementer passes through zero.
7	Thermal Management	Any programmer-specified thermal management condition
Instruction Fetch Exceptions		
0	ISI	ISI exception conditions due to: 1. No-execute segment 2. Direct-store (T=1) segment
Instruction Dispatch/Execution Exceptions		
0	Instruction Address Breakpoint	Any instruction address breakpoint exception condition
1	Program	Illegal instruction, privileged instruction, or trap exception condition. Note that floating-point enabled program exceptions have lower priority.
2	System Call	System call (**sc**) instruction
3	Floating-point Unavailable	Any floating-point unavailable exception condition
4	Altivec Unavailable	Any unavailable AltiVec exception condition
5	Program	A floating-point enabled exception condition (lowest-priority program exception)
6	DSI	DSI exception due to **eciwx** or **ecowx** with EAR[E] = 0 (DSISR[11]). Lower priority DSI exception conditions are shown below.
7	Alignment	Any alignment exception condition, prioritized as follows: 1. Floating-point access not word-aligned 2. **lmw**, **stmw**, **lwarx**, or **stwcx.** not word-aligned 3. **eciwx** or **ecowx** not word-aligned 4. Multiple or string access with MSR[LE] set 5. **dcbz** to a locked L1 data cache
8	DSI	Page fault with SR[T] = 0
9	Alignment	**dcbz** to memory with write-through memory/cache access attributes or a disabled L1 data cache
10	DSI	DSI exception due to: • BAT/page protection violation (DSISR[4]) or • **lwarx**/**stwcx.** to BAT entry with write-through attributes (W = 1) or to BAT entry with caching-allowed attributes (I = 0) but with a locked L1 data cache (DSISR[5]) Note that if both occur simultaneously, both bits 4 and 5 of the DSISR are set.

Table 4-3. MPC7410 Exception Priorities (continued)

Priority	Exception	Cause
11	DSI	DSI exception due to any access except cache operations to a segment where SR[T] = 1 (DSISR[5]) or an access crosses from a T = 0 segment to one where T = 1 (DSISR[5])
12	DSI	DSI exception due to: • TLB page protection violation or • **lwarx**/**stwcx.** to page table entry with write-through attributes (W = 1) or to a page table entry with caching-allowed attributes (I = 0) but with a locked L1 data cache (DSISR[5]). Note that if both occur simultaneously, both bits 4 and 5 of the DSISR are set.
13	DSI	DSI exception due to DABR address match (DSISR[11]). Note that even though DSISR[5] and DSISR[11] are set by exceptions with different priorities, they can be set simultaneously.
14	AltiVec Assist	Denormalized data detected as input or output in the AltiVec vector floating-point unit (VFPU) while in Java mode
15	Trace	MSR[SE] = 1 (or MSR[BE] = 1 for branches)

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

4.3 Exception Processing

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

When an exception occurs, the address saved in SRR0 helps determine where instruction processing should resume when the exception handler returns control to the interrupted process. Depending on the exception, this may be the address in SRR0 or at the next address in the program flow. All instructions in the program flow preceding this one will have completed execution and no subsequent instruction will have begun execution. This may be the address of the instruction that caused the exception or the next one (as in the case of a system call or trace exception). The SRR0 register is shown in Figure 4-1.

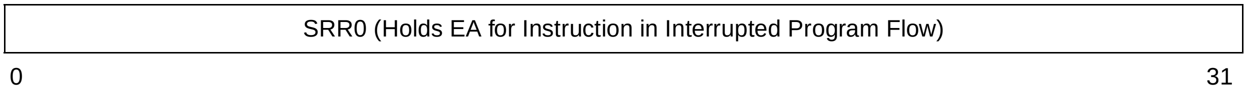


Figure 4-1. Machine Status Save/Restore Register 0 (SRR0)

SRR1 is used to save machine status (selected MSR bits and possibly other status bits) on exceptions and to restore those values when an **rfi** instruction is executed. SRR1 is shown in Figure 4-2.

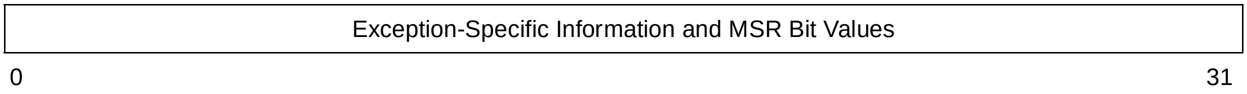


Figure 4-2. Machine Status Save/Restore Register 1 (SRR1)

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

The MPC7410's MSR is shown in Figure 4-3.

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

Figure 4-3. Machine State Register (MSR)

The MSR bits are defined in Table 4-4.

Table 4-4. MSR Bit Settings

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

Table 4-4. MSR Bit Settings (continued)

Bit(s)	Name	Description
19	ME	Machine check enable 0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.
20	FE0[2]	IEEE floating-point exception mode 0 (see Table 4-5)
21	SE	Single-step trace enable 0 The processor executes instructions normally. 1 The processor generates a single-step trace exception upon the successful execution of every instruction except **rfi**, **isync**, and **sc**. Successful execution means that the instruction caused no other exception.
22	BE	Branch trace enable 0 The processor executes branch instructions normally. 1 The processor generates a branch type trace exception when a branch instruction executes successfully.
23	FE1[2]	IEEE floating-point exception mode 1 (see Table 4-5)
24	—	Reserved. This bit corresponds to the AL bit of the POWER architecture.
25	IP	Exception prefix. The setting of this bit specifies whether an exception vector offset is prepended with Fs or 0s. In the following description, *nnnnn* is the offset of the exception. 0 Exceptions are vectored to the physical address 0x000*n_nnnn*. 1 Exceptions are vectored to the physical address 0xFFF*n_nnnn*.
26	IR[5]	Instruction address translation 0 Instruction address translation is disabled. 1 Instruction address translation is enabled. For more information see Chapter 5, “Memory Management.”
27	DR[4]	Data address translation 0 Data address translation is disabled. 1 Data address translation is enabled. For more information see Chapter 5, “Memory Management.”
28	—	Reserved
29	PMM[1]	Performance monitor marked mode 0 Process is not a marked process. 1 Process is a marked process. This bit can be set when statistics need to be gathered on a specific (marked) process. The statistics will only be gathered when the marked process is executing. MPC7410–specific; defined as optional by the architecture. For more information about the performance monitor marked mode bit, see Section 11.4, “Event Counting.”
30	RI	Indicates whether system reset or machine check exception is recoverable. 0 Exception is not recoverable. 1 Exception is recoverable. The RI bit indicates whether from the perspective of the processor, it is safe to continue (that is, processor state data such as that saved to SRR0 is valid), but it does not guarantee that the interrupted process is recoverable.
31	LE[6]	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

[1] Optional to the architecture

[2] A context synchronizing instruction must follow an **mtmsr** instruction.

[3] A **dssal**l and **sync** must precede an **mtmsr** instruction and then a context synchronizing instruction must follow.

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

[5] A context synchronizing instruction must follow an **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, 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 MPC7410-specific system management, performance monitor, and thermal management exceptions.

The IEEE floating-point exception mode bits (FE0 and FE1) together define whether floating-point exceptions are handled precisely, imprecisely, or whether they are taken at all. As shown in Table 4-5, if either FE0 or FE1 are set, the MPC7410 treats exceptions as precise. MSR bits are guaranteed to be written to SRR1 when the first instruction of the exception handler is encountered. For further details, see Chapter 2, “PowerPC Register Set” and Chapter 6, “Exceptions,” of *The Programming Environments Manual*.

Table 4-5. IEEE Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled
0	1	Imprecise nonrecoverable. For this setting, the MPC7410 operates in floating-point precise mode.
1	0	Imprecise recoverable. For this setting, the MPC7410 operates in floating-point precise mode.
1	1	Floating-point precise mode

4.3.1 Enabling and Disabling Exceptions

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

- System reset exceptions cannot be masked.
- A machine check exception can occur only if the machine check enable bit, MSR[ME], is set. If MSR[ME] is cleared, the processor goes directly into checkstop state when a machine check exception condition occurs. Individual machine check exceptions can be enabled and disabled through the following bits in the HID0 register, which is described in Table 4-8.
- Asynchronous, maskable exceptions (such as the external interrupt and decrementer) are enabled by setting MSR[EE]. When MSR[EE] = 0, recognition of these exception conditions is delayed. MSR[EE] is cleared automatically when an exception is taken to delay recognition of conditions causing those exceptions.
- The performance monitor exception is enabled for a specific process by setting MSR[PMM].
- The floating-point unavailable exception can be masked by setting MSR[FP].
- The AltiVec unavailable exception can be masked by setting MSR[VEC].

- IEEE floating-point enabled exceptions (a type of program exception) are ignored when both MSR[FE0] and MSR[FE1] are cleared. If either bit is set, all IEEE enabled floating-point exceptions are taken and cause a program exception.
- The trace exception is enabled by setting either MSR[SE] or MSR[BE].

4.3.2 Steps for Exception Processing

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

1. SRR0 is loaded with an instruction address that depends on the type of exception. See the individual exception description for details about how this register is used for specific exceptions.
2. SRR1[0, 7–9] are cleared; SRR1[1–5, 10–15] are loaded with information specific to the exception type; and SRR1[6, 16–31] are loaded with a copy of the corresponding MSR bits.
3. The MSR is set as described in Table 4-6. The new values take effect as the first instruction of the exception-handler routine is fetched.

   Note that MSR[IR] and MSR[DR] are cleared for all exception types; therefore, address translation is disabled for both instruction fetches and data accesses beginning with the first instruction of the exception-handler routine.
4. Instruction fetch and execution resumes, using the new MSR value, at a location specific to the exception type. The location is determined by adding the exception’s vector (see Table 4-2) to the base address determined by MSR[IP]. If IP is cleared, exceptions are vectored to the physical address 0x000*n_nnnn*. If IP is set, exceptions are vectored to the physical address 0xFFF*n_nnnn*. For a machine check exception that occurs when MSR[ME] = 0 (machine check exceptions are disabled), the checkstop state is entered (the machine stops executing instructions). See Section 4.6.2, “Machine Check Exception (0x00200).”

4.3.3 Setting MSR[RI]

An operating system may handle MSR[RI] as follows:

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

4.3.4 Returning from an Exception Handler

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

- All previous instructions have completed to a point where they can no longer cause an exception.
- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The **rfi** instruction copies SRR1 bits back into the MSR.
- Instructions fetched after this instruction execute in the context established by this instruction.
- Program execution resumes at the instruction indicated by SRR0.

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

4.4 Process Switching

The following instructions are useful for restoring proper context during process switching:

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

The operating system should set MSR[RI] as described in Section 4.3.3, “Setting MSR[RI].”

4.5 Data Stream Prefetching and Exceptions

As described in Chapter 5, “Cache, Exceptions, and Memory Management,” of the *AltiVec Technology Programming Environments Manual*, exceptions do not automatically cancel data stream prefetching. The operating system must stop streams explicitly when warranted—for example, when switching processes or changing virtual memory context. Care must be taken if data stream prefetching is used while in supervisor mode (MSR[PR] = 0).

4.6 Exception Definitions

Table 4-6 shows all the types of exceptions that can occur with the MPC7410 and the MSR settings when the processor goes into supervisor mode due to an exception. Depending on the exception, certain of these bits are stored in SRR1 when an exception is taken.

Table 4-6. MSR Setting Due to Exception

Exception Type	MSR Bit Name MSR Bit Number																
	VEC **6**	**POW** **13**	**ILE** **15**	**EE** **16**	**PR** **17**	**FP** **18**	**ME** **19**	**FE0** **20**	**SE** **21**	**BE** **22**	**FE1** **23**	**IP** **25**	**IR** **26**	**DR** **27**	**PM** **29**	**RI** **30**	**LE** **31**
System Reset	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Machine Check	0	0	—	0	0	0	0	0	0	0	0	—	0	0	0	0	ILE
DSI	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
ISI	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
External Interrupt	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Alignment	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Program	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Floating-point Unavailable	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Decrementer	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
System Call	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Trace Exception	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Performance Monitor	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Altivec Unavailable	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Instruction Address Breakpoint	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
System Management	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Altivec Assist	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE
Thermal Management	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE

Key:		
	0	Bit is cleared
	ILE	Bit is copied from the MSR[ILE]
	—	Bit is not altered
	Reserved bits are read as if written as 0	

The setting of the exception prefix bit (IP) determines how exceptions are vectored. If the bit is cleared, exceptions are vectored to the physical address 0x000*n_nnnn* (where *n_nnnn* is the vector offset); if IP is set, exceptions are vectored to physical address 0xFFF*n_nnnn*. Table 4-2 shows the exception vector offset of the first instruction of the exception handler routine for each exception type.

4.6.1 System Reset Exception (0x00100)

The MPC7410 implements the system reset exception as defined in the architecture (OEA). The system reset exception is a nonmaskable, asynchronous exception signaled to the processor through the assertion of system-defined signals. In the MPC7410, the exception is signaled by the assertion of either the HRESET or SRESET input signals, described more fully in Chapter 8, “Signal Descriptions.”

A hard reset is initiated by asserting HRESET. A hard reset is used primarily for power-on reset (POR) (in which case TRST must also be asserted), but can also be used to restart a running processor. The HRESET signal must be asserted during power up and must remain asserted for a period that allows the PLL to achieve lock and the internal logic to be reset. This period is specified in the *MPC7410 Hardware Specifications*. If HRESET is asserted for less than the required interval, the results are not predictable.

If a hard reset request occurs (HRESET asserted), the processor immediately branches to the system reset exception vector (0xFFF0_0100) without attempting to reach a recoverable state. If HRESET is asserted during normal operation, all operations cease and the machine state is lost. The MPC7410 internal state after a hard reset is defined in Table 2-17.

A soft reset is initiated by asserting SRESET. If SRESET is asserted, the processor is first put in a recoverable state. To do this, the MPC7410 allows any instruction at the point of completion to either complete or take an exception (note that load/store string or multiple accesses are not split), blocks completion of any following instructions and allows the completion queue to empty. If the soft reset request is made while the MPC7410 is in trace mode (MSR[SE] = 1 or MSR[BE] = 1), the exception is set as nonrecoverable and SRR1[30] is cleared (SRR1[30] = 0). The state before the exception occurred is then saved as specified in the architecture and instruction fetching begins at the system reset exception vector offset, 0x00100. The vector base address for a soft reset depends on the setting of MSR[IP] (either 0x0000_0100 or 0xFFF0_0100). Soft resets are third in priority, after hard reset and machine check. Except for the trace mode condition, this exception is recoverable provided attaining a recoverable state does not generate a machine check.

SRESET is an edge-sensitive signal that can be asserted and negated asynchronously, provided the minimum pulse width specified in the *MPC7410 Hardware Specifications* is met. The system reset exception modifies the MSR, SRR0, and SRR1, as described in *The Programming Environments Manual*. Unlike hard reset, soft reset does not directly affect the states of output signals. Attempts to use SRESET during a hard reset sequence or while the JTAG logic is non-idle can cause unpredictable results.

The MPC7410 implements HID0[NHR], which helps software distinguish a hard reset from a soft reset. Because this bit is cleared by a hard reset, but not by a soft reset, software can set this bit after a hard reset and determine whether a subsequent reset is a hard or soft reset (by examining whether this bit is still set). See Section 2.1.5.1, “Hardware Implementation-Dependent Register 0 (HID0).”

Table 4-7 lists register settings when a system reset exception is taken.

Table 4-7. System Reset Exception—Register Settings

Register	Setting Description			
SRR0	Cleared to zero by a hard reset On a soft reset, set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31 Loaded with equivalent MSR bits Note that if the processor state is corrupted to the extent that execution cannot resume reliably, MSR[RI] (SRR1[30]) is cleared.			
MSR	VEC 0 POW 0 ILE — EE 0 LE ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

4.6.2 Machine Check Exception (0x00200)

The MPC7410 implements the machine check exception as defined in the architecture (OEA). The MPC7410 conditionally initiates a machine check exception if MSR[ME] = 1 and any of the following occur:

- A system bus error ($\overline{\text{TEA}}$ assertion on data bus)
- Assertion of the machine check ($\overline{\text{MCP}}$) signal
- Address bus parity error on system bus
- Data bus parity error on system bus
- L2 data bus parity error

As defined in the architecture, the exception is not taken if MSR[ME] is cleared, in which case the processor enters a checkstop state.

Certain machine check conditions can be enabled and disabled using HID0 bits, as described in Table 4-8.

Table 4-8. HID0 Machine Check Enable Bits

Bit	Name	Function
0	EMCP	Enable $\overline{\text{MCP}}$. The primary purpose of this bit is to mask further machine check exceptions caused by assertion of $\overline{\text{MCP}}$, similar to how MSR[EE] can mask external interrupts. 0 Masks $\overline{\text{MCP}}$. Assertion of $\overline{\text{MCP}}$ does not generate a machine check exception or a checkstop. 1 Assertion of $\overline{\text{MCP}}$ causes a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1.
2	EBA	Enable/disable 60x bus address parity checking 0 Prevents address parity checking. 1 Allows an address parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. EBA and EBD allow the processor to operate with memory subsystems that do not generate parity.
3	EBD	Enable 60x bus data parity checking 0 Parity checking is disabled. 1 Allows a data parity error to cause a checkstop if MSR[ME] = 0 or a machine check exception if MSR[ME] = 1. EBA and EBD allow the processor to operate with memory subsystems that do not generate parity.
15	NHR	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 MPC7410 never writes this bit unless executing an **mtspr**(HID0).

A $\overline{\text{TEA}}$ indication on the bus can result from any load or store operation initiated by the processor. In general, $\overline{\text{TEA}}$ is expected to be used by a memory controller to indicate that a memory parity error or an uncorrectable memory ECC error has occurred. Note that the resulting machine check exception is imprecise and unordered with respect to the instruction that originated the bus operation.

If MSR[ME] and the appropriateHID0 and bits are set, the exception is recognized and handled; otherwise, in most cases, the processor generates an internal checkstop condition. When a processor is in checkstop state, instruction processing is suspended and generally cannot continue without restarting the processor. Note that many conditions may lead to the checkstop condition; the disabled machine check exception is only one of these.

A machine check exception may result from referencing a nonexistent physical address, either directly (with MSR[DR] = 0) or through an invalid translation. If a **dcbz** instruction introduces a block into the cache associated with a nonexistent physical address, a machine check exception can be delayed until an attempt is made to store that block to main memory. Not all processors built on Power Architecture technology provide the same level of error checking. Checkstop sources are implementation-dependent.

Machine check exceptions are enabled when MSR[ME] = 1; this is described in Section 4.6.2.1, “Machine Check Exception Enabled (MSR[ME] = 1).” If MSR[ME] = 0 and a machine check occurs, the processor enters the checkstop state. The checkstop state is described in Section 4.6.2.2, “Checkstop State (MSR[ME] = 0).”

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

Machine check exceptions are enabled when MSR[ME] = 1. When a machine check exception is taken, registers are updated as shown in Table 4-9.

Table 4-9. Machine Check Exception—Register Settings

Register	Setting Description			
SRR0	On a best-effort basis the MPC7410 sets this to an EA of some instruction that was executing or about to be executing when the machine check condition occurred.			
SRR1	0 Cleared 1–5 Reserved 6 Loaded with equivalent MSR bit 7–9 Cleared 10 Reserved 11 L2DP 12 MCP. Set when $\overline{\text{MCP}}$ signal is asserted; otherwise zero 13 TEA. Set when $\overline{\text{TEA}}$ signal is asserted; otherwise zero 14 DP. Set when a data bus parity error is detected on MPXbus; otherwise zero 15 AP. Set when a address bus parity error is detected on MPXbus; otherwise zero 16–29 Loaded with equivalent MSR bits 30 Set in case of a recoverable exception 31 Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 PR 0	FP 0 ME 0 FE0 0 SE 0	BE 0 FE1 0 IP — IR 0	DR 0 PM 0 RI 0 LE ILE

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

Note that to handle another machine check exception, the exception handler should set MSR[ME] as soon as it is practical after a machine check exception is taken. Otherwise, subsequent machine check exceptions cause the processor to enter the checkstop state.

When the MPC7410 takes the machine check exception, it sets one or more error bits in SRR1. The MPC7410 has two data parity error sources that can cause a machine check exception. TheL2DP bit indicates a data parity error on the L2 bus, and DP indicates a data parity error on the system bus. The MCP bit (SRR1[12]) indicates that the machine check signal was asserted. The TEA bit (SRR1[13]) indicates that the machine check was caused by a $\overline{\text{TEA}}$ assertion on the system bus.

The machine check exception is usually unrecoverable in the sense that execution cannot resume in the context that existed before the exception. If the condition that caused the machine check does not otherwise prevent continued execution, MSR[ME] is set by software to allow the processor to continue execution at the machine check exception vector address. Typically, earlier processes cannot resume; however, operating systems can use the machine check exception handler to try to identify and log the cause of the machine check condition.

When a machine check exception is taken, instruction fetching resumes at offset 0x00200 from the physical base address indicated by MSR[IP].

4.6.2.2 Checkstop State (MSR[ME] = 0)

If MSR[ME] = 0 and a machine check condition occurs, the processor enters the checkstop state.

When a processor is in checkstop state, instruction processing is suspended and generally cannot resume without the processor being reset. The contents of all latches are frozen within two cycles upon entering checkstop state.

Note that the MPC7410 has a $\overline{\text{CKSTP\_OUT}}$ signal (open-drain) that is asserted when the MPC7410 enters the checkstop state. Also, external logic can cause the MPC7410 to enter the checkstop state by asserting CKSTP_IN. See Section 8.5.3.5, "Checkstop Input (CKSTP_IN)—Input" and Section 8.5.3.6, "Checkstop Output (CKSTP_OUT)—Output" for more information on these checkstop signals.

4.6.3 DSI Exception (0x00300)

A DSI exception occurs when no higher priority exception exists and an error condition related to a data memory access occurs. The DSI exception is implemented as it is defined in the architecture (OEA). For details on the DSI exception, see "DSI Exception (0x00300)," in *The Programming Environments Manual*. For example, a **lwarx** or **stwcx.** instruction that addresses memory to be mapped with the write-through (W = 1) or caching-inhibited (I = 1) attribute causes a DSI exception.

4.6.3.1 DSI Exception—Page Fault

When there is a TLB miss for a load, store, or cache operation, a DSI exception is taken if the resulting hardware table search causes a page fault.

The condition that caused the exception is defined in the DSISR. These conditions also use the data address register (DAR) as shown in Table 4-10.

Table 4-10. DSI Exception—Register Settings

Register	Setting Description
DSISR	0 Cleared 1 Set by the hardware (if HID0[STEN]=0) or the DTLB miss exception handler if the translation of an attempted access is not found in the primary page table entry group (PTEG), or in the rehashed secondary PTEG, or in the range of a DBAT register; otherwise cleared. 2–3 Cleared 4 Set if a memory access is not permitted by the page or BAT protection mechanism; otherwise cleared. 5 Set if the **lwarx** or **stwcx.** instruction is attempted to write-through (W =1) or caching-inhibited (I = 1) memory. 6 Set for a store operation and cleared for a load operation. 7–8 Cleared 9 Set if DABR match occurs, otherwise cleared. 10 Cleared 11 Set if **eciwx** or **ecowx** instruction is executed when EAR[E] = 0; otherwise cleared. 12-31 Cleared
DAR	Set to the effective address of a memory element that caused the DSI, as described in *The Programming Environments Manual.*

4.6.3.2 DSI Exception—Data Address Breakpoint Facility

The MPC7410 also implements the data address breakpoint facility, which is defined as optional in the architecture and is supported by the optional data address breakpoint register (DABR) and the DSI exception. Although the architecture does not strictly prescribe how this facility must be implemented, the MPC7410 follows the recommendations provided by the architecture and described in Chapter 2, “Programming Model,” and Chapter 6 “Exceptions,” in *The Programming Environments Manual*. The granularity of the data address breakpoint compare is a double word for all accesses except AltiVec quad-word loads and stores. For AltiVec accesses, the least significant bit of the DAB field (DABR[28]) is ignored, thus providing quad-word granularity. For these quad-word DAB matches, the DAR register is loaded with a quad-word aligned address.

When a DSI exception is taken, instruction fetching resumes at offset 0x00300 from the physical base address indicated by MSR[IP].

4.6.4 ISI Exception (0x00400)

An ISI exception occurs when no higher priority exception exists and an attempt to fetch the next instruction fails. This exception is implemented as it is defined by the architecture (OEA), and is taken for the following conditions:

- The effective address cannot be translated.
- The fetch access is to a no-execute segment (SR[N] = 1).
- The fetch access is to guarded storage and MSR[IR] = 1.
- The fetch access violates memory protection.

When an ISI exception is taken, instruction fetching resumes at offset 0x00400 from the physical base address indicated by MSR[IP].

4.6.5 External Interrupt Exception (0x00500)

An external interrupt is signaled to the processor by the assertion of the external interrupt signal ($\overline{\text{INT}}$) when MSR[EE] = 1. The $\overline{\text{INT}}$ signal is expected to remain asserted until the MPC7410 takes the external interrupt exception. If $\overline{\text{INT}}$ is negated early, recognition of the interrupt request is not guaranteed. After the MPC7410 begins execution of the external interrupt handler, the system can safely negate $\overline{\text{INT}}$. When the MPC7410 detects assertion of $\overline{\text{INT}}$, it stops dispatching and waits for all pending instructions to complete, including string and multiple instructions. This allows any instructions in progress that need to take an exception to do so before the external interrupt is taken. After all instructions have vacated the completion buffer, the MPC7410 takes the external interrupt exception as defined in the architecture (OEA).

An external interrupt may be delayed by other higher priority exceptions or if MSR[EE] is cleared when the exception occurs.

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

Table 4-11 lists register settings when an external interrupt exception is taken.

Table 4-11. External Interrupt Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0 Cleared 1 Set when an external interrupt exception is caused by the ICTRL[CIRQ] bit 2–5 Cleared 6 Loaded with equivalent MSR bits 7–9 Cleared 10 Set when an external interrupt exception is caused by $\overline{\text{INT}}$ assertion 11–15Cleared 16–31Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

4.6.6 Alignment Exception (0x00600)

The MPC7410 implements the alignment exception as defined by the architecture (OEA). An alignment exception is initiated when any of the following occurs:

- The operand of a floating-point load or store is not word-aligned.
- The operand of **lmw**, **stmw**, **lwarx**, or **stwcx.** is not word-aligned.
- The operand of **dcbz** is in a page that is write-through or cache-inhibited.
- An attempt is made to execute **dcbz** when the data cache is disabled or locked.
- An **eciwx** or **ecowx** is not word-aligned.
- A multiple or string access is attempted with MSR[LE] set.

When an alignment exception is taken, instruction fetching resumes at offset 0x00600 from the physical base address indicated by MSR[IP].

The register settings for alignment exceptions are shown in Table 4-12.

Table 4-12. Alignment Interrupt—Register Settings

Register	Setting
DSISR	0—14 Cleared 15–16 For instructions that use register indirect with index addressing—set to bits 29–30 of the instruction. For instructions that use register indirect with immediate index addressing—cleared. 17 For instructions that use register indirect with index addressing—set to bit 25 of the instruction. For instructions that use register indirect with immediate index addressing— Set to bit 5 of the instruction 18–21 For instructions that use register indirect with index addressing—set to bits 21–24 of the instruction. For instructions that use register indirect with immediate index addressing—set to bits 1–4 of the instruction. 22–26 Set to bits 6–10 (identifying either the source or destination) of the instruction. Undefined for **dcbz**. 27–31 Set to bits 11–15 of the instruction (**r**A) for instructions that use the update form. For **lmw**, **lswi**, and **lswx** instructions, set to either bits 11–15 of the instruction or to any register number not in the range of registers loaded by a valid form instruction. Otherwise undefined.
DAR	Set to the EA of the data access as computed by the instruction causing the alignment exception.

4.6.7 Program Exception (0x00700)

The MPC7410 implements the program exception as it is defined by the architecture (OEA). A program exception occurs when no higher priority exception exists and one or more of the exception conditions defined in the OEA occur.

The MPC7410 invokes the system illegal instruction program exception when it detects any instruction from the illegal instruction class. The MPC7410 fully decodes the SPR field of the instruction. If an undefined SPR is specified, a program exception is taken.

The UISA defines **mtspr** and **mfspr** with the record bit (Rc) set as causing a program exception or giving a boundedly undefined result. In the MPC7410, the appropriate condition register (CR) should be treated as undefined. Likewise, the architecture states that the Floating Compared Unordered (**fcmpu**) or Floating Compared Ordered (**fcmpo**) instructions with the record bit set can either cause a program exception or provide a boundedly undefined result. In the MPC7410 the BF field in an instruction encoding for these cases is considered undefined.

The MPC7410 does not support either of the two floating-point imprecise modes supported by the architecture. Unless exceptions are disabled (MSR[FE0] = MSR[FE1] = 0), all floating-point exceptions are treated as precise.

When a program exception is taken, instruction fetching resumes at offset 0x00700 from the physical base address indicated by MSR[IP]. Chapter 6, “Exceptions,” in *The Programming Environments Manual* describes register settings for this exception.

4.6.8 Floating-Point Unavailable Exception (0x00800)

The floating-point unavailable exception is implemented as defined in the architecture. A floating-point unavailable exception occurs when no higher priority exception exists, an attempt is made to execute a floating-point instruction (including floating-point load, store, or move instructions), and the floating-point available bit in the MSR is disabled, (MSR[FP] = 0). Register settings for this exception are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

When a floating-point unavailable exception is taken, instruction fetching resumes at offset 0x00800 from the physical base address indicated by MSR[IP].

4.6.9 Decrementer Exception (0x00900)

The decrementer exception is implemented in the MPC7410 as it is defined by the architecture. The decrementer exception occurs when no higher priority exception exists, a decrementer exception condition occurs (for example, the decrementer register has completed decrementing), and MSR[EE] = 1. In the MPC7410, the decrementer register is decremented at one fourth the bus clock rate. Register settings for this exception are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

When a decrementer exception is taken, instruction fetching resumes at offset 0x00900 from the physical base address indicated by MSR[IP].

4.6.10 System Call Exception (0x00C00)

A system call exception occurs when a System Call (**sc**) instruction is executed. In the MPC7410, the system call exception is implemented as it is defined in the architecture. Register settings for this exception are described in Chapter 6, “Exceptions,” in *The Programming Environments Manual*.

When a system call exception is taken, instruction fetching resumes at offset 0x00C00 from the physical base address indicated by MSR[IP].

4.6.11 Trace Exception (0x00D00)

The trace exception is taken if MSR[SE] = 1 or if MSR[BE] = 1 and the currently completing instruction is a branch. Each instruction considered during trace mode completes before a trace exception is taken. When a **mtmsr** instruction is executed and the MSR[SE] transitions from 0 to 1, following the completion of that **mtmsr**, a trace exception is taken.

Implementation Note—The MPC7410 processor diverges from the architecture in that it does not take trace exceptions on the **isync** instruction.

When a trace exception is taken, instruction fetching resumes at offset 0x00D00 from the base address indicated by MSR[IP].

4.6.12 Floating-Point Assist Exception (0x00E00)

The optional floating-point assist exception defined by the architecture is not implemented in the MPC7410.

4.6.13 Performance Monitor Exception (0x00F00)

The MPC7410 microprocessor provides a performance monitor facility to monitor and count predefined events such as processor clocks, misses in either the instruction cache or the data cache, instructions dispatched to a particular execution unit, mispredicted branches, and other occurrences. An overflow of the counter in such events can be used to trigger the performance monitor exception. The performance monitor facility is not defined by the architecture.

The performance monitor provides the ability to generate a performance monitor exception triggered by an enabled condition or event. This exception is triggered by an enabled condition or event defined as follows:

- A PMC*x* register overflow condition occurs
  - — MMCR0[PMC1CE] and PMC1[OV] are both set
  - — MMCR0[PMCjCE] and PMCj[OV] are both set (j> 1)
- A time base event: MMCR0[TBEE] = 1 and the TBL bit specified in MMCR0[TBSEL] changes from 0 to 1
- An $\overline{\text{SMI}}$ event: MMCR2[SMINTENABLE] = 1 and $\overline{\text{SMI}}$ is asserted.

MMCR0[PMXE] must be set for any of these conditions to signal a performance monitor exception.

Although the performance monitor exception may occur with MSR[EE] = 0, the exception is not taken until MSR[EE] = 1.

As a result of a performance monitor exception being generated, the performance monitor saves in the SIAR the effective address of the last instruction completed before the exception is generated. Note that SIAR is not updated if performance monitor counting has been disabled by setting MMCR0[0].

The performance monitor can receive a performance monitor exception request from an off-chip performance monitor or device. This is accomplished by setting the mask bit in MMCR2[SMINTENABLE] and asserting $\overline{\text{SMI}}$. Under this condition, the MPC7410 takes a performance monitor exception rather than an SMI exception.

The performance monitor can be used for the following:

- To increase system performance with efficient software, especially in a multiprocessing system. Memory hierarchy behavior must be monitored and studied to develop algorithms that schedule tasks (and perhaps partition them) and that structure and distribute data optimally.
- To help system developers bring up and debug their systems.

The performance monitor uses the following SPRs:

- The performance monitor counter registers (PMC1–PMC4) are used to record the number of times a certain event has occurred. UPMC1–UPMC4 provide user-level read access to these registers.
- The monitor mode control registers (MMCR0–MMCR2) are used to enable various performance monitor exception functions. UMMCR0–UMMCR2 provide user-level read access to these registers.
- The sampled instruction address register (SIAR) contains the effective address of an instruction executing at or around the time that the processor signals the performance monitor exception condition. The USIAR register provides user-level read access to the SIAR.

Table 4-13 lists register settings when a performance monitor exception is taken.

Table 4-13. Performance Monitor Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key:	0	Bit is cleared
	ILE	Bit is copied from the MSR[ILE]
	—	Bit is not altered

As with other exceptions, the performance monitor exception follows the normal exception model with a defined exception vector offset (0x00F00). The priority of the performance monitor exception lies between the external exception and the decrementer exception (see Table 4-3). The contents of the SIAR are described in Section 2.1.5.7, “Performance Monitor Registers.” The performance monitor is described in Chapter 11, “Performance Monitor.”

4.6.14 AltiVec Unavailable Exception (0x00F20)

The AltiVec facility includes another instruction-caused, precise exception in addition to the exceptions defined by the architecture (OEA). An AltiVec unavailable exception occurs when no higher priority exception exists (see Table 4-3), and an attempt is made to execute an AltiVec instruction that accesses the vector register (VR) or the vector status and control register (VSCR) when MSR[VEC] = 0.

Note that the data streaming instructions, **dss**, **dst**, and **dstst** do not cause an AltiVec unavailable exception: the VR and VSCR registers are available to the data streaming instructions even when MSR[VEC] = 0.

4.6.15 Instruction Address Breakpoint Exception (0x01300)

An instruction address breakpoint exception occurs when all of the following conditions are met:

- The instruction breakpoint address IABR[0–29] matches EA[0–29] of the next instruction to complete in program order. The instruction that triggers the instruction address breakpoint exception is not executed before the exception handler is invoked.
- The breakpoint enable bit (IABR[BE]) is set.

The instruction tagged with the match does not complete before the breakpoint exception is taken.

Table 4-14 lists register settings when an instruction address breakpoint exception is taken.

Table 4-14. Instruction Address Breakpoint Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE Set to value of ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key:		
	0	Bit is cleared
	ILE	Bit is copied from the MSR[ILE]
	—	Bit is not altered

The MPC7410 requires that an **mtspr** to the IABR be followed by a context-synchronizing instruction. The MPC7410 cannot generate a breakpoint response for that context-synchronizing instruction if the breakpoint is enabled by the **mtspr**[IABR] immediately preceding it. The MPC7410 also cannot block a breakpoint response on the context-synchronizing instruction if the breakpoint was disabled by the **mtspr**[IABR] instruction immediately preceding it. The format of the IABR register is shown in Section 2.1.5.5, "Instruction Address Breakpoint Register (IABR)."

When an instruction address breakpoint exception is taken, instruction fetching resumes at offset 0x01300 from the base address indicated by MSR[IP].

4.6.16 System Management Interrupt Exception (0x01400)

The MPC7410 implements a system management interrupt, which is not defined by the architecture. The system management interrupt is very similar to the external interrupt and it must be enabled with MSR[EE] = 1. It is particularly useful in implementing the nap mode. It has priority over an external interrupt (see Table 4-3) and uses a different vector in the exception table (offset 0x01400).

Table 4-15 lists register settings when a system management interrupt is taken.

Table 4-15. System Management Interrupt Exception—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31Loaded with equivalent MSR bits

Table 4-15. System Management Interrupt Exception—Register Settings (continued)

MSR	VEC 0 POW 0 ILE — EE 0 LE Set to value of ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

Like the external interrupt, a system management interrupt is signaled to the MPC7410 by the assertion of an input signal. The system management interrupt signal ($\overline{\text{SMI}}$) is expected to remain asserted until the exception is taken. If $\overline{\text{SMI}}$ is negated early, recognition of the interrupt request is not guaranteed. After the MPC7410 begins execution of the system management interrupt handler, the system can safely negate $\overline{\text{SMI}}$. After the assertion of $\overline{\text{SMI}}$ is detected, the MPC7410 stops dispatching instructions and waits for all pending instructions to complete. This allows any instructions in progress that need to take an exception to do so before the system management interrupt exception is taken.

When a system management interrupt exception is taken, instruction fetching resumes as offset 0x01400 from the base address indicated by MSR[IP].

4.6.17 AltiVec Assist Exception (0x01600)

The MPC7410 implements an AltiVec assist exception to handle denormalized numbers in Java mode (VSCR[NJ] = 0). An AltiVec assist exception occurs when no higher priority exception exists and an instruction causes a trap condition as defined in Section 7.1.3, “Vector Floating Point Data Considerations.” Note that the MPC7410 handles most denormalized numbers in Java mode by taking a trap to the AltiVec assist exception, but for some instructions, the MPC7410 can produce the exact result without trapping.

Table 4-15 lists register settings when an AltiVec assist exception is taken.

Table 4-16. AltiVec Assist Exception—Register Settings

Register	Setting Description			
SRR0	Set to the effective address of the instruction that caused the exception.			
SRR1	0–5 Cleared 6 Loaded with equivalent MSR bit 7–15 Cleared 16–31Loaded with equivalent MSR bits			
MSR	VEC 0 POW 0 ILE — EE 0 LE Set to value of ILE	PR 0 FP 0 ME — FE0 0	SE 0 BE 0 FE1 0 IP —	IR 0 DR 0 PM 0 RI 0

Key: 0 Bit is cleared
ILE Bit is copied from the MSR[ILE]
— Bit is not altered

When an AltiVec assist exception is taken, instruction fetching resumes at offset 0x01600 from the base address indicated by MSR[IP].

4.6.18 Thermal Management Exception (0x01700)

A thermal management exception is generated when the junction temperature crosses a threshold programmed in either THRM1 or THRM2. The exception is enabled by the TIE bit of either THRM1 or THRM2, and can be masked by clearing MSR[EE].

Table 4-17 lists register settings when a thermal management exception is taken.

Table 4-17. Thermal Management Exception—Register Settings

<table>
<tr><th>Register</th><th colspan="4">Setting Description</th></tr>
<tr><td>SRR0</td><td colspan="4">Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.</td></tr>
<tr><td>SRR1</td><td colspan="4">0–5 Cleared
6 Loaded with equivalent MSR bit
7–15 Cleared
16–31Loaded with equivalent MSR bits</td></tr>
<tr><td>MSR</td><td>VEC 0
POW 0
ILE —
EE 0
LE Set to value of ILE</td><td>PR 0
FP 0
ME —
FE0 0</td><td>SE 0
BE 0
FE1 0
IP —</td><td>IR 0
DR 0
PM 0
RI 0</td></tr>
</table>

Key:	0	Bit is cleared
	ILE	Bit is copied from the MSR[ILE]
	—	Bit is not altered

The thermal management exception is similar to the system management and external interrupts. The MPC7410 requires the next instruction in program order to complete or take an exception, blocks completion of any following instructions, and allows the completed store queue to drain. Any exceptions encountered in this process are taken first and the thermal management exception is delayed until a recoverable halt is achieved, at which point the MPC7410 saves the machine state, as shown in Table 4-17.

When a thermal management exception is taken, instruction fetching resumes as offset 0x01700 from the base address indicated by MSR[IP].

Chapter 5
Memory Management

This chapter describes the MPC7410 microprocessor's implementation of the memory management unit (MMU) specifications provided by the operating environment architecture (OEA) for processors. The primary function of the MMU in a processor is the translation of logical (effective) addresses to physical addresses (referred to as real addresses in the architecture specification) for memory accesses and I/O accesses (I/O accesses are assumed to be memory-mapped). In addition, the MMU provides access protection on a segment, block, or page basis. This chapter describes the specific hardware used to implement the MMU model of the OEA in the MPC7410. Refer to Chapter 7, "Memory Management," in *The Programming Environments Manual* for a complete description of the conceptual model. Note that the MPC7410 does not implement the optional direct-store facility and it is not likely to be supported in future devices.

AltiVec Technology and the MMU Implementation

The AltiVec functionality in the MPC7410 affects the MMU model in the following ways:

- A data stream instruction (**dst**[**t**] or **dstst**[**t**]) can cause table search operations to occur after the instruction is retired.
- MMU exception conditions can cause a data stream operation to abort.
- Aborted VTQ-initiated table search operations can cause a line fetch skip.
- Execution of a **tlbsync** instruction can cancel an outstanding table search operation for a VTQ.

Two general types of memory accesses generated by processors require address translation—instruction accesses and data accesses generated by load and store instructions. Generally, the address translation mechanism is defined in terms of the segment descriptors and page tables processors use to locate the effective-to-physical address mapping for memory accesses. The segment information translates the effective address to an interim virtual address, and the page table information translates the interim virtual address to a physical address.

The segment descriptors, used to generate the interim virtual addresses, are stored as on-chip segment registers on 32-bit implementations (such as the MPC7410). In addition, two translation lookaside buffers (TLBs) are implemented on the MPC7410 to keep recently used page address translations on-chip. Although the OEA describes one MMU (conceptually), the MPC7410 hardware maintains separate TLBs and table search resources for instruction and data accesses that can be performed independently (and simultaneously). Therefore, the MPC7410 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, and in the MPC7410, they reside in the instruction and data MMUs, respectively.

The MMUs, together with the exception processing mechanism, provide the necessary support for the operating system to implement a paged virtual memory environment and for enforcing protection of designated memory areas. Exception processing is described in Chapter 4, “Exceptions.” Section 4.3, “Exception Processing,” describes the MSR, which controls some of the critical functionality of the MMUs.

5.1 MMU Overview

The MPC7410 implements the memory management specification of the OEA for 32-bit implementations. Thus, it provides 4 Gbytes of effective address space accessible to supervisor and user programs, with a 4-Kbyte page size and 256-Mbyte segment size. In addition, the MMUs of 32-bit processors use an interim virtual address (52 bits) and hashed page tables in the generation of 32-bit physical addresses. Processors also have a BAT mechanism for mapping large blocks of memory. Block sizes range from 128 Kbyte to 256 Mbyte and are software-programmable.

Basic features of the MPC7410 MMU implementation defined by the OEA are as follows:

- Support for real addressing mode—Effective-to-physical address translation can be disabled separately for data and instruction accesses.
- Block address translation—Each of the BAT array entries (four IBAT entries and four DBAT entries) provides a mechanism for translating blocks as large as 256 Mbytes from the 32-bit effective address space into the physical memory space. This can be used for translating large address ranges whose mappings do not change frequently.
- Segmented address translation—The 32-bit effective address is extended to a 52-bit virtual address by substituting 24 bits of upper address bits from the segment register, for the four upper bits of the EA, which are used as an index into the segment register file. This 52-bit virtual address space is divided into 4-Kbyte pages, each of which can be mapped to a physical page.

The MPC7410 processor also provides the following features that are not required by the architecture:

- Separate translation lookaside buffers (TLBs)—The 128-entry, two-way set-associative ITLBs and DTLBs keep recently used page address translations on-chip.
- Table search operations performed in hardware—The 52-bit virtual address is formed and the MMU attempts to fetch the PTE, which contains the physical address, from the appropriate TLB on-chip. If the translation is not found in a TLB (that is, a TLB miss occurs), the hardware performs a table search operation (using a hashing function) to search for the PTE.
- TLB invalidation—The MPC7410 implements the optional TLB Invalidate Entry (**tlbie**) and TLB Synchronize (**tlbsync**) instructions, which can be used to invalidate TLB entries. For more information on the **tlbie** and **tlbsync** instructions, see Section 5.4.3.2, "TLB Invalidation."

Table 5-1 summarizes the MPC7410 MMU features, including those defined by the architecture (OEA) for 32-bit processors and those specific to the MPC7410.

Table 5-1. MMU Feature Summary

Feature Category	Architecturally Defined/ MPC7410-Specific	Feature
Address ranges	Architecturally defined	2^{32} bytes of effective address
		2^{52} bytes of virtual address
		2^{32} bytes of physical address
Page size	Architecturally defined	4 Kbytes
Segment size	Architecturally defined	256 Mbytes
Block address translation	Architecturally defined	Range of 128 Kbyte–256 Mbyte sizes
		Implemented with IBAT and DBAT registers in BAT array
Memory protection	Architecturally defined	Segments selectable as no-execute
		Pages selectable as user/supervisor and read-only or guarded
		Blocks selectable as user/supervisor and read-only or guarded
Page history	Architecturally defined	Referenced and changed bits defined and maintained
Page address translation	Architecturally defined	Translations stored as PTEs in hashed page tables in memory
		Page table size determined by mask in SDR1 register
TLBs	Architecturally defined	Instructions for maintaining TLBs (**tlbie** and **tlbsync** instructions in MPC7410)
	MPC7410-specific	128-entry, two-way set associative ITLB 128-entry, two-way set associative DTLB LRU replacement algorithm
Segment descriptors	Architecturally defined	Stored as segment registers on-chip (two identical copies maintained)
Page table search support	MPC7410-specific	The MPC7410 performs the table search operation in hardware.

5.1.1 Memory Addressing

A program references memory using the effective (logical) address computed by the processor when it executes a load, store, branch, or cache instruction, and when it fetches the next instruction. The effective address is translated to a physical address according to the procedures described in Chapter 7, “Memory Management,” in *The Programming Environments Manual*, augmented with information in this chapter. The memory subsystem uses the physical address for the access.

For a complete discussion of effective address calculation, see Section 2.3.2.3, “Effective Address Calculation.”

5.1.2 MMU Organization

Figure 5-1 shows the conceptual organization of a PowerPC MMU in a 32-bit implementation; note that it does not describe the specific hardware used to implement the memory management function for a particular processor. Processors may optionally implement on-chip TLBs, hardware support for the automatic search of the page tables for PTEs, and other hardware features (invisible to the system software) not shown.

The MPC7410 processor maintains two on-chip TLBs with the following characteristics:

- 128 entries, two-way set associative (64 x 2), LRU replacement
- Data TLB supports the DMMU; instruction TLB supports the IMMU.
- Hardware TLB update
- Hardware update of referenced (R) and changed (C) bits in the translation table

In the event of a TLB miss, the hardware attempts to load the TLB based on the results of a translation table search operation.

Figure 5-2 and Figure 5-3 show the conceptual organization of the MPC7410 instruction and data MMUs, respectively. The instruction addresses shown in Figure 5-2 are generated by the processor for sequential instruction fetches and addresses that correspond to a change of program flow. Data addresses shown in Figure 5-3 are generated by load, store, and cache instructions.

As shown in the figures, after an address is generated, the high-order bits of the effective address, EA[0:19] (or a smaller set of address bits, EA[0:*n*], in the cases of blocks), are translated into physical address bits PA[0:19]. The low-order address bits, A[20:31], are untranslated and are therefore identical for both effective and physical addresses. After translating the address, the MMUs pass the resulting 32-bit physical address to the memory subsystem.

The MMUs record whether the translation is for an instruction or data access, whether the processor is in user or supervisor mode and, for data accesses, whether the access is a load or a store operation. The MMUs use this information to appropriately direct the address translation and to enforce the protection hierarchy programmed by the operating system. Section 4.3, “Exception Processing,” describes the MSR, which controls some of the critical functionality of the MMUs.

The figures show how address bits A[20:26] index into the on-chip instruction and data caches to select a cache set. The remaining physical address bits are then compared with the tag fields (comprised of bits PA[0:19]) of the two selected cache blocks to determine if a cache hit has occurred. In the case of a cache miss on the MPC7410, the instruction or data access is then forwarded to the L2 interface tags to check for an L2 cache hit. In case of a miss the access is forwarded to the bus interface unit which initiates an external memory access.

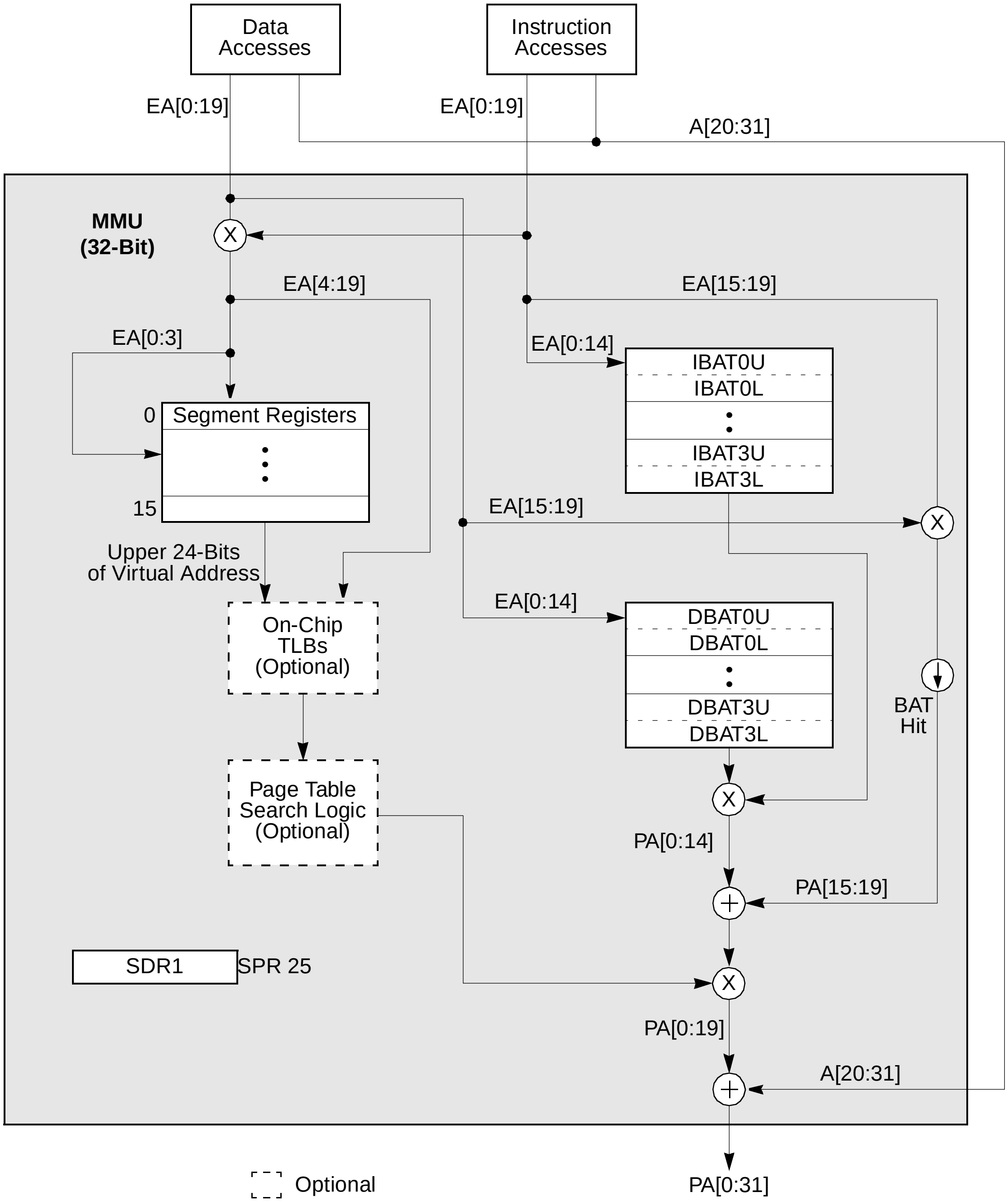


Figure 5-1. MMU Conceptual Block Diagram—32-Bit Implementations

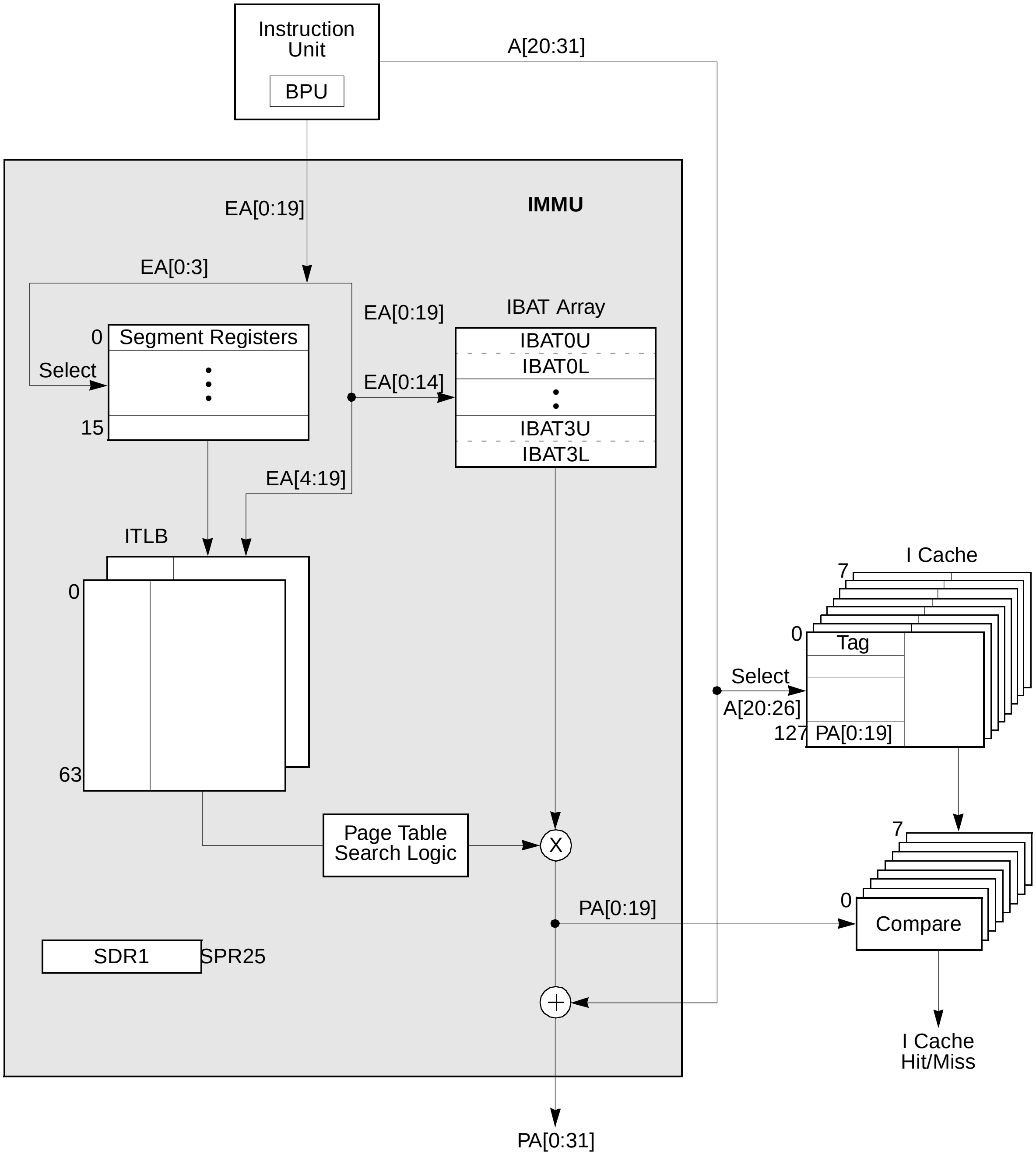


Figure 5-2. MPC7410 Microprocessor IMMU Block Diagram

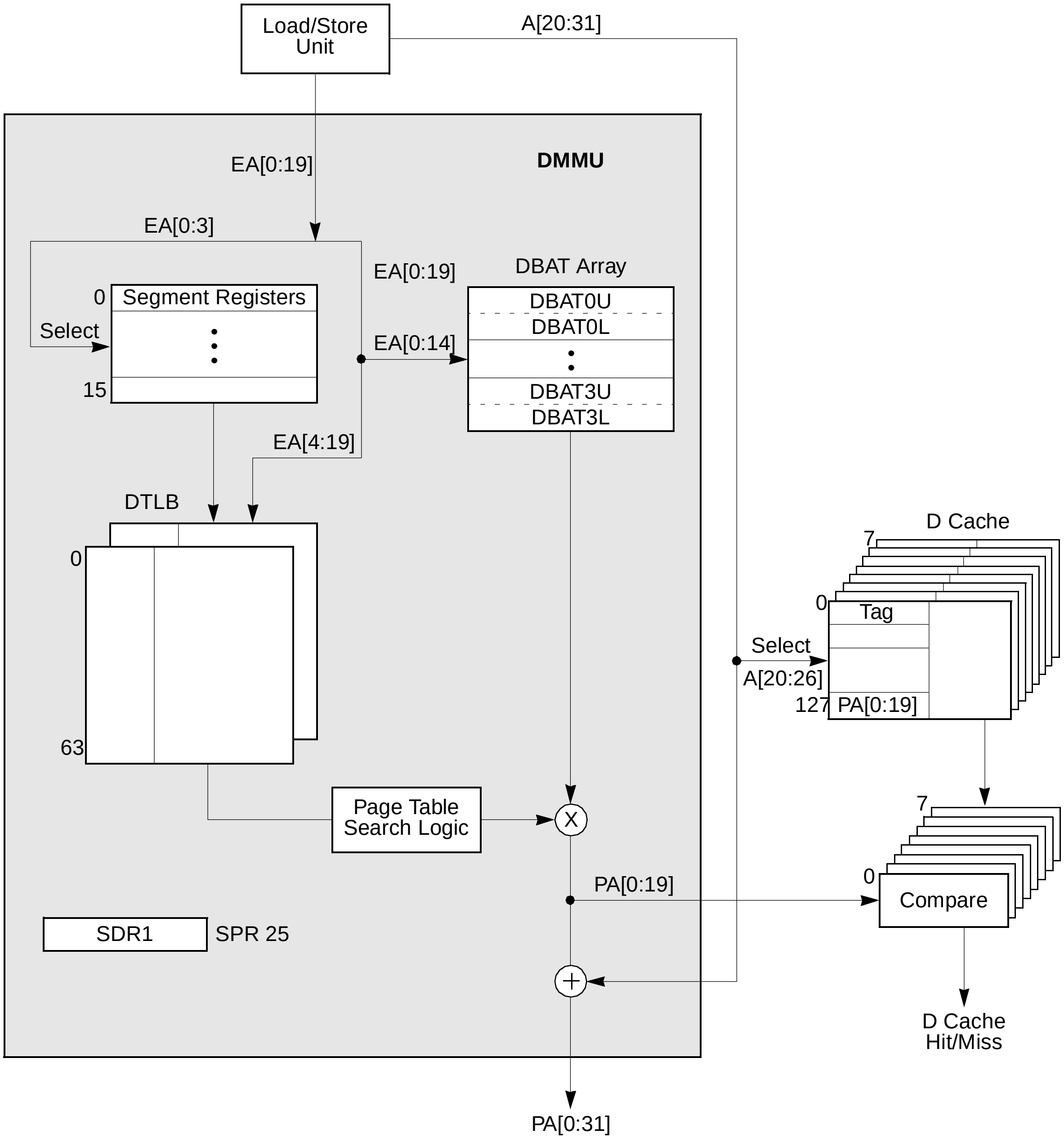


Figure 5-3. MPC7410 Microprocessor DMMU Block Diagram

5.1.3 Address Translation Mechanisms

Processors that implement the PowerPC ISA support the following three types of address translation:

- Page address translation—Translates the page frame address for a 4-Kbyte page size
- Block address translation—Translates the block number for blocks that range in size from 128 Kbytes to 256 Mbytes.
- Real addressing mode address translation—When address translation is disabled, the physical address is identical to the effective address.

Figure 5-4 shows the three address translation mechanisms provided by the MMUs. The segment descriptors shown in the figure control the page address translation mechanism. When an access uses page address translation, the appropriate segment descriptor is required. In 32-bit implementations, the appropriate segment descriptor is selected from the 16 on-chip segment registers by the four highest-order effective address bits.

A control bit in the corresponding segment descriptor then determines if the access is to memory (memory-mapped) or to the direct-store interface space. Note that the direct-store interface was present in the architecture only for compatibility with existing I/O devices that used this interface. However, it is being removed from the architecture, and the MPC7410 does not support it. When an access is determined to be to the direct-store interface space, the MPC7410 takes a DSI exception if it is a data access (see Section 4.6.3, “DSI Exception (0x00300)”), and takes an ISI exception if it is an instruction access (see Section 4.6.4, “ISI Exception (0x00400)”).

For memory accesses translated by a segment descriptor, the interim virtual address is generated using the information in the segment descriptor. Page address translation corresponds to the conversion of this virtual address into the 32-bit physical address used by the memory subsystem. In most cases, the physical address for the page resides in an on-chip TLB and is available for quick access. However, if the page address translation misses in the on-chip TLB, the MMU causes a search of the page tables in memory (using the virtual address information and a hashing function) to locate the required physical address.

Because blocks are larger than pages, there are fewer upper-order effective address bits to be translated into physical address bits (more low-order address bits—at least 17—are untranslated to form the offset into a block) for block address translation. Also, instead of segment descriptors and a TLB, block address translations use the on-chip BAT registers as a BAT array. If an effective address matches the corresponding field of a BAT register, the information in the BAT register is used to generate the physical address; in this case, the results of the page translation (occurring in parallel) are ignored.

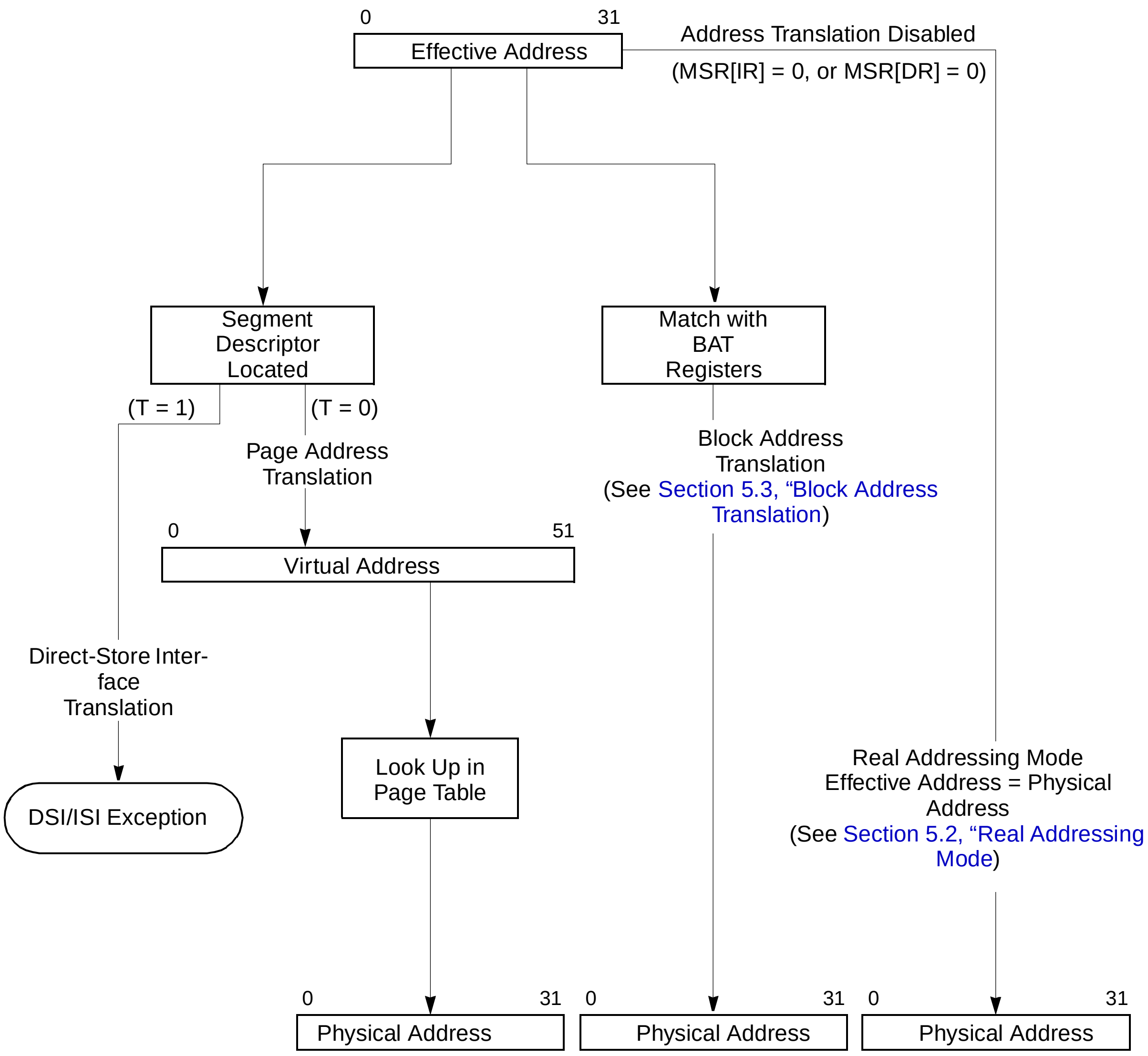

0
31
Effective Address
Address Translation Disabled
(MSR[IR] = 0, or MSR[DR] = 0)
Segment Descriptor Located
Match with BAT Registers
(T = 1)
(T = 0)
Page Address Translation
Block Address Translation
(See Section 5.3, “Block Address Translation)
0
51
Virtual Address
Direct-Store Interface Translation
DSI/ISI Exception
Look Up in Page Table
Real Addressing Mode
Effective Address = Physical Address
(See Section 5.2, “Real Addressing Mode)
0
31
Physical Address
0
31
Physical Address
0
31
Physical Address

Figure 5-4. Address Translation Types

When the processor generates an access, and the corresponding address translation enable bit in MSR is cleared, the resulting physical address is identical to the effective address and all other translation mechanisms are ignored. Instruction address translation and data address translation are enabled by setting MSR[IR] and MSR[DR], respectively.

5.1.4 Memory Protection Facilities

In addition to the translation of effective addresses to physical addresses, the MMUs provide access protection of supervisor areas from user access and can designate areas of memory as read-only as well as no-execute or guarded. Table 5-2 shows the protection options supported by the MMUs for pages.

Table 5-2. Access Protection Options for Pages

Option	User Read		User Write	Supervisor Read		Supervisor Write
	I-Fetch	Data		I-Fetch	Data	
Supervisor-only	—	—	—	√	√	√
Supervisor-only-no-execute	—	—	—	—	√	√
Supervisor-write-only	√	√	—	√	√	√
Supervisor-write-only-no-execute	—	√	—	—	√	√
Both (user/supervisor)	√	√	√	√	√	√
Both (user-/supervisor) no-execute	—	√	√	—	√	√
Both (user-/supervisor) read-only	√	√	—	√	√	—
Both (user/supervisor) read-only-no-execute	—	√	—	—	√	—

Key: √ Access permitted
— Protection violation

The no-execute option provided in the segment register lets the operating system program determine whether instructions can be fetched from an area of memory. The remaining options are enforced based on a combination of information in the segment descriptor and the page table entry. Thus, the supervisor-only option allows only read and write operations generated while the processor is operating in supervisor mode (MSR[PR] = 0) to access the page. User accesses that map into a supervisor-only page cause an exception.

Finally, a facility in the VEA and OEA allows pages or blocks to be designated as guarded, preventing out-of-order accesses that may cause undesired side effects. For example, areas of the memory map used to control I/O devices can be marked as guarded so accesses do not occur unless they are explicitly required by the program.

For more information on memory protection, see “Memory Protection Facilities,” in Chapter 7, “Memory Management,” in the *The Programming Environments Manual*.

5.1.5 Page History Information

The MMUs of processors also define referenced (R) and changed (C) bits in the page address translation mechanism that can be used as history information relevant to the page. The operating system can use these bits to determine which areas of memory to write back to disk when new pages must be allocated in main memory. While these bits are initially programmed by the operating system into the page table, the architecture specifies that they can be maintained either by the processor hardware (automatically) or by some software-assist mechanism.

When loading the TLB, the MPC7410 checks the state of the changed and referenced bits for the matched PTE. If the referenced bit is not set and the table search operation is initially caused by a load operation or by an instruction fetch, the MPC7410 automatically sets the referenced bit in the translation table. Similarly, if the table search operation is caused by a store operation and either the referenced bit or the changed bit is not set, the hardware automatically sets both bits in the translation table. In addition, when the address translation of a store operation hits in the DTLB, the MPC7410 checks the state of the changed bit. If the bit is not already set, the hardware automatically updates the DTLB and the translation table in memory to set the changed bit. For more information, see Section 5.4.1, “Page History Recording.”

5.1.6 General Flow of MMU Address Translation

The following sections describe the general flow used by processors to translate effective addresses to virtual and then physical addresses.

5.1.6.1 Real Addressing Mode and Block Address Translation Selection

When an instruction or data access is generated and the corresponding instruction or data translation is disabled (MSR[IR] = 0 or MSR[DR] = 0), real addressing mode is used (physical address equals effective address) and the access continues to the memory subsystem as described in Section 5.2, “Real Addressing Mode.”

Figure 5-5 shows the flow the MMUs use in determining whether to select real addressing mode, block address translation, or the segment descriptor to select page address translation.

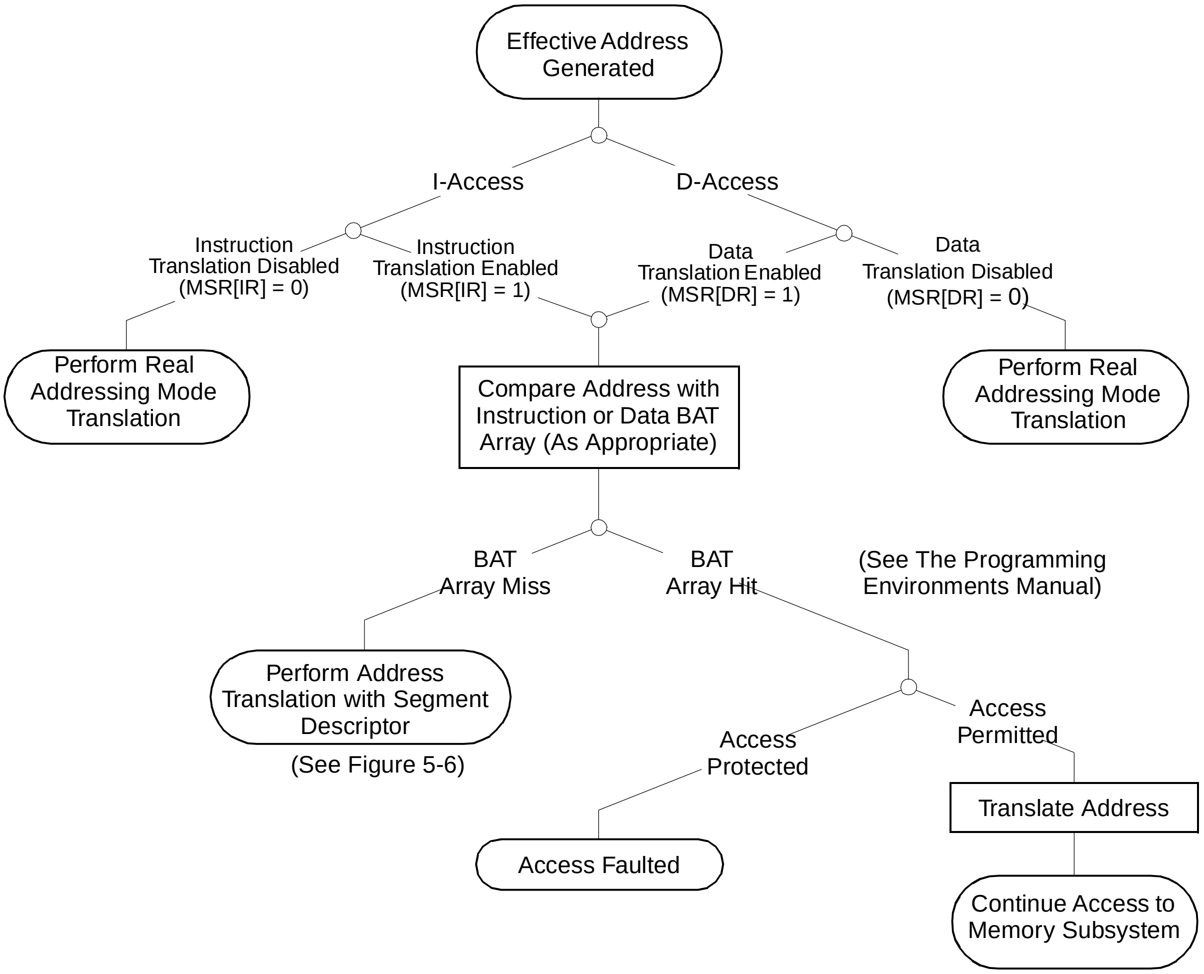

Effective Address Generated
I-Access
D-Access
Instruction Translation Disabled (MSR[IR] = 0)
Instruction Translation Enabled (MSR[IR] = 1)
Data Translation Enabled (MSR[DR] = 1)
Data Translation Disabled (MSR[DR] = 0)
Perform Real Addressing Mode Translation
Compare Address with Instruction or Data BAT Array (As Appropriate)
Perform Real Addressing Mode Translation
BAT Array Miss
BAT Array Hit
(See The Programming Environments Manual)
Perform Address Translation with Segment Descriptor
(See Figure 5-6)
Access Protected
Access Permitted
Translate Address
Access Faulted
Continue Access to Memory Subsystem

Figure 5-5. General Flow of Address Translation (Real Addressing Mode and Block)

Note that if the BAT array search results in a hit, the access is qualified with the appropriate protection bits. If the access violates the protection mechanism, an exception (ISI or DSI exception) is generated.

5.1.6.2 Page Address Translation Selection

If address translation is enabled and the effective address information does not match a BAT array entry, the segment descriptor must be located. When the segment descriptor is located, the T bit in the segment descriptor selects whether the translation is to a page or to a direct-store segment as shown in Figure 5-6. For 32-bit implementations, the segment descriptor for an access is contained in one of 16 on-chip segment registers; effective address bits EA[0:3] select one of the 16 segment registers.

Note that the MPC7410 does not implement the direct-store interface, and accesses to these segments cause a DSI or ISI exception. In addition, Figure 5-6 also shows the way in which the no-execute protection is enforced; if the N bit in the segment descriptor is set and the access is an instruction fetch, the access is faulted as described in Chapter 7, “Memory Management,” in *The Programming Environments Manual*. Note that the figure shows the flow for these cases as described by the OEA, and so the TLB references are shown as optional. Because the MPC7410 implements TLBs, these branches are valid and are described in more detail throughout this chapter.

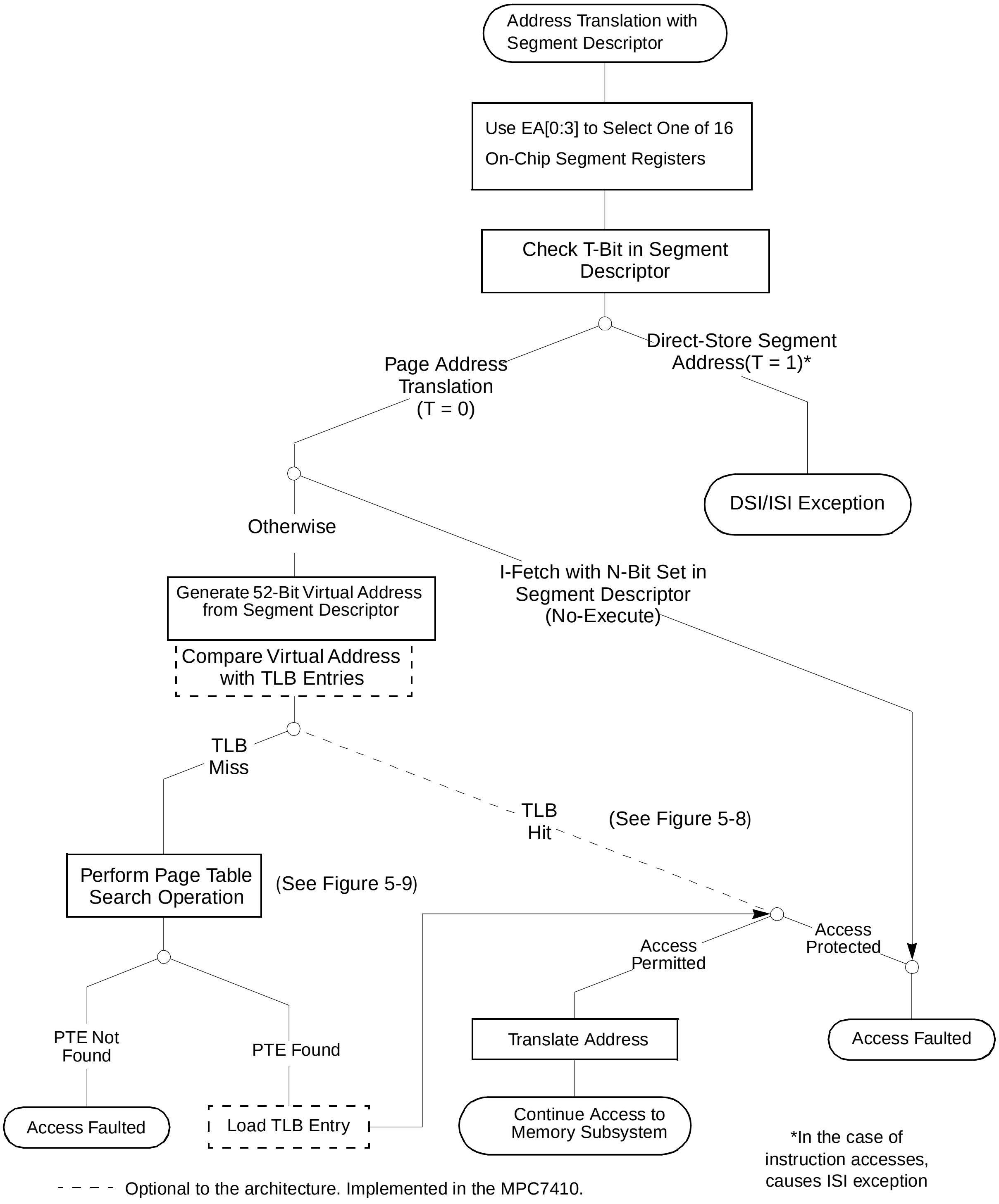

Address Translation with Segment Descriptor
Use EA[0:3] to Select One of 16 On-Chip Segment Registers
Check T-Bit in Segment Descriptor
Page Address Translation (T = 0)
Direct-Store Segment Address(T = 1)*
DSI/ISI Exception
Otherwise
I-Fetch with N-Bit Set in Segment Descriptor (No-Execute)
Generate 52-Bit Virtual Address from Segment Descriptor
Compare Virtual Address with TLB Entries
TLB Miss
TLB Hit
(See Figure 5-8)
Perform Page Table Search Operation
(See Figure 5-9)
Access Permitted
Access Protected
PTE Not Found
PTE Found
Translate Address
Access Faulted
Access Faulted
Load TLB Entry
Continue Access to Memory Subsystem
*In the case of instruction accesses, causes ISI exception
Optional to the architecture. Implemented in the MPC7410.

Figure 5-6. General Flow of Page and Direct-Store Interface Address Translation

If SR[T] = 0, page address translation is selected. The information in the segment descriptor is then used to generate the 52-bit virtual address. The virtual address is then used to identify the page address translation information (stored as page table entries (PTEs) in a page table in memory). For increased performance, the MPC7410 has two on-chip TLBs to cache recently used translations on-chip.

If an access hits in the appropriate TLB, page translation succeeds and the physical address bits are forwarded to the memory subsystem. If the required translation is not resident, the MMU performs a search of the page table. If the required PTE is found, a TLB entry is allocated and the page translation is attempted again. This time, the TLB is guaranteed to hit. When the translation is located, the access is qualified with the appropriate protection bits. If the access causes a protection violation, either an ISI or DSI exception is generated.

If the PTE is not found by the table search operation, a page fault condition exists, and an ISI or DSI exception occurs so software can handle the page fault.

5.1.7 MMU Exceptions Summary

To complete any memory access, the effective address must be translated to a physical address. As specified by the architecture, an MMU exception condition occurs if this translation fails for one of the following reasons:

- Page fault—There is no valid entry in the page table for the page specified by the effective address (and segment descriptor) and there is no valid BAT translation.
- An address translation is found but the access is not allowed by the memory protection mechanism.

The translation exception conditions defined by the OEA for 32-bit implementations cause either the ISI or the DSI exception to be taken as shown in Table 5-3.

The state saved by the processor for each of these exceptions contains information that identifies the address of the failing instruction. Refer to Chapter 4, “Exceptions,” for a more detailed description of exception processing.

Table 5-3. Translation Exception Conditions

Condition	Description	Exception
Page fault (no PTE found)	No matching PTE found in page tables (and no matching BAT array entry)	I access: ISI exception SRR1[1] = 1
		D access: DSI exception DSISR[1] =1
Block protection violation	Conditions described for block in “Block Memory Protection” in Chapter 7, “Memory Management,” in *The Programming Environments Manual.*“	I access: ISI exception SRR1[4] = 1
		D access: DSI exception DSISR[4] =1
Page protection violation	Conditions described for page in “Page Memory Protection” in Chapter 7, “Memory Management,” in *The Programming Environments Manual.*	I access: ISI exception SRR1[4] = 1
		D access: DSI exception DSISR[4] =1

Table 5-3. Translation Exception Conditions (continued)

Condition	Description	Exception
No-execute protection violation	Attempt to fetch instruction when SR[N] = 1	ISI exception SRR1[3] = 1
Instruction fetch from direct-store segment	Attempt to fetch instruction when SR[T] = 1	ISI exception SRR1[3] =1
Data access to direct-store segment (including floating-point accesses)	Attempt to perform load or store (including FP load or store) when SR[T] = 1	DSI exception DSISR[5] =1
Instruction fetch from guarded memory	Attempt to fetch instruction when MSR[IR] = 1 and either matching xBAT[G] = 1, or no matching BAT entry and PTE[G] = 1	ISI exception SRR1[3] =1

In addition to the translation exceptions, there are other MMU-related conditions (some of them defined as implementation-specific, and therefore not required by the architecture) that can cause an exception to occur. These exception conditions map to processor exceptions as shown in Table 5-4. The only MMU exception conditions that occur when MSR[DR] = 0 are those that cause an alignment exception for data accesses. For more detailed information about the conditions that cause an alignment exception (in particular for string/multiple instructions), see Section 4.6.6, “Alignment Exception (0x00600).”

Note that some exception conditions depend upon whether the memory area is set up as write-though (W = 1) or cache-inhibited (I = 1). These bits are described fully in “Memory/Cache Access Attributes,” in Chapter 5, “Cache Model and Memory Coherency,” of *The Programming Environments Manual*. Refer to Chapter 4, “Exceptions,” and to Chapter 6, “Exceptions,” in *The Programming Environments Manual* for a complete description of the SRR1 and DSISR bit settings for these exceptions.

For data accesses, the MPC7410 LSU initiates out-of-order accesses without knowledge of whether it is legal to do so. The MMU detects protection violations and **dcbz** alignment exceptions. The MMU prevents the changed bit in the PTE from being updated erroneously in these cases, but the LRU algorithm is updated. The MMU does not initiate exception processing for any exception conditions until the instruction that caused the exception is the next instruction to be retired. Also, the MPC7410 MMU does not perform a hardware table search operation due to TLB misses until the request is required by the program flow.

Table 5-4. Other MMU Exception Conditions for the MPC7410 Processor

Condition	Description	Exception
dcbz with W = 1 or I = 1	**dcbz** instruction to write-through or cache-inhibited segment or block	Alignment exception (not required by architecture for this condition)
lwarx, **stwcx.**, **eciwx**, or **ecowx** instruction to direct-store segment	Reservation instruction or external control instruction when SR[T] =1	DSI exception DSISR[5] =1
Floating-point load or store to direct-store segment	FP memory access when SR[T] =1	See data access to direct-store segment in Table 5-3.

Table 5-4. Other MMU Exception Conditions for the MPC7410 Processor

Condition	Description	Exception
Load or store that results in a direct-store error	Does not occur in MPC7410	Does not apply
eciwx or **ecowx** attempted when external control facility disabled	**eciwx** or **ecowx** attempted with EAR[E] = 0	DSI exception DSISR[11] = 1
lmw, **stmw**, **lswi**, **lswx**, **stswi**, or **stswx** instruction attempted in little-endian mode	**lmw**, **stmw**, **lswi**, **lswx**, **stswi**, or **stswx** instruction attempted while MSR[LE] = 1	Alignment exception
Operand misalignment	Translation enabled and a floating-point load/store, **stmw**, **stwcx.**, **lmw**, **lwarx**, **eciwx**, or **ecowx** instruction operand is not word-aligned	Alignment exception (some of these cases are implementation-specific)

5.1.8 MMU Instructions and Register Summary

The MMU instructions and registers allow the operating system to set up the block address translation areas and the page tables in memory.

Note that because the implementation of TLBs is optional, the instructions that refer to these structures are also optional. However, as these structures serve as caches of the page table, the architecture specifies a software protocol for maintaining coherency between these caches and the tables in memory whenever the tables in memory are modified. When the tables in memory are changed, the operating system purges these caches of the corresponding entries, allowing the translation caching mechanism to refetch from the tables when the corresponding entries are required.

Note that the MPC7410 implements all TLB-related instructions except **tlbia**, which is treated as an illegal instruction.

Because the MMU specification for processors is so flexible, it is recommended that the software that uses these instructions and registers be encapsulated into subroutines to minimize the impact of migrating across the family of implementations.

Table 5-5 summarizes MPC7410 instructions that specifically control the MMU. For more detailed information about the instructions, refer to Chapter 2, “Programming Model,” in this book and Chapter 8, “Instruction Set,” in *The Programming Environments Manual.*

Table 5-5. MPC7410 Microprocessor Instruction Summary—Control MMUs

Instruction[1]	Description
mtsr SR,**r**S	Move to Segment Register SR[SR#]← **r**S
mtsrin rS,**r**B	Move to Segment Register Indirect SR[**r**B[0–3]]←**r**S
mfsr rD,SR	Move from Segment Register **r**D←SR[SR#]
mfsrin rD,**r**B	Move from Segment Register Indirect **r**D←SR[**r**B[0–3]]

Table 5-5. MPC7410 Microprocessor Instruction Summary—Control MMUs (continued)

Instruction[1]	Description
tlbie rB*	TLB Invalidate Entry For effective address specified by **r**B, TLB[V]⟵0 The **tlbie** instruction invalidates all TLB entries indexed by the EA, and operates on both the instruction and data TLBs simultaneously invalidating four TLB entries. The index corresponds to bits 14–19 of the EA. In addition, execution of this instruction causes all entries in the congruence class corresponding to the EA to be invalidated in the other processors attached to the same bus. Software must ensure that instruction fetches or memory references to the virtual pages specified by the **tlbie** instruction have been completed prior to executing the **tlbie** instruction.
tlbsync*	TLB Synchronize Synchronizes the execution of all other **tlbie** instructions in the system. Specifically, this instruction causes a global (M = 1) TLBSYNC address-only transaction (TT[0:4] = 01001) on the bus. The TLBSYNC transaction terminates normally (without a retry) when all processors on the bus have completed pending TLB invalidations. See Section 5.4.3.2, “TLB Invalidation,” for more detailed information on the **tlbsync** instruction

[1] These instructions are defined by the architecture, but are optional.

Table 5-6 summarizes the registers that the operating system uses to program the MPC7410 MMUs. These registers are accessible to supervisor-level software only. These registers are described in Chapter 2, “Programming Model.”

Table 5-6. MPC7410 Microprocessor MMU Registers

Register	Description
Segment registers (SR0–SR15)	The sixteen 32-bit segment registers are present only in 32-bit implementations of the architecture. The fields in the segment register are interpreted differently depending on the value of bit 0. The segment registers are accessed by the **mtsr**, **mtsrin**, **mfsr**, and **mfsrin** instructions.
BAT registers (IBAT0U–IBAT3U, IBAT0L–IBAT3L, DBAT0U–DBAT3U, and DBAT0L–DBAT3L)	There are 16 BAT registers, organized as four pairs of instruction BAT registers (IBAT0U–IBAT3U paired with IBAT0L–IBAT3L) and four pairs of data BAT registers (DBAT0U–DBAT3U paired with DBAT0L–DBAT3L). The BAT registers are defined as 32-bit registers in 32-bit implementations. These are special-purpose registers that are accessed by the **mtspr** and **mfspr** instructions.
SDR1	The SDR1 register specifies the variables used in accessing the page tables in memory. SDR1 is defined as a 32-bit register for 32-bit implementations. This special-purpose register is accessed by the **mtspr** and **mfspr** instructions.

If an MMU register is being accessed by an instruction in the instruction stream, the IMMU stalls for one translation cycle to perform that operation. The sequencer serializes instructions to ensure the data correctness. Updates to the IBATs and SRs are classified as fetch serializing operations by the sequencer. After such an instruction is dispatched, the instruction buffer is flushed and the fetch stalls until the instruction completes. Reads from the IBATs are classified as execution serializing. Once the LSU ensures that all previous instructions can be executed, subsequent instructions can be fetched and dispatched.

5.2 Real Addressing Mode

If address translation is disabled (MSR[IR] = 0 or MSR[DR] = 0) for a particular access, the effective address is treated as the physical address and is passed directly to the memory subsystem as described in Chapter 7, "Memory Management," in *The Programming Environments Manual*.

Note that the default WIMG bits (0b0011) cause data accesses to be considered cacheable (I = 0) and thus load and store accesses are weakly ordered. This is the case even if the data cache is disabled in the HID0 register (as it is out of hard reset). If I/O devices require load and store accesses to occur in strict program order (strongly ordered), translation must be enabled so that the corresponding I bit can be set. Note also, that the G bit must be set to ensure that store operations are strongly ordered with other store operations and load operations are strongly ordered with other load operations For instruction accesses, the default memory access mode bits (WIMG) are also 0b0011. That is, instruction accesses are considered cacheable (I = 0), and the memory is guarded. Again, instruction accesses are considered cacheable even if the instruction cache is disabled in the HID0 register (as it is out of hard reset). The W and M bits have no effect on the instruction cache.

For information on the synchronization requirements for changes to MSR[IR] and MSR[DR], refer to Section 2.3.2.4, "Synchronization," in this manual, and "Synchronization Requirements for Special Registers and for Lookaside Buffers" in Chapter 2, "PowerPC Register Set," in *The Programming Environments Manual.*

5.3 Block Address Translation

The block address translation (BAT) mechanism in the OEA provides a way to map ranges of effective addresses larger than a single page into contiguous areas of physical memory. Such areas can be used for data that is not subject to normal virtual memory handling (paging), such as a memory-mapped display buffer or an extremely large array of numerical data.

Block address translation in the MPC7410 is described in Chapter 7, "Memory Management," in *The Programming Environments Manual* for 32-bit implementations.

The MPC7410 BAT registers are not initialized by the hardware after the power-up or reset sequence. Consequently, all valid bits in both instruction and data BAT areas must be explicitly cleared before setting any BAT area for the first time and before enabling translation. Also, note that software must avoid overlapping blocks while updating a BAT area or areas. Even if translation is disabled, multiple BAT area hits (with the valid bits set) can corrupt the remaining portion (any bits except the valid bits) of the BAT registers.

Thus, multiple BAT hits (with valid bits set) that map a given effective address to different physical addresses are considered a programming error whether translation is enabled or disabled. This can lead to unpredictable results if translation is enabled, or if translation is disabled when translation is eventually enabled. For the case of unused BATs, if translation is to be enabled, it is a sufficient precaution to simply clear the valid bits of the unused BAT entries.

5.4 Memory Segment Model

The MPC7410 adheres to the memory segment model as defined in Chapter 7, “Memory Management,” in *The Programming Environments Manual* for 32-bit implementations. Memory in the OEA is divided into 256-Mbyte segments. This segmented memory model provides a way to map 4-Kbyte pages of effective addresses to 4-Kbyte pages in physical memory (page address translation), while providing the programming flexibility afforded by a large virtual address space (52 bits).

The segment/page address translation mechanism may be superseded by the block address translation (BAT) mechanism described in Section 5.3, “Block Address Translation.” If not, the translation proceeds in the following two steps:

1. from effective address to the virtual address (which never exists as a specific entity but can be considered to be the concatenation of the virtual page number and the byte offset within a page), and
2. from virtual address to physical address.

This section highlights those areas of the memory segment model defined by the OEA that are specific to the MPC7410.

5.4.1 Page History Recording

Referenced (R) and changed (C) bits in each PTE keep history information about the page. They are maintained by a combination of the MPC7410 table search hardware and the system software. The operating system uses this information to determine which areas of memory to write back to disk when new pages must be allocated in main memory. Referenced and changed recording is performed only for accesses made with page address translation and not for translations made with the BAT mechanism or for accesses that correspond to direct-store (T = 1) segments. Furthermore, R and C bits are maintained only for accesses made while address translation is enabled (MSR[IR] = 1 or MSR[DR] = 1).

In the MPC7410, the referenced and changed bits are updated as follows:

- For TLB hits, the C bit is updated according to Table 5-7.
- For TLB misses, when a table search operation is in progress to locate a PTE. The R and C bits are updated (set, if required) to reflect the status of the page based on this access.

Table 5-7. Table Search Operations to Update History Bits—TLB Hit Case

R and C bits in TLB Entry	Processor Action
00	Combination doesn’t occur
01	Combination doesn’t occur
10	Read: No special action Write: The MPC7410 initiates a table search operation to update C.
11	No special action for read or write

Table 5-7 shows that the status of the C bit in the TLB entry (in the case of a TLB hit) is what causes the processor to update the C bit in the PTE (the R bit is assumed to be set in the page tables if there is a TLB hit). Therefore, when software clears the R and C bits in the page tables in memory, it must invalidate the TLB entries associated with the pages whose referenced and changed bits were cleared.

In some previous implementations, the **dcbt** and **dcbtst** instructions would execute only if there was a TLB/BAT hit or if the processor is in real addressing mode. In case of a TLB or BAT miss, these instructions would be treated as no-ops and did not initiate a table search operation and did not set either the R or C bits. In the MPC7410, the **dcbt**, **dcbtst**, and data stream touch instructions (**dst**[**t**] and **dstst**[**t**]) do cause a table search operation in the case of a TLB miss. However, they never cause the C bit to be set.

As defined by the architecture, the referenced and changed bits are updated as if address translation were disabled (real addressing mode). If these update accesses hit in the data cache, they are not seen on the external bus. If they miss in the data cache, they are performed as typical cache line fill accesses on the bus (if the data cache is enabled), or as discrete read and write accesses (if the data cache is disabled).

5.4.1.1 Referenced Bit

The referenced (R) bit of a page is located in the PTE in the page table. Every time a page is referenced (with a read or write access) and the R bit is zero, the MPC7410 sets the R bit in the page table. The OEA specifies that the referenced bit may be set immediately, or the setting may be delayed until the memory access is determined to be successful. Because the reference to a page is what causes a PTE to be loaded into the TLB, the referenced bit in all MPC7410 TLB entries is effectively always set. The processor never automatically clears the referenced bit.

The referenced bit is only a hint to the operating system about the activity of a page. At times, the referenced bit may be set although the access was not logically required by the program or even if the access was prevented by memory protection. Examples of this in PowerPC ISA systems include the following:

- Fetching of instructions not subsequently executed
- A memory reference caused by a speculatively executed instruction that is mispredicted
- Accesses generated by an **lswx** or **stswx** instruction with a zero length
- Accesses generated by an **stwcx.** instruction when no store is performed because a reservation does not exist
- Accesses that cause exceptions and are not completed

5.4.1.2 Changed Bit

The changed bit of a page is located both in the PTE in the page table and in the copy of the PTE loaded into the TLB (if a TLB is implemented, as in the MPC7410). Whenever a data store instruction is executed successfully, if the TLB search (for page address translation) results in a hit, the changed bit in the matching TLB entry is checked. If it is already set, it is not updated. If the TLB changed bit is 0, the MPC7410 initiates the table search operation to set the C bit in the corresponding PTE in the page table. The MPC7410 then reloads the TLB (with the C bit set).

The changed bit (in both the TLB and the PTE in the page tables) is set only when a store operation is allowed by the page memory protection mechanism and the store is guaranteed to be in the execution path (unless an exception, other than those caused by the **sc**, **rfi**, or trap instructions, occurs). Furthermore, the following conditions may cause the C bit to be set:

- The execution of an **stwcx.** instruction is allowed by the memory protection mechanism but a store operation is not performed.
- The execution of an **stswx** instruction is allowed by the memory protection mechanism but a store operation is not performed because the specified length is zero.
- The store operation is not performed because an exception occurs before the store is performed.

Again, note that the execution of the **dcbt**, **dcbtst** and data stream touch instructions (**dst**[**t**] and **dstst**[**t**]) never cause the C bit to be set.

5.4.1.3 Scenarios for Referenced and Changed Bit Recording

This section provides a summary of the model (defined by the OEA) that is used by processors for maintaining the referenced and changed bits. In some scenarios, the bits are guaranteed to be set by the processor, in some scenarios, the architecture allows that the bits may be set (not absolutely required), and in some scenarios, the bits are guaranteed to not be set. Note that when the MPC7410 updates the R and C bits in memory, the accesses are performed as if MSR[DR] = 0 and G = 0 (that is, as nonguarded cacheable operations in which coherency is required).

Table 5-8 defines a prioritized list of the R and C bit settings for all scenarios. The entries in the table are prioritized from top to bottom, such that a matching scenario occurring closer to the top of the table takes precedence over a matching scenario closer to the bottom of the table. For example, if an **stwcx.** instruction causes a protection violation and there is no reservation, the C bit is not altered, as shown for the protection violation case. Note that in the table, load operations include those generated by load instructions, by the **eciwx** instruction, and by the cache management instructions that are treated as a load with respect to address translation. Similarly, store operations include those operations generated by store instructions, by the **ecowx** instruction, and by the cache management instructions that are treated as a store with respect to address translation.

Table 5-8. Model for Guaranteed R and C Bit Settings

Priority	Scenario	Causes Setting of R Bit		Causes Setting of C Bit	
		OEA	MPC7410	OEA	MPC7410
1	No-execute protection violation	No	No	No	No
2	Page protection violation	Maybe	Yes	No	No
3	Out-of-order instruction fetch or load operation	Maybe	No	No	No
4	Out-of-order store operation. Would be required by the sequential execution model in the absence of system-caused or imprecise exceptions, or of floating-point assist exception for instructions that would cause no other kind of precise exception.	Maybe[1]	No	No	No
5	All other out-of-order store operations	Maybe[1]	No	Maybe[1]	No
6	Zero-length load (**lswx**)	Maybe	No	No	No
7	Zero-length store (**stswx**)	Maybe[1]	No	Maybe[1]	No

Table 5-8. Model for Guaranteed R and C Bit Settings (continued)

Priority	Scenario	Causes Setting of R Bit		Causes Setting of C Bit	
		OEA	MPC7410	OEA	MPC7410
8	Store conditional (**stwcx.**) that does not store	Maybe[1]	Yes	Maybe[1]	Yes
9	In-order instruction fetch	Yes[2]	Yes	No	No
10	Load instruction or **eciwx**	Yes	Yes	No	No
11	Store instruction, **ecowx** or **dcbz** instruction	Yes	Yes	Yes	Yes
12	**icbi**, **dcbt**, or **dcbtst** instruction	Maybe	No	No	No
13	**dcbst** or **dcbf** instruction	Maybe	Yes	No	No
14	**dcbi** instruction	Maybe[1]	Yes	Maybe[1]	Yes

[1] If C is set, R is guaranteed to be set also.

[2] Includes the case in which the instruction is fetched out of order and R is not set (does not apply for MPC7410).

For more information, see “Page History Recording” in Chapter 7, “Memory Management,” of *The Programming Environments Manual*.

5.4.2 Page Memory Protection

The MPC7410 implements page memory protection as it is defined in Chapter 7, “Memory Management,” in *The Programming Environments Manual*.

5.4.3 TLB Description

The MPC7410 implements separate 128-entry data and instruction TLBs to maximize performance. This section describes the hardware resources provided in the MPC7410 to facilitate page address translation. Note that the hardware implementation of the MMU is not specified by the architecture, and while this description applies to the MPC7410, it does not necessarily apply to other processors.

5.4.3.1 TLB Organization and Operation

Because the MPC7410 has two MMUs (IMMU and DMMU) that operate in parallel, some of the MMU resources are shared, and some are actually duplicated (shadowed) in each MMU to maximize performance. For example, although the architecture defines a single set of segment registers for the MMU, the MPC7410 maintains two identical sets of segment registers, one for the IMMU and one for the DMMU; when an instruction that updates the segment register executes, the MPC7410 automatically updates both sets.

The TLB entries are on-chip copies of PTEs in the page tables in memory and are similar in structure. To uniquely identify a TLB entry as the required PTE, the TLB entry also contains four more bits of the page index, EA[10:13], called the extended API (EAPI) in addition to the API bits in of the PTE.

Each TLB contains 128 entries organized as a two-way set-associative array with 64 sets as shown in Figure 5-7 for the DTLB (the ITLB organization is the same). When an address is being translated, a set of two TLB entries is indexed in parallel with the access to a segment register. If the address in one of the two TLB entries is valid and matches the 40-bit virtual page number, that TLB entry contains the translation. If no match is found, a TLB miss occurs.

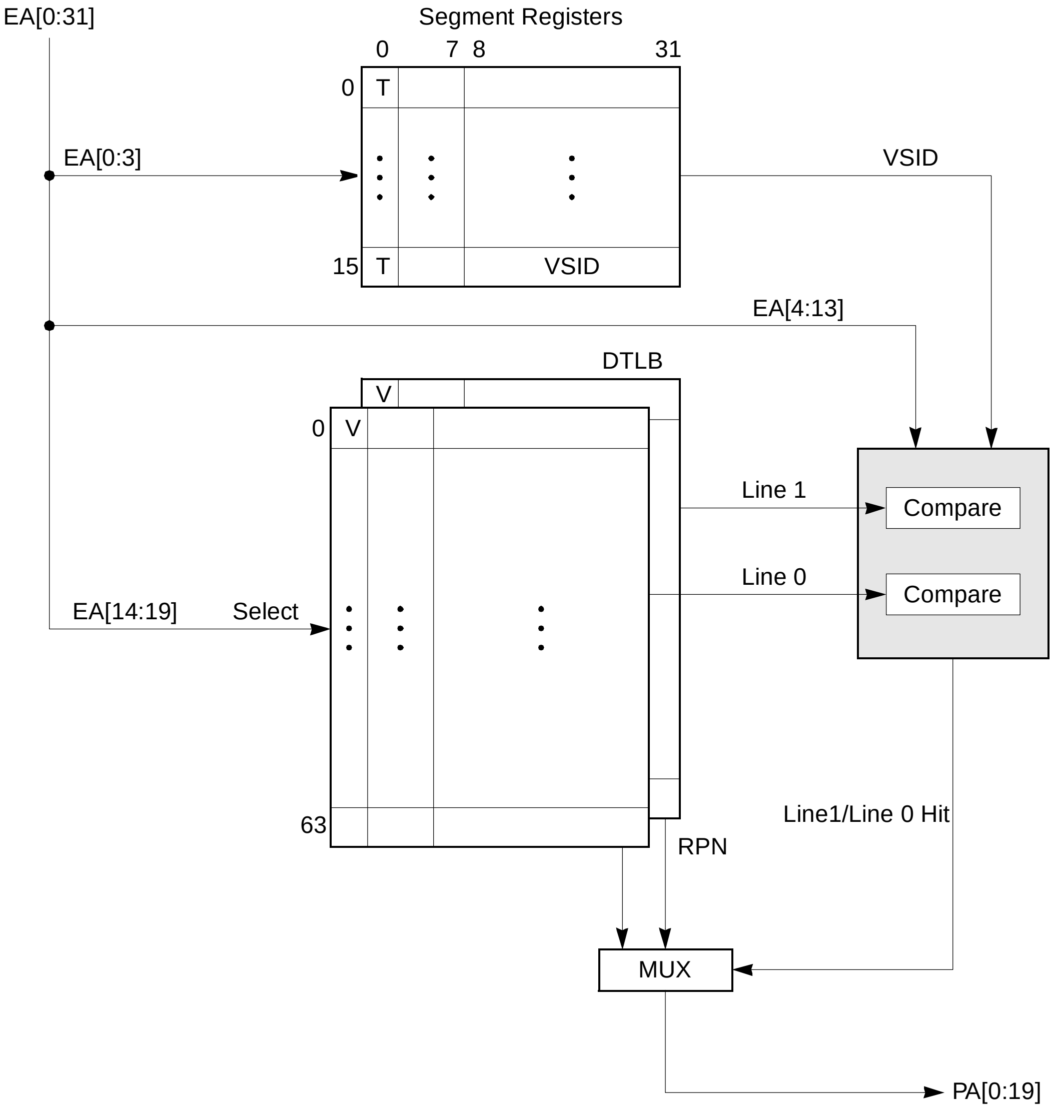


Figure 5-7. Segment Register and DTLB Organization

Unless the access is the result of an out-of-order access, a hardware table search operation begins if there is a TLB miss. If the access is out of order, the table search operation is postponed until the access is required, at which point the access is no longer out of order. When the matching PTE is found in memory, it is loaded into the TLB entry selected by the least-recently used (LRU) replacement algorithm, and the translation process begins again, this time with a TLB hit.

Software cannot access the TLB arrays directly, except to invalidate an entry with the **tlbie** instruction.

Each set of TLB entries has one associated LRU bit. The LRU bit for a set is updated any time either entry is used, even if the access is speculative. Invalid entries are always the first to be replaced.

Although both MMUs can be accessed simultaneously (both sets of segment registers and TLBs can be accessed in the same clock), only one exception condition is reported at a time. Exceptions are processed in strict program order, and a particular exception is processed when the instruction that caused it is the next instruction to be retired. When a particular instruction causes an instruction MMU exception, that exception is processed before that instruction can cause a data MMU exception.

ITLB miss conditions are reported when there are no more instructions to be dispatched or retired (the pipeline is empty), and DTLB miss conditions are reported when the load or store instruction is the next instruction to be retired. In the case that both an ITLB and DTLB miss are reported in the same clock, the DTLB miss takes precedence and is handled first. Refer to Chapter 6, "Instruction Timing," for more detailed information about the internal pipelines and the reporting of exceptions.

Although address translation is disabled on a soft or hard reset condition, the valid bits of TLB entries are not automatically cleared. Thus, TLB entries must be explicitly cleared by the system software (with the **tlbie** instruction) before address translation is enabled. Also, note that the segment registers do not have a valid bit, and so they should also be initialized before translation is enabled.

5.4.3.2 TLB Invalidation

The MPC7410 implements the optional **tlbie** and **tlbsync** instructions, which are used to invalidate TLB entries.

The **tlbia** instruction is not implemented on the MPC7410 and when its opcode is encountered, an illegal instruction program exception is generated. To invalidate all entries of both TLBs, 64 **tlbie** instructions must be executed, incrementing the value in EA14–EA19 by one each time. See Chapter 8, "Instruction Set," in *The Programming Environments Manual* for architecture information about the **tlbie** instruction.

5.4.3.2.1 tlbie Instruction

The execution of the **tlbie** instruction always invalidates four entries—both the ITLB and DTLB entries indexed by EA[14:19]. The **tlbie** instruction executes regardless of the setting of the MSR[DR] and MSR[IR] bits.

The architecture allows **tlbie** to optionally enable a TLB invalidate signaling mechanism in hardware so that other processors also invalidate their resident copies of the matching PTE. When an MPC7410 processor executes a **tlbie** instruction it always broadcasts this operation on the system bus as a global (M = 1) TLBIE address-only transaction (TT[0:4] = 11000) with the 32-bit effective (not physical) address reflected on the address bus. Figure 5-8 shows the flow of events caused by execution of the **tlbie** instruction as well as the actions taken by the MPC7410 when a TLBIE transaction is detected on the processor bus.

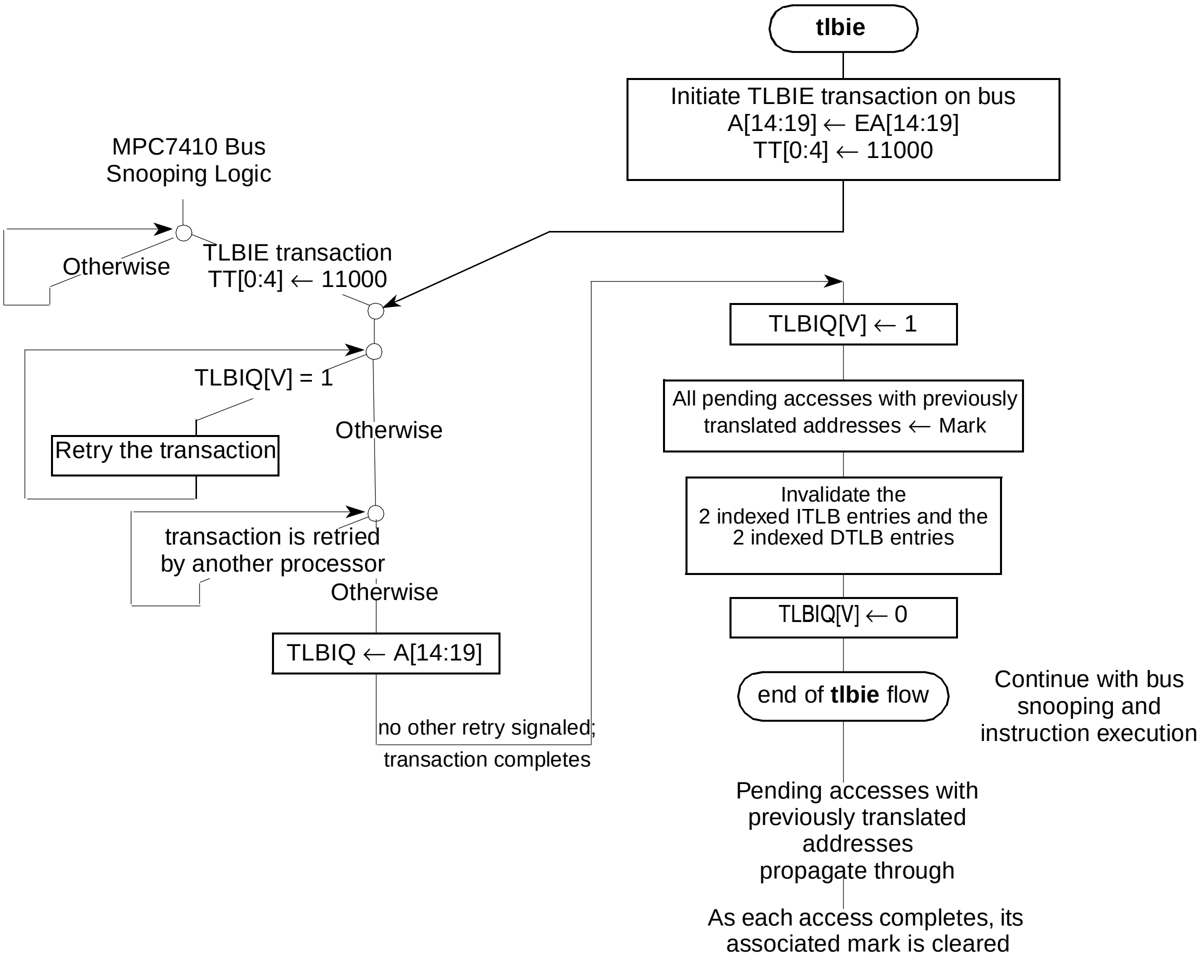


Figure 5-8. tlbie Instruction Execution and Bus Snooping Flow

The execution of the **tlbie** instruction is performed as if the TLBIE operation was snooped from the system bus by loading a single-entry TLBIQ that contains EA[14:19] and a valid bit. When the invalidation of the TLBs is complete, the TLBIQ is invalidated. Also, all valid queues in the machine that contain a previously translated address (physical address) are internally marked because these queues could contain references to addresses from the just invalidated TLB entries. These references propagate through to completion, but are marked for the purposes of synchronizing multiple TLB invalidations in multiple processors. See Section 5.4.3.2.2, "tlbsync Instruction," for more information on the use of these internal marks.

When another processor on the system bus performs a TLBIE address-only transaction, the MPC7410 snoops the transaction and checks the status of its internal TLBIQ. If the TLBIQ is valid (that is, the processor is in the process of performing a TLB invalidation), it causes a retry of the transaction until the TLBIQ empties. If the TLBIQ is invalid and the transaction is not retried by any other processor, the MPC7410 loads the TLBIQ with EA[14:19] and sets the TLBIQ valid bit. This causes the MPC7410 to invalidate the four TLB entries (both the ITLB and DTLB entries indexed by EA[14:19]), and internally mark all accesses with previously translated addresses.

The **tlbie** instruction does not affect the instruction fetch operation—that is, the prefetch buffer is not purged and the machine does not cause these instructions to be refetched.

5.4.3.2.2 tlbsync Instruction

The **tlbsync** instruction ensures that all previous **tlbie** instructions executed by the system have completed. Specifically, **tlbsync** causes a global (M = 1) TLBSYNC address-only transaction (TT[0:4] = 01001) on the bus if that processor has completed all previous **tlbie** instructions and any memory operations based on the contents of those invalidated TLB entries have propagated through to completion.

Execution of a **tlbsync** instruction affects outstanding VTQ operations in the same way as a **sync** instruction, (see Chapter 7, "AltiVec Technology Implementation") with the following additional effect: an outstanding table search operation for a VTQ-initiated access is cancelled when **tlbsync** is dispatched to the LSU, possibly causing a line fetch skip as described in Section 5.4.5, "Page Table Search Operation."

The **tlbsync** instruction does not complete until it is the oldest instruction presented to the on-chip memory subsystem. This occurs when all of the following conditions exist:

- The **tlbsync** instruction is the oldest instruction in the store queue,
- The instruction and data cache reload tables are idle, and
- There are no outstanding table search operations (note that a table search operation for a VTQ-initiated access may have been cancelled as described above).

Figure 5-9 shows the flow of events caused by execution of the **tlbsync** instruction as well as the actions taken by the MPC7410 when a TLBSYNC transaction is detected on the processor bus.

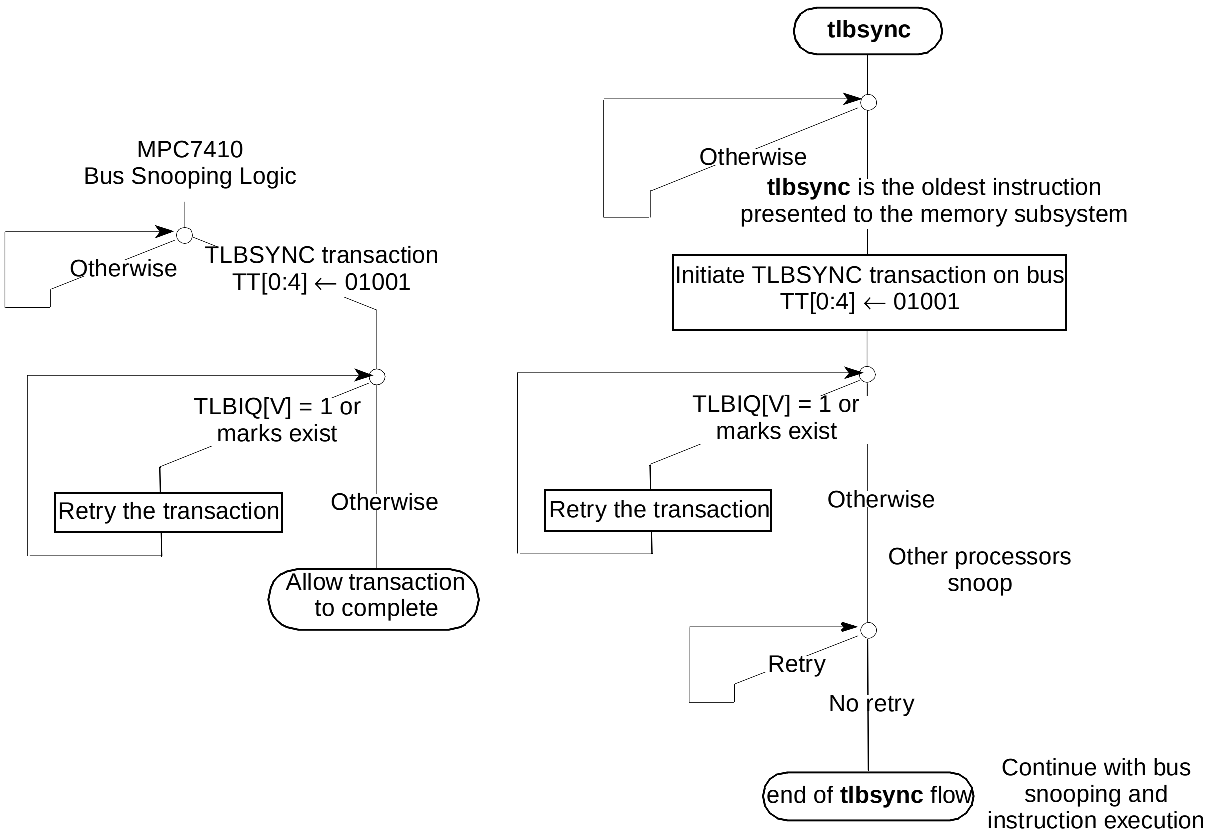


Figure 5-9. tlbsync Instruction Execution and Bus Snooping Flow

When an MPC7410 processor detects a TLBSYNC broadcast transaction, it causes a retry of that transaction until all pending TLB invalidate operations have completed. In this snoop process, the MPC7410 checks its TLBIQ and any pending marks for previously translated addresses. If the queue is valid or if any marks exist, the TLBSYNC transaction is retried, until the queue is invalid (idle) and no marks exist.

5.4.3.2.3 Synchronization Requirements for tlbie and tlbsync

In order to guarantee that a particular MPC7410 processor executing a **tlbie** instruction has completed the operation, a **sync** instruction must be placed after the **tlbie** instruction. A **tlbsync** instruction can also be used instead of the **sync** instruction for this purpose, but a **sync** will suffice for that processor. However, in order to guarantee that all MPC7410 processors in a system have coherently invalidated their respective TLB entries due to a **tlbie** instruction executing on any one of those processors, a **tlbsync** instruction is required.

The architecture requires that when a **tlbsync** instruction has been executed by a processor, a **sync** instruction must be executed by that processor before a **tlbie** or **tlbsync** instruction is executed by another processor. If this requirement is not met, a livelock situation may occur in a system with multiple

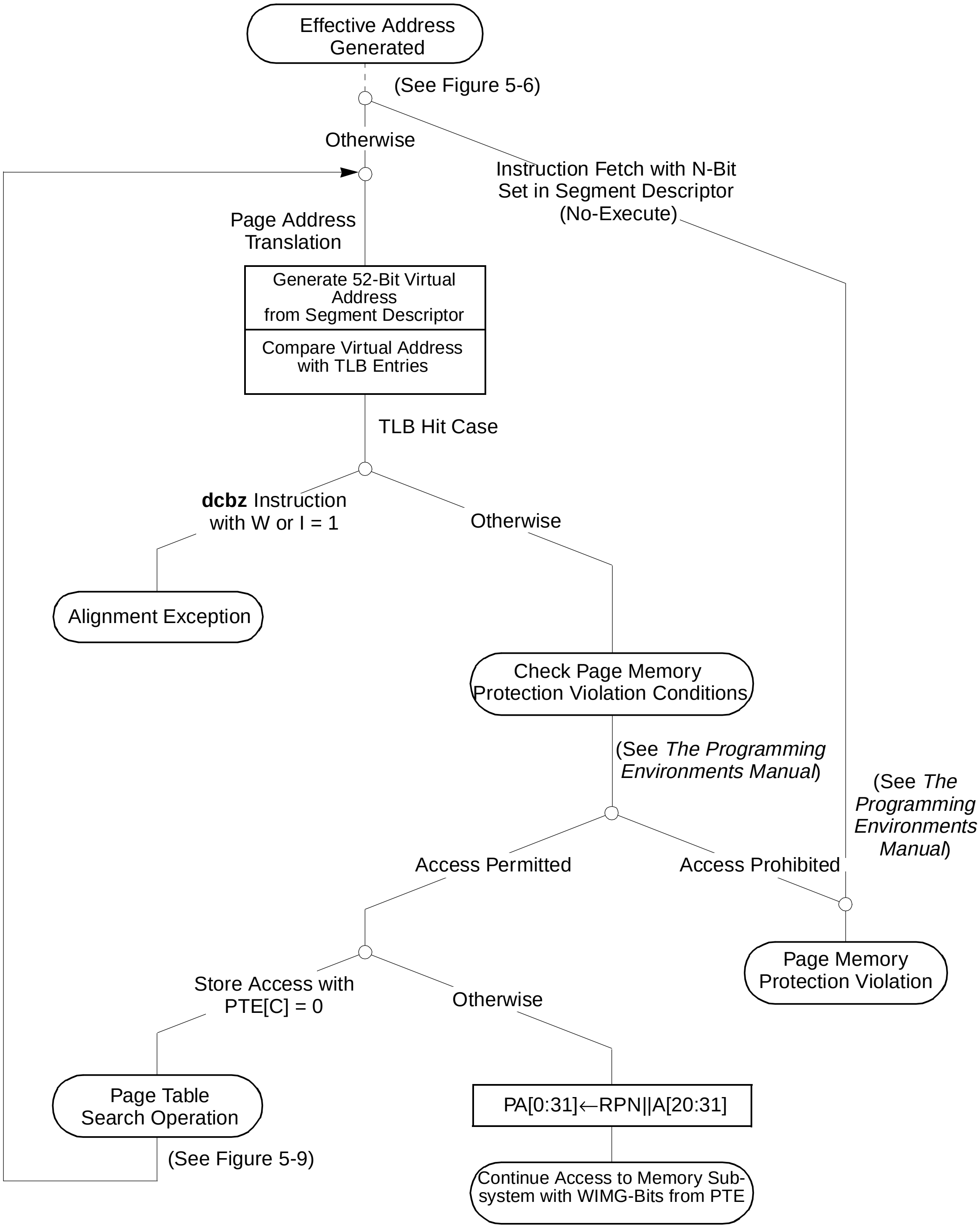
MPC7410 processors. Specifically, if more than one processor executes **tlbie** or **tlbsync** instructions simultaneously, it is likely that these processors will cause a system livelock.

5.4.4 Page Address Translation Summary

Figure 5-10 provides the detailed flow for the page address translation mechanism.

When an instruction or data access occurs, the effective address is routed to the appropriate MMU. EA0–EA3 select one of the 16 segment registers and the remaining effective address bits and the VSID field from the segment register are passed to the TLB. EA[14:19] then select two entries in the TLB; the valid bits are checked and the 40-bit virtual page number (24-bit VSID concatenated with EA4:EA19]) must match the VSID, EAPI, and API fields of the TLB entries. If one of the entries hits, the PP bits are checked for a protection violation. If these bits do not cause an exception, the C bit is checked. If the C bit must be updated, a table search operation is initiated. If the C bit does not require updating, the RPN value is passed to the memory subsystem and the WIMG bits are then used as attributes for the access.

Figure 5-10 includes the checking of the N bit in the segment descriptor and then expands on the ‘TLB Hit’ branch of Figure 5-6. The detailed flow for the ‘TLB Miss’ branch of Figure 5-6 is described in Section 5.4.5, “Page Table Search Operation.” Note that as in the case of block address translation, if an attempt is made to execute a **dcbz** instruction to a page marked either write-through or caching-inhibited (W = 1 or I = 1), an alignment exception is generated. The checking of memory protection violation conditions is described in Chapter 7, “Memory Management,” in *The Programming Environments Manual*.



Effective Address Generated
(See Figure 5-6)
Otherwise
Instruction Fetch with N-Bit Set in Segment Descriptor (No-Execute)
Page Address Translation
Generate 52-Bit Virtual Address from Segment Descriptor
Compare Virtual Address with TLB Entries
TLB Hit Case
dcbz Instruction with W or I = 1
Otherwise
Alignment Exception
Check Page Memory Protection Violation Conditions
(See *The Programming Environments Manual*)
(See *The Programming Environments Manual*)
Access Permitted
Access Prohibited
Page Memory Protection Violation
Store Access with PTE[C] = 0
Otherwise
Page Table Search Operation
(See Figure 5-9)
PA[0:31]←RPN||A[20:31]
Continue Access to Memory Sub-system with WIMG-Bits from PTE

Figure 5-10. Page Address Translation Flow—TLB Hit

5.4.5 Page Table Search Operation

If the translation is not found in the TLBs (a TLB miss), the MPC7410 initiates a table search operation which is described in this section. Formats for the PTE are given in “PTE Format for 32-Bit Implementations,” in Chapter 7, “Memory Management,” of *The Programming Environments Manual*.

5.4.5.1 Conditions for a Page Table Search Operation

For instruction accesses, the MPC7410 processor does not initiate a table search operation for an ITLB miss until the completion buffer is empty and the completed store queue is empty. Also, the instruction buffer must be empty, and there must be no other exceptions pending.

Also, the MMU does not perform a hardware table search due to DTLB misses (or to modify the C bit) until the access is absolutely required by the program flow and there are no other exceptions pending.

In the MPC7410, a TLB miss (and subsequent page table search operation) occurs transparently to the program. Thus, if a TLB miss occurs as a misaligned access crosses a translation boundary, the second portion of the misaligned access is completed automatically once the table search operation completes successfully. If the table search operation results in a page fault, an exception occurs and upon returning from the page fault handling routine, the entire misaligned access is restarted beginning with the first portion of the access.

Note that, as described in Chapter 6, “Instruction Timing,” store gathering does not occur while a page table search operation is in progress.

The AltiVec data stream touch instructions (**dst**[**t**] and **dstst**[**t**]) provide the ability to prefetch up to 128 Kbytes of data per instruction. As described in Chapter 6, “Instruction Timing,” a **dst**[**t**] or **dstst**[**t**] instruction can be retired from the completion buffer as soon as the instruction is loaded into the vector touch queue (VTQ). However, if a line fetch in the VTQ requires a table search operation before the instruction is retired, then the table search operation is delayed until the instruction is retired. If a line fetch in the VTQ requires a table search operation after the instruction has been retired, the table search operation is initiated immediately.

To further increase performance, the VTQ stream engines operate in parallel with the other execution units. Thus, the TLBs are non-blocking, and are available to the instruction unit and LSU for both instruction and data address translation during a VTQ-initiated table search operation.

5.4.5.2 AltiVec Line Fetch Skipping

As described in Chapter 7, “AltiVec Technology Implementation,” there are many conditions (exceptions, etc.) that cause the stream fetch performed by a VTQ stream engine to abort. In the case of a VTQ-initiated table search operation, when an exception or interrupt condition occurs, the stream engine pauses, the line-fetch that caused the table search operation is effectively dropped, and no MMU exceptions are reported for this line-fetch. When the stream engine resumes operation, the next line fetch is attempted, causing a skip of one line fetch in the stream engine.

Also, when a **tlbsync** instruction is executed while a VTQ-initiated table search operation is in progress, that table search operation is aborted, potentially causing a line fetch skip.

5.4.5.3 Page Table Search Operation Flow

The following is a summary of the page table search process performed by the MPC7410:

1. The 32-bit physical address of the primary PTEG is generated as described in “Page Table Addresses” in Chapter 7, “Memory Management,” of *The Programming Environments Manual*.
2. The first PTE (PTE0) in the primary PTEG is read from memory. PTE reads occur with an implied WIM memory/cache mode control bit setting of 0b001. Therefore, they are considered cacheable and read (burst) from memory and placed in the cache. Because the table search operation is never speculative and is cacheable, the G-bit has no effect
3. The PTE in the selected PTEG is tested for a match with the virtual page number (VPN) of the access. The VPN is the VSID concatenated with the page index field of the virtual address. For a match to occur, the following must be true:
   - PTE[H] = 0
   - PTE[V] = 1
   - PTE[VSID] = VA[0:23]
   - PTE[API] = VA[24:29]
4. If a match is not found, step 3 is repeated for each of the other seven PTEs in the primary PTEG. If a match is found, the table search process continues as described in step 8. If a match is not found within the 8 PTEs of the primary PTEG, the address of the secondary PTEG is generated.
5. The first PTE (PTE0) in the secondary PTEG is read from memory. Again, because PTE reads have a WIM bit combination of 0b001, an entire cache line is read into the on-chip cache.

6. The PTE in the selected secondary PTEG is tested for a match with the virtual page number (VPN) of the access. For a match to occur, the following must be true:
   - PTE[H] = 1
   - PTE[V] = 1
   - PTE[VSID] = VA[0:23]
   - PTE[API] = VA[24:29]
7. If a match is not found, step 6 is repeated for each of the other seven PTEs in the secondary PTEG. If it is never found, an exception is taken (step 9).
8. If a match is found, the PTE is written into the on-chip TLB and the R bit is updated in the PTE in memory (if necessary). If there is no memory protection violation, the C bit is also updated in memory (if the access is a write operation) and the table search is complete.
9. If a match is not found within the 8 PTEs of the secondary PTEG, the search fails, and a page fault exception condition occurs (either an ISI exception or a DSI exception).

Figure 5-11 and Figure 5-12 show how the conceptual model for the primary and secondary page table search operations, described in *The Programming Environments Manual*, are realized in the MPC7410.

Figure 5-11 shows the case of a **dcbz** instruction that is executed with W = 1 or I = 1, and that the R bit may be updated in memory (if required) before the operation is performed or the alignment exception occurs. The R bit may also be updated if memory protection is violated.

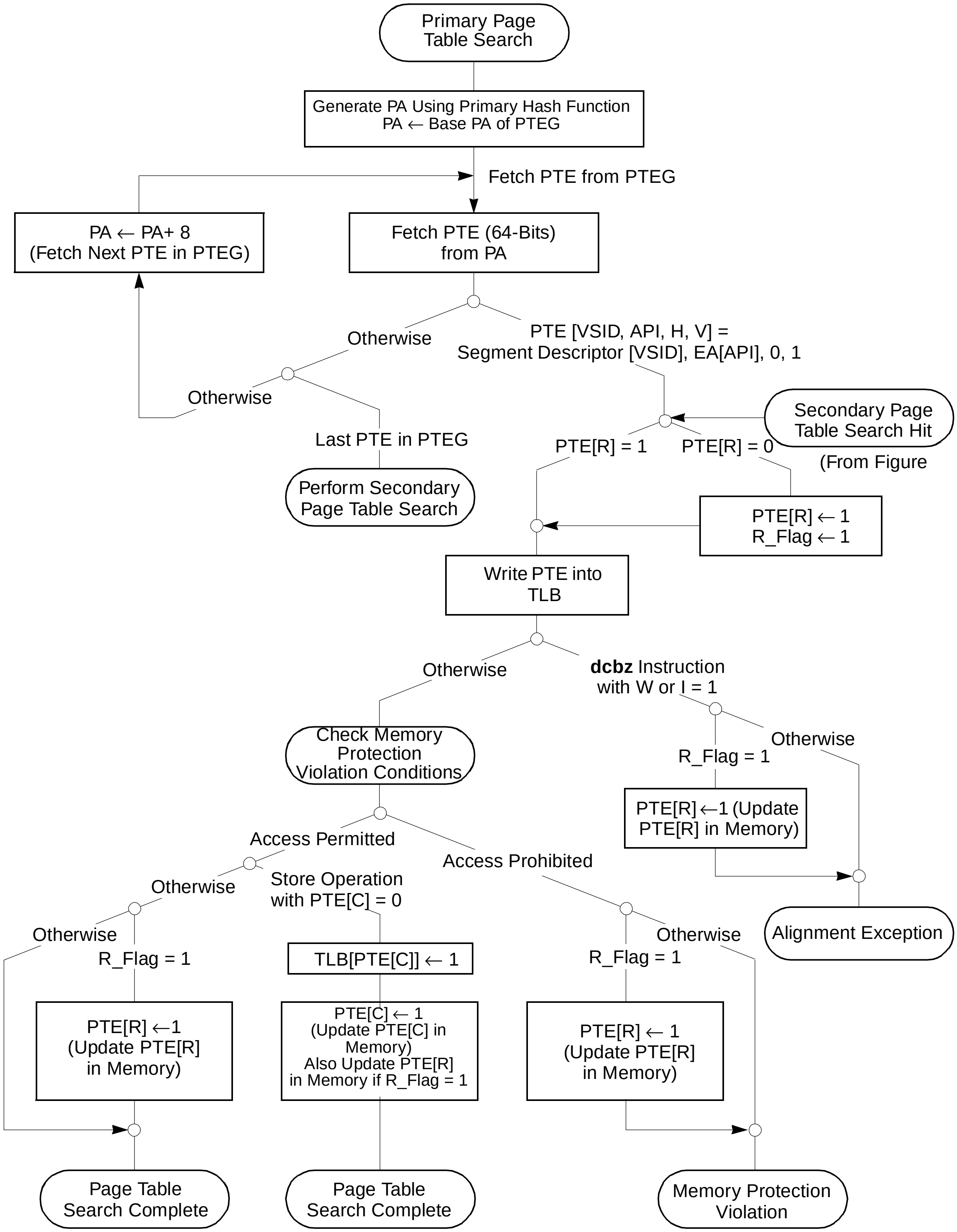


Figure 5-11. Primary Page Table Search

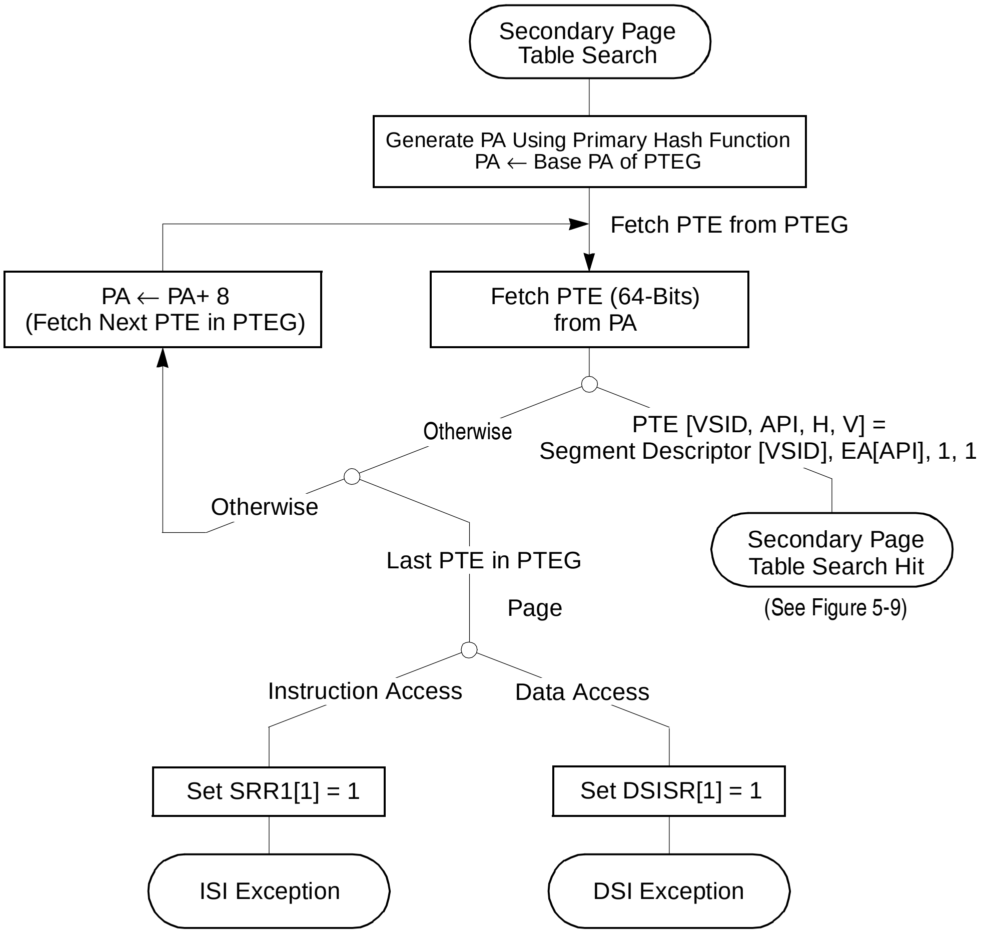


Figure 5-12. Secondary Page Table Search Flow

5.4.6 Page Table Updates

When TLBs are implemented (as in the MPC7410) they are defined as noncoherent caches of the page tables. TLB entries must be flushed explicitly with the TLB invalidate entry instruction (**tlbie**) whenever the corresponding PTE is modified.

Chapter 7, "Memory Management," in *The Programming Environments Manual* describes some required sequences of instructions for modifying the page tables. In a multiprocessor MPC7410 environment, PTEs can only be modified by adhering to the procedure for deleting a PTE, followed by the procedure for adding a PTE. Thus, the following code should be used:

```
/* Code for Modifying a Page Table Entry */
/* First delete the current page table entry */
PTE$_{V}$ <- 0         /* (other fields don’t matter) */
sync     /* ensure update completed */
tlbie(old_EA)/* invalidate old translation */
eieio    /* order tlbie before tlbsync */
tlbsync  /* ensure tlbie completed on all processors */
sync     /* ensure tlbsync completed */
/* Then add new PTE over old */
PTE$_{RPN,R,C,WIMG,PP}$ <- new values
eieio    /* order 1st PTE update before 2nd */
PTE$_{VSID,API,H,V}$ <- new values (V=1)
sync     /* ensure updates completed */
```

Processors may write referenced and changed bits with unsynchronized, atomic byte store operations. Note that the V, R, and C bits each reside in a distinct byte of a PTE. Therefore, extreme care must be taken to use byte writes when updating only one of these bits.

Explicitly altering certain MSR bits (using the **mtmsr** instruction), or explicitly altering PTEs, or certain system registers, may have the side effect of changing the effective or physical addresses from which the current instruction stream is being fetched. This kind of side effect is defined as an implicit branch. Implicit branches are not supported and an attempt to perform one causes boundedly-undefined results. Therefore, PTEs must not be changed in a manner that causes an implicit branch. Chapter 2, “PowerPC Register Set,” in *The Programming Environments Manual*, lists the possible implicit branch conditions that can occur when system registers and MSR bits are changed.

5.4.7 Segment Register Updates

Synchronization requirements for using the move to segment register instructions are described in “Synchronization Requirements for Special Registers and for Lookaside Buffers” in Chapter 2, “PowerPC Register Set,” in *The Programming Environments Manual*.

Chapter 6
Instruction Timing

This chapter describes how the MPC7410 microprocessor fetches, dispatches, and executes instructions and how it reports the results of instruction execution. It gives detailed descriptions of how the MPC7410 execution units work and how those units interact with other parts of the processor, such as the instruction fetching mechanism, register files, and caches. It gives examples of instruction sequences, showing potential bottlenecks and how to minimize their effects. Finally, it includes tables that identify the unit that executes each instruction implemented on the MPC7410, the latency for each instruction, and other information that is useful for the assembly language programmer.

AltiVec Technology and Instruction Timing

The AltiVec functionality in the MPC7410 affects instruction timing in the following ways:

- Additional execution units are provided for handling AltiVec permute (VPU) and ALU instructions (VALU)
- The VALU consists of three independent execution units:
  - Vector simple integer unit (VSIU). See Section 6.4.8.2.1, “Vector Simple Integer Unit (VSIU) Execution Timing.”
  - Vector complex integer unit (VCIU). See Section 6.4.8.2.2, “Vector Complex Integer Unit (VCIU) Execution Timing.”
  - Vector floating-point unit (VFPU). See Section 6.4.8.2.3, “Vector Floating-Point Unit (VFPU) Execution Timing.”
- The AltiVec technology defines data streaming instruction that allows automated loading of data for nonspeculative accesses. These instructions can be identified as either static (likely to be reused) or transient (unlikely to be reused). See Section Chapter 7, “AltiVec Technology Implementation.”
- The AltiVec technology defines load and store instructions that can be identified as least-recently-used, in order to free up data with low likelihood for reuse. See Section 6.4.7.1, “LRU Instructions.”
- Latencies for AltiVec instructions are listed in Table 6-9

6.1 Terminology and Conventions

This section provides an alphabetical glossary of terms used in this chapter. These definitions are provided as a review of commonly used terms and as a way to point out specific ways these terms are used in this chapter.

- Branch prediction—The process of guessing whether a branch will be taken. Such predictions can be correct or incorrect; the term ‘predicted’ as it is used here does not imply that the prediction is correct (successful). The architecture defines a means for static branch prediction as part of the instruction encoding.
- Branch resolution—The determination of whether a branch is taken or not taken. A branch is said to be resolved when the processor can determine which instruction path to take. If the branch is resolved as predicted, the instructions following the predicted branch that may have been speculatively executed can complete (see completion). If the branch is not resolved as predicted, instructions on the mispredicted path, and any results of speculative execution, are purged from the pipeline and fetching continues from the nonpredicted path.
- Completion—Completion occurs when an instruction has finished executing, written back any results, and is removed from the completion queue (CQ). When an instruction completes, it is guaranteed that this instruction and all previous instructions can cause no exceptions.
- Fall-through (branch fall-through)—A not-taken branch. On the MPC7410, fall-through branch instructions are removed from the instruction stream at dispatch. That is, these instructions are allowed to fall through the instruction queue via the dispatch mechanism, without either being passed to an execution unit and or given a position in the CQ.
- Fetch—The process of bringing instructions from memory (such as a cache or system memory) into the instruction queue. In this chapter, the fetch stage is considered to end when the instruction is dispatched.
- Folding (branch folding)—The replacement with target instructions of a branch instruction and any instructions along the not-taken path when a branch is either taken or predicted as taken.
- Finish—Finishing occurs in the last cycle of execution. In this cycle, the CQ entry is updated to indicate that the instruction has finished executing.
- Latency— The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.
- Pipeline—In the context of instruction timing, the term ‘pipeline’ refers to the interconnection of the stages. The events necessary to process an instruction are broken into several cycle-length tasks to allow work to be performed on several instructions simultaneously—analogous to an assembly line. As an instruction is processed, it passes from one stage to the next. When it does, the stage becomes available for the next instruction.

  Although an individual instruction may take many cycles to complete (the number of cycles is called instruction latency), pipelining makes it possible to overlap the processing so that the throughput (number of instructions completed per cycle) is greater than if pipelining were not implemented.
- Program order—The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the cache.

- Rename register—Temporary buffers used by instructions that have finished execution but have not completed.
- Reservation station—A buffer between the dispatch and execute stages that allows instructions to be dispatched even though the results of instructions on which the dispatched instruction may depend are not available.
- Retirement—Removal of the completed instruction from the CQ.
- Stage—The term ‘stage’ is used in two different senses, depending on whether the pipeline is being discussed as a physical entity or a sequence of events. In the latter case, a stage is an element in the pipeline during which certain actions are performed, such as decoding the instruction, performing an arithmetic operation, or writing back the results. Typically, the latency of a stage is one processor clock cycle. Some events, such as dispatch, write-back, and completion, happen instantaneously and may be thought to occur at the end of a stage.

  An instruction can spend multiple cycles in one stage. An integer multiply, for example, takes multiple cycles in the execute stage. When this occurs, subsequent instructions may stall.

  An instruction may also occupy more than one stage simultaneously, especially in the sense that a stage can be seen as a physical resource—for example, when instructions are dispatched they are assigned a place in the CQ at the same time they are passed to the execute stage. They can be said to occupy both the complete and execute stages in the same clock cycle.
- Stall—An occurrence when an instruction cannot proceed to the next stage.
- Superscalar—A superscalar processor is one that can issue multiple instructions concurrently from a conventional linear instruction stream. In a superscalar implementation, multiple instructions can be in the execute stage at the same time.
- Throughput—A measure of the number of instructions that are processed per cycle. For example, a series of double-precision floating-point multiply instructions has a throughput of one instruction per clock cycle.
- Write-back—In the context of instruction handling, write-back occurs when a result is written into the architectural registers (typically the GPRs, FPRs, and VRs). Results are written back at completion time. Results in the write-back buffer cannot be flushed. If an exception occurs, results from previous instructions must write back before the exception is taken.

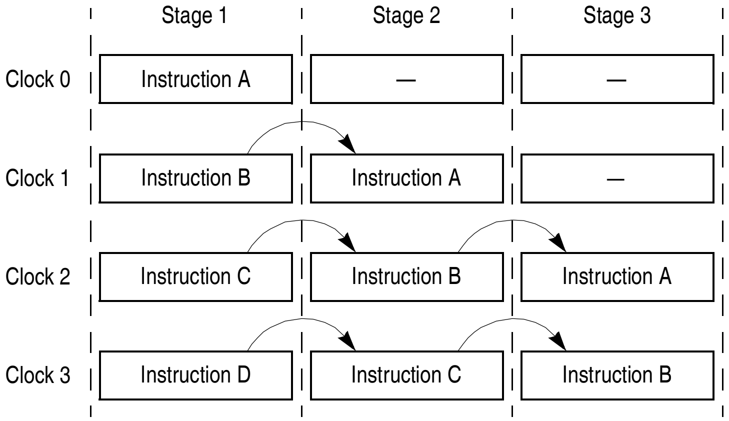
6.2 Instruction Timing Overview

The MPC7410 design minimizes average instruction execution latency, the number of clock cycles it takes to fetch, decode, dispatch, and execute instructions and make the results available for a subsequent instruction. Some instructions, such as loads and stores, access memory and require additional clock cycles between the execute phase and the write-back phase. These latencies vary depending on whether the access is to cacheable or noncacheable memory, whether it hits in the L1 or L2 cache, whether the cache access generates a write-back to memory, whether the access causes a snoop hit from another device that generates additional activity, and other conditions that affect memory accesses.

The MPC7410 implements many features to improve throughput, such as pipelining, superscalar instruction issue, branch folding, removal of fall-through branches, two-level speculative branch handling, and multiple execution units that operate independently and in parallel.

As an instruction passes from stage to stage in a pipelined system, the following instruction can follow through the stages as the former instruction vacates them, allowing several instructions to be processed simultaneously. While it may take several cycles for an instruction to pass through all the stages, when the pipeline has been filled, one instruction can complete its work on every clock cycle.

Figure 6-1 represents a generic pipelined execution unit.



	Stage 1	Stage 2	Stage 3
Clock 0	Instruction A	—	—
Clock 1	Instruction B	Instruction A	—
Clock 2	Instruction C	Instruction B	Instruction A
Clock 3	Instruction D	Instruction C	Instruction B

Figure 6-1. Pipelined Execution Unit

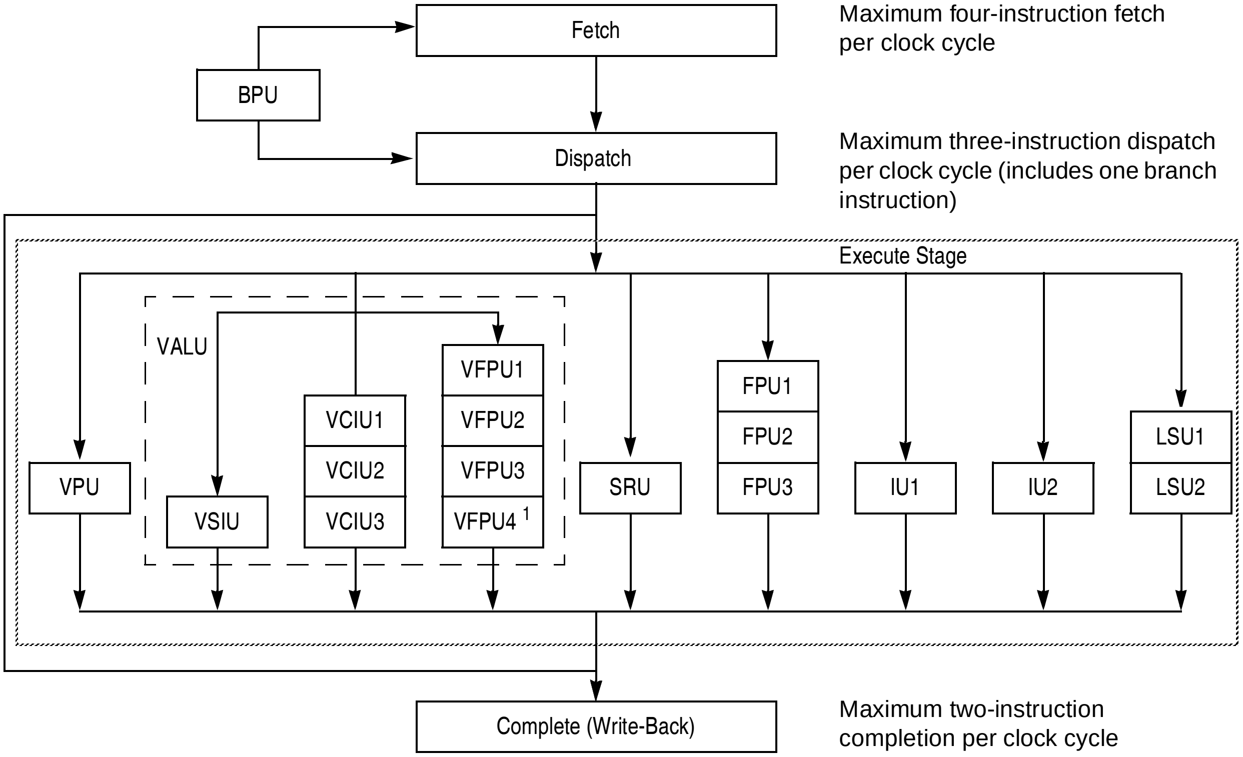
The entire path that instructions take through the fetch, decode/dispatch, execute, complete, and write-back stages is considered the MPC7410's master pipeline, and four of the MPC7410's execution units (FPU, LSU, VCIU, and VFPU) are also multiple-stage pipelines.

The MPC7410 contains the following execution units that operate independently and in parallel:

- Branch processing unit (BPU)
- Integer unit 1 (IU1)—executes all integer instructions
- Integer unit 2 (IU2)—executes all integer instructions except multiplies and divides
- 64-bit floating-point unit (FPU)
- Load/store unit (LSU)
- System register unit (SRU)
- AltiVec permute unit (VPU)
- AltiVec arithmetic logical unit (VALU), which contains the following three independent execution units for vector computations:
  - Vector simple integer unit (VSIU)
  - Vector complex integer unit (VCIU)
  - Vector floating-point unit (VFPU)

  One instruction can be dispatched to the VALU per clock cycle; however, the three vector arithmetic units are independent and can simultaneously execute separate instructions. Moreover, the VCIU and VFPU are pipelined, so they can operate on multiple instructions.

The MPC7410 can retire two instructions on every clock cycle. In general, the MPC7410 processes instructions in four stages—fetch, decode/dispatch, execute, and complete as shown in Figure 6-2. Note that the example of a pipelined execution unit in Figure 6-1 is similar to the three-stage FPU pipeline in Figure 6-2.


Fetch
Maximum four-instruction fetch per clock cycle
BPU
Dispatch
Maximum three-instruction dispatch per clock cycle (includes one branch instruction)
Execute Stage
VALU
VPU
VSIU
VCIU1
VCIU2
VCIU3
VFPU1
VFPU2
VFPU3
VFPU4 1
SRU
FPU1
FPU2
FPU3
IU1
IU2
LSU1
LSU2
Complete (Write-Back)
Maximum two-instruction completion per clock cycle

1 In non-Java mode, all VFPU instructions are pipelined as shown. In Java mode, all VFPU instructions need a fifth execution cycle; however, data forwarding for instruction depedency can still occur at the end of the fourth execution cycle as in non-Java mode.

Figure 6-2. Superscalar/Pipeline Diagram

The instruction pipeline stages are described as follows:

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

  Because there are so many variables, unless otherwise specified, the instruction timing examples below assume optimal performance and show the portion of the fetch stage in which the instruction is already in the instruction queue. The fetch stage ends when the instruction is dispatched.
- The decode/dispatch stage consists of the time it takes to fully decode the instruction and dispatch it from the instruction queue to the appropriate execution unit. Instruction dispatch requires the following:
  - Instructions can be dispatched only from the two lowest instruction queue entries, IQ0 and IQ1.
  - A maximum of two instructions can be dispatched per clock cycle (although an additional branch instruction can be handled by the BPU).

- — Only one instruction can be dispatched to each execution unit (IU1, IU2, FPU, LSU, SRU, VPU, and VALU) per clock cycle.
  - — There must be a vacancy in the specified execution unit.
  - — A rename register must be available for each destination operand specified by the instruction.
  - — For an instruction to dispatch, the appropriate execution unit must be available and there must be an open position in the CQ. If no entry is available, the instruction remains in the IQ.
- The execute stage consists of the time between dispatch to the execution unit (or reservation station) and the point at which the instruction vacates the execution unit.

  Most integer instructions have a one-cycle latency; results of these instructions can be used in the clock cycle after an instruction enters the execution unit. However, integer multiply and divide instructions take multiple clock cycles to complete. The IU1 can process all integer instructions; the IU2 can process all integer instructions except multiply and divide instructions.

  The LSU, FPU, VCIU and VFPU units are pipelined, as shown in Figure 6-2.

  Note the following regarding AltiVec instruction latency:

  - — In non-Java mode, all VFPU instructions are pipelined as shown in Figure 6-2. In Java mode, all VFPU instructions need an additional execution cycle before they can get to the completion stage; however, they can still forward their result to subsequent dependent instructions at the end of the fourth execution cycle as in non-Java mode
  - — All VSIU instructions have a one-cycle latency, except **mfvscr** and **mtvscr**, which may need additional execution cycles because of execution serialization.
- The complete (complete/write-back) pipeline stage maintains the correct architectural machine state and commits it to the architectural registers at the proper time. If the completion logic detects an instruction containing an exception status, all following instructions are canceled, their execution results in rename registers are discarded, and the correct instruction stream is fetched.

  The complete stage ends when the instruction is retired. Two instructions can be retired per cycle. Instructions are retired only from the two lowest CQ entries, CQ0 and CQ1.

The notation conventions used in the instruction timing examples are as follows:

□ Fetch—Although it is not shown in these figures, the fetch stage includes the time between when an instruction is requested and when it is dispatched from the instruction queue. The latency associated with accessing an instruction varies greatly, depending upon whether the instruction is in the BTIC, the on-chip cache, the L2 cache, or system memory (in which case latency can be affected by bus speed and traffic on the system bus, and address translation issues). Therefore, in the examples in this chapters, the fetch stage is usually idealized, that is, an instruction is usually shown to be in the fetch stage when it is a valid instruction in the instruction queue. The instruction queue has six entries, IQ0–IQ5.

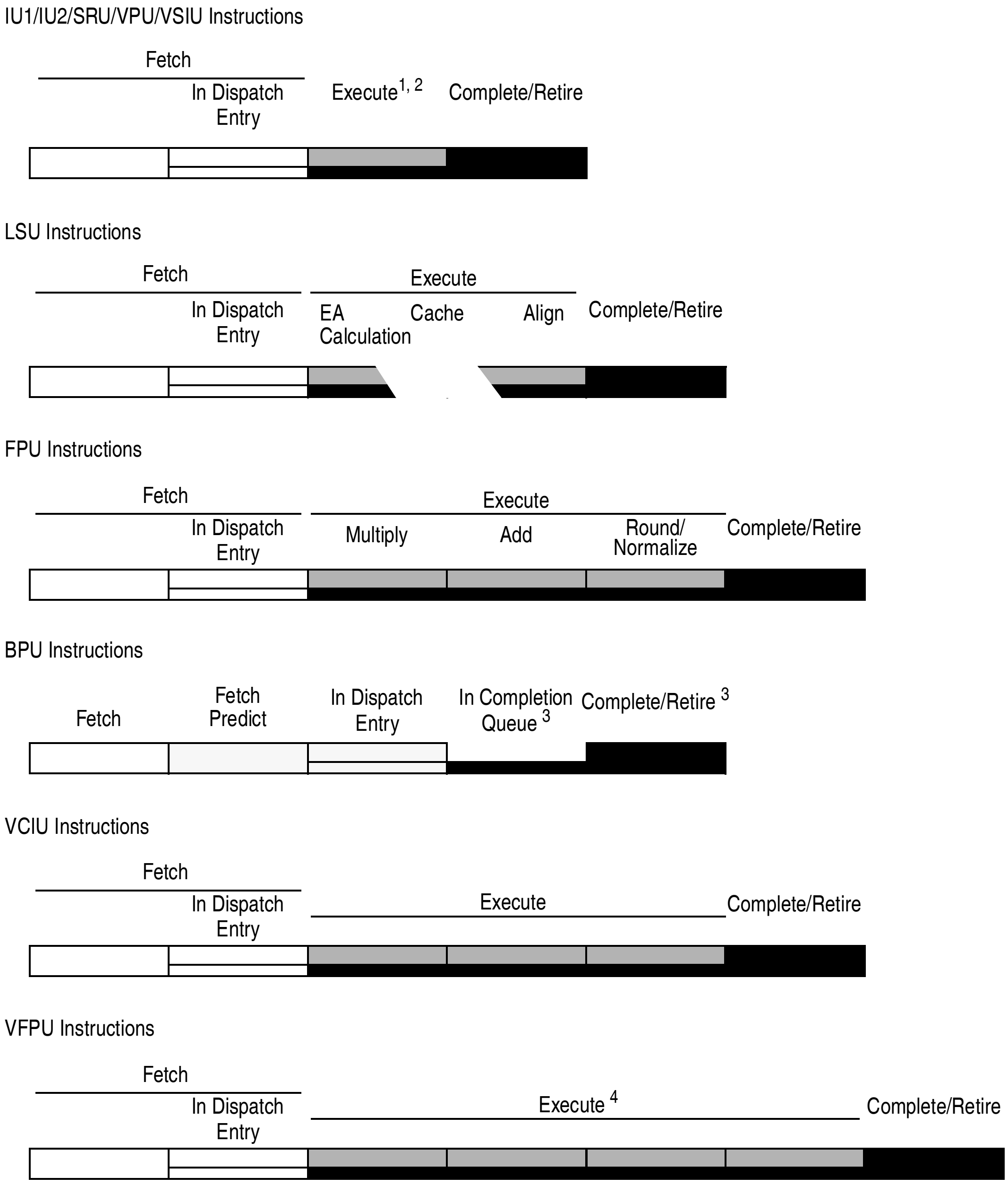
▭ In dispatch entry (IQ0/IQ1)—Instructions can be dispatched from IQ0 and IQ1. Because dispatch is instantaneous, it is perhaps more useful to describe it as an event that marks the point in time between the last cycle in the fetch stage and the first cycle in the execute stage.

▤ Execute—The operations specified by an instruction are being performed by the appropriate execution unit. The black stripe is a reminder that the instruction occupies an entry in the CQ, described in Figure 6-3.

■ Complete—The instruction is in the CQ. In the final stage, the results of the executed instruction are written back and the instruction is retired. The CQ has eight entries, CQ0–CQ7.

☰ In retirement entry—Completed instructions can be retired from CQ0 and CQ1. Like dispatch, retirement is an event that in this case occurs at the end of the final cycle of the complete stage.

Figure 6-3 shows the stages of MPC7410 execution units.



[1] Several integer instructions, such as multiply and divide instructions, require multiple cycles in the execute stage.

[2] **mtvscr** and **mfvscr** may need additional execution cycles because of execution serialization.

[3] Only those branch instructions that update the LR or CTR take an entry in the completion queue.

[4] In Java mode, VFPU instructions require an additional (fifth) execution cycle; however, data forwarding for instruction dependency can still occur at the end of the fourth execution cycle.

Figure 6-3. MPC7410 Microprocessor Pipeline Stages

6.3 Timing Considerations

The MPC7410 is a superscalar processor; as many as three instructions can be issued to the execution units (one branch instruction to the BPU, and two instructions issued from the IQ to the other execution units) during each clock cycle. Only one instruction can be dispatched to each execution unit.

Although instructions appear to the programmer to execute in program order, the MPC7410 improves performance by executing multiple instructions at a time, using hardware to manage dependencies. When an instruction is dispatched, the register file provides the source data to the execution unit. The register files and rename register have sufficient bandwidth to allow dispatch of two instructions per clock under most conditions.

The MPC7410's BPU decodes and executes branches immediately after they are fetched. When a conditional branch cannot be resolved due to a CR data dependency, the branch direction is predicted and execution continues from the predicted path. If the prediction is incorrect, the following steps are taken:

1. The instruction queue is purged and fetching continues from the correct path.
2. Any instructions ahead of the predicted branch in the CQ are allowed to complete.
3. Instructions after the mispredicted branch are purged.
4. Dispatching resumes from the correct path.

After an execution unit finishes executing an instruction, it places resulting data into the appropriate GPR, FPR, or VR rename register. The results are then stored into the correct GPR, FPR, or VR during the write-back stage. If a subsequent instruction needs the result as a source operand, it is made available simultaneously to the appropriate execution unit, which allows a data-dependent instruction to be decoded and dispatched without waiting to read the data from the register file. Branch instructions that update either the LR or CTR write back their results in a similar fashion.

The following section describes this process in greater detail.

6.3.1 General Instruction Flow

As many as four instructions can be fetched into the instruction queue (IQ) in a single clock cycle. Instructions are issued to the various execution units from the IQ. The MPC7410 tries to keep the IQ full at all times, unless instruction cache throttling is operating.

The number of instructions requested in a clock cycle is determined by the number of vacant spaces in the IQ during the previous clock cycle. This is shown in the examples in this chapter. Although the instruction queue can accept as many as four new instructions in a single clock cycle, if only one IQ entry is vacant, only one instruction is fetched. Typically instructions are fetched from the on-chip instruction cache, but they may also be fetched from the branch target instruction cache (BTIC). If the instruction request hits in the BTIC, it can usually present the first two instructions of the new instruction stream in the next clock cycle, giving enough time for the next pair of instructions to be fetched from the instruction cache with no idle cycles. If instructions are not in the BTIC or the on-chip instruction cache, they are fetched from the L2 cache or from system memory.

The MPC7410’s instruction cache throttling feature, managed through the instruction cache throttling control (ICTC) register, can lower the processor’s overall junction temperature by slowing the instruction fetch rate. See Chapter 10, “Power Management.”

Branch instructions are identified by the fetcher, and forwarded to the BPU directly, bypassing the IQ. If the branch is unconditional or if the specified conditions are already known, the branch can be resolved immediately. That is, the branch direction is known and instruction fetching can continue from the correct location. Otherwise, the branch direction must be predicted. The MPC7410 offers several resources to aid in quick resolution of branch instructions and for improving the accuracy of branch predictions. These include the following:

- Branch target instruction cache—The 64-entry (four-way-associative) branch target instruction cache (BTIC) holds branch target instructions so when a branch is encountered in a repeated loop, usually the first two instructions in the target stream can be fetched into the instruction queue on the next clock cycle. The BTIC can be disabled and invalidated through bits in HID0.
- Dynamic branch prediction—The 512-entry branch history table (BHT) is implemented with two bits per entry for four degrees of prediction—not taken, strongly not taken, taken, strongly taken. Whether a branch instruction is taken or not taken can change the strength of the next prediction. This dynamic branch prediction is not defined by the architecture.

  To reduce aliasing, only predicted branches update the BHT entries. Dynamic branch prediction is enabled by setting HID0[BHT]; otherwise, static branch prediction is used.
- Static branch prediction—Static branch prediction is defined by the architecture and involves encoding the branch instructions. See Section 6.4.1.3.1, “Static Branch Prediction.”

Branch instructions that do not update the LR or CTR are removed from the instruction stream either by branch folding or removal of fall-through branch instructions, as described in Section 6.4.1.1, “Branch Folding and Removal of Fall-Through Branch Instructions.” Branch instructions that update the LR or CTR are treated as if they require dispatch (even though they are not issued to an execution unit in the process). They are assigned a position in the CQ to ensure that the CTR and LR are updated sequentially.

All other instructions are issued from the IQ0 and IQ1. The dispatch rate depends upon the availability of resources such as the execution units, rename registers, and CQ entries, and upon the serializing behavior of some instructions. Instructions are dispatched in program order; an instruction in IQ1 cannot be dispatched ahead of one in IQ0.

Figure 6-4 shows the paths taken by instructions.

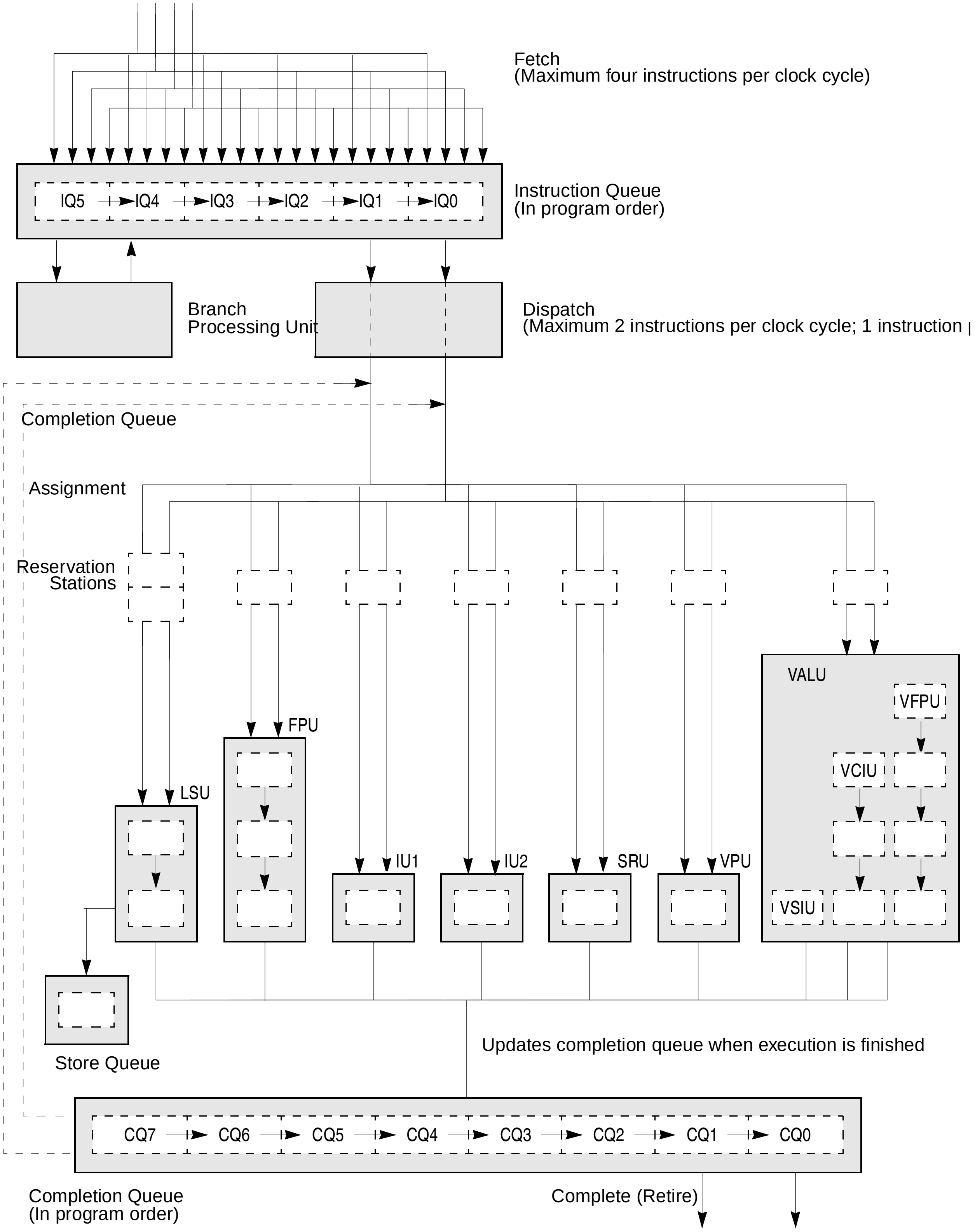


Figure 6-4. Instruction Flow Diagram

6.3.2 Instruction Fetch Timing

Instruction fetch latency depends on whether the fetch hits the BTIC, the on-chip instruction cache, or the L2 cache, if one is implemented. If no cache hit occurs, a memory transaction is required in which case fetch latency is affected by bus traffic, bus clock speed, and memory translation. These issues are discussed further in the following sections.

6.3.2.1 Cache Arbitration

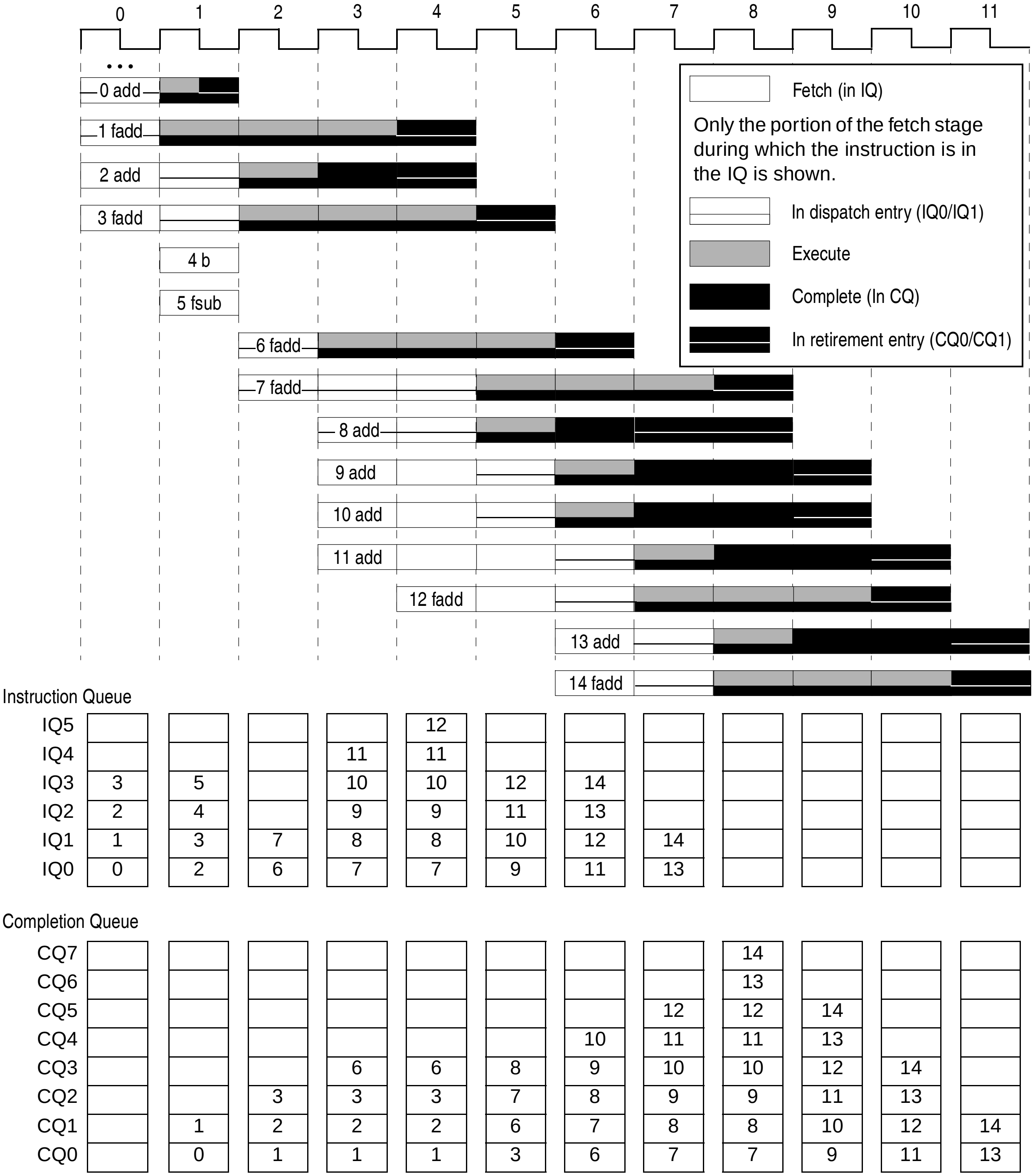
When the instruction fetcher requests instructions from the instruction cache, two things may happen. If the instruction cache is idle and the requested instructions are present, they are provided on the next clock cycle. However, if the instruction cache is busy due to a cache-line-reload operation, instructions cannot be fetched until that operation completes.

6.3.2.2 Cache Hit

If the instruction fetch hits the instruction cache, it takes only one clock cycle after the request for as many as four instructions to enter the instruction queue. Note that the cache is not blocked to internal accesses during a cache reload completes (hits under misses). The critical double word is written simultaneously to the cache and forwarded to the requesting unit, minimizing stalls due to load delays.

Figure 6-5 shows a simple example of instruction fetching that hits in the on-chip cache. This example uses a series of integer add and double-precision floating-point add instructions to show how the number of instructions to be fetched is determined, how program order is maintained by the IQ and CQ, how instructions are dispatched and retired in pairs (maximum), and how the FPU, IU1, and IU2 pipelines function. The following instruction sequence is examined:

```
0    add
1    fadd
2    add
3    fadd
4    br 6
5    fsub
6    fadd
7    fadd
8    add
9    add
10   add
11   add
12   fadd
13   add
14   fadd
15   .
16   .
17   .
```


0
1
2
3
4
5
6
7
8
9
10
11
• • •
0 add
1 fadd
2 add
3 fadd
4 b
5 fsub
6 fadd
7 fadd
8 add
9 add
10 add
11 add
12 fadd
13 add
14 fadd
Fetch (in IQ)
Only the portion of the fetch stage during which the instruction is in the IQ is shown.
In dispatch entry (IQ0/IQ1)
Execute
Complete (In CQ)
In retirement entry (CQ0/CQ1)

Instruction Queue

IQ5					12							
IQ4				11	11							
IQ3	3	5		10	10	12	14					
IQ2	2	4		9	9	11	13					
IQ1	1	3	7	8	8	10	12	14				
IQ0	0	2	6	7	7	9	11	13				

Completion Queue

CQ7									14			
CQ6									13			
CQ5								12	12	14		
CQ4							10	11	11	13		
CQ3				6	6	8	9	10	10	12	14	
CQ2			3	3	3	7	8	9	9	11	13	
CQ1		1	2	2	2	6	7	8	8	10	12	14
CQ0		0	1	1	1	3	6	7	7	9	11	13

Figure 6-5. Instruction Timing—Cache Hit

The instruction timing for this example is described cycle-by-cycle as follows:

0. In cycle 0, instructions 0–3 are fetched from the instruction cache. Instructions 0 and 1 are placed in the two entries in the instruction queue from which they can be dispatched on the next clock cycle.
1. In cycle 1, instructions 0 and 1 are dispatched to the IU2 and FPU, respectively. Notice that for instructions to be dispatched they must be assigned positions in the CQ. In this case, because the CQ is empty, instructions 0 and 1 take the two lowest entries in the CQ. This cycle also shows a special case for instruction 0. Because it can take a position in CQ0, this single-cycle integer instruction can execute and complete in the same cycle.

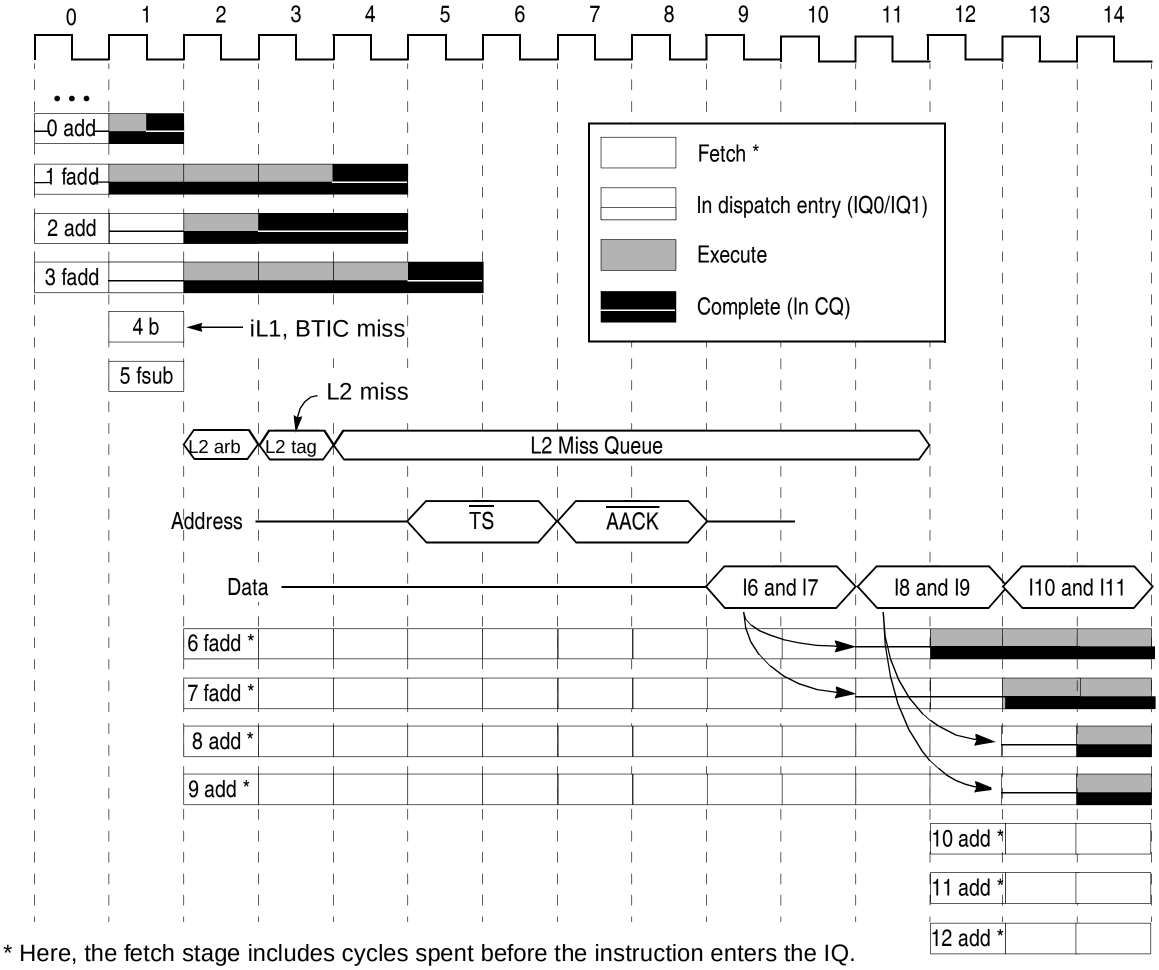
   Instructions 2 and 3 drop into the two dispatch positions in the instruction queue. Because there were two positions available in the instruction queue in clock cycle 0, two instructions (4 and 5) are fetched into the instruction queue. Instruction 4 is a branch unconditional instruction, which resolves immediately as taken. Because the branch is taken, it can therefore be folded from the instruction queue.
2. In cycle 2, assume a BTIC hit occurs and target instructions 6 and 7 are fetched into the instruction queue, replacing the folded **b** instruction (4) and instruction 5. Instruction 0 completes, writes back its results and vacates the CQ by the end of the clock cycle. Instruction 1 enters the second FPU execute stage, instruction 2 is dispatched to the IU2, and instruction 3 is dispatched into the first FPU execute stage. Because the taken branch instruction (4) does not update either CTR or LR, it does not require a position in the CQ and can be folded.
3. In cycle 3, target instructions (6 and 7) are fetched, replacing instructions 4 and 5 in IQ0 and IQ1. This replacement on taken branches is called branch folding. Instruction 1 proceeds through the last of the three FPU execute stages. Instruction 2 has executed but must remain in the CQ until instruction 1 completes. Instruction 3 replaces instruction 1 in the second stage of the FPU, and instruction 6 replaces instruction 3 in the first stage. Also, as will be shown in cycle 4, there is a single-cycle stall that occurs when the FPU pipeline is full.

   Because there were three vacancies in the instruction queue in the previous clock cycle, instructions 8–11 are fetched in this clock cycle.
4. Instruction 1 completes in cycle 4, allowing instruction 2 to complete. Instructions 3 and 6 continue through the FPU pipeline. Although instruction 7 is in IQ1, it cannot be dispatched because the FPU is busy, and because instruction 7 cannot be dispatched neither can instruction 8. The additional cycle stall allows the instruction queue to be completely filled. Because there was one opening in the instruction queue in clock cycle 3, one instruction is fetched (12) and the instruction queue is full.
5. In cycle 5, instruction 3 completes, allowing instruction 7 to be dispatched to the FPU, which in turn allows instruction 8 to be dispatched to the IU2. Instructions 9 and 10 drop to the dispatch positions in the instruction queue. No instructions are fetched in this clock cycle because there were no vacant IQ entries in clock cycle 4.
6. In cycle 6, instruction 6 completes, instruction 7 is in stage 2 of the FPU execute stage, and although instruction 8 has executed, it must wait for instruction 7 to complete. The two integer instructions, 9 and 10, are dispatched to the IU2 and IU1, respectively. Fetching resumes with instructions 13 and 14.

7. In cycle 7, instruction 7 is in the final FPU execute stage and instructions 8–10 wait in the CQ for instruction 7 to complete. Instructions 11 and 12 are dispatched to the IU2 and FPU, respectively.
8. In cycle 8, instructions 7–11 are through executing. Instructions 7 and 8 complete, write back, and vacate the CQ. Instruction 12 is in FPU stage 2 Instructions 13 and 14 are dispatched, filling the CQ.
9. In cycle 9, two more instructions (instructions 9 and 10) are retired from the CQ.

6.3.2.3 Cache Miss

Figure 6-6 shows an instruction fetch that misses both the on-chip cache and L2 cache. A processor/bus clock ratio of 2:1 is used. The same instruction sequence is used as in Section 6.3.2.2, “Cache Hit,” however in this example, the branch target instruction is not in either the L1 or L2 cache.



* Here, the fetch stage includes cycles spent before the instruction enters the IQ.

Instruction Queue

IQ5															
IQ4															
IQ3	3	5													
IQ2	2	4													
IQ1	1	3										7		9	
IQ0	0	2										6	7	8	

Completion Queue

CQ7															
CQ6															
CQ5															
CQ4															
CQ3															9
CQ2			3	3	3										8
CQ1		1	2	2	2									7	7
CQ0		0	1	1	1	3							6	6	6

Figure 6-6. Instruction Timing—Cache Miss

A cache miss extends the latency of the fetch stage, so in this example, the fetch stage shown represents not only the time the instruction spends in the IQ, but the time required for the instruction to be loaded from system memory, beginning in clock cycle 3.

By clock cycle 3, a memory access must occur because the target instruction for the **b** instruction is not in the BTIC (the target instruction is not in the L1 cache, so it cannot be in the BTIC), the instruction cache, or the L2 cache. During clock cycle 5, the address of the block of instructions is sent to the system bus. During clock cycle 9, two instructions (64 bits) are returned from memory on the first beat and are forwarded both to the cache and the instruction fetcher.

6.3.3 Memory Subsystem-Specific Pipeline Diagrams

Figure 6-7 shows the pipelining for a series of three loads.

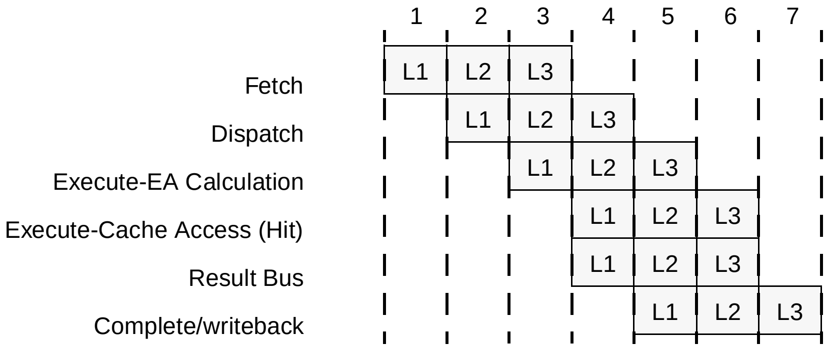

1 2 3 4 5 6 7
Fetch: L1 L2 L3
Dispatch: L1 L2 L3
Execute-EA Calculation: L1 L2 L3
Execute-Cache Access (Hit): L1 L2 L3
Result Bus: L1 L2 L3
Complete/writeback: L1 L2 L3

Figure 6-7. Data L1 Load Hit (No Stalls)

Figure 6-8 shows a series of three store operations that hit in the L1.

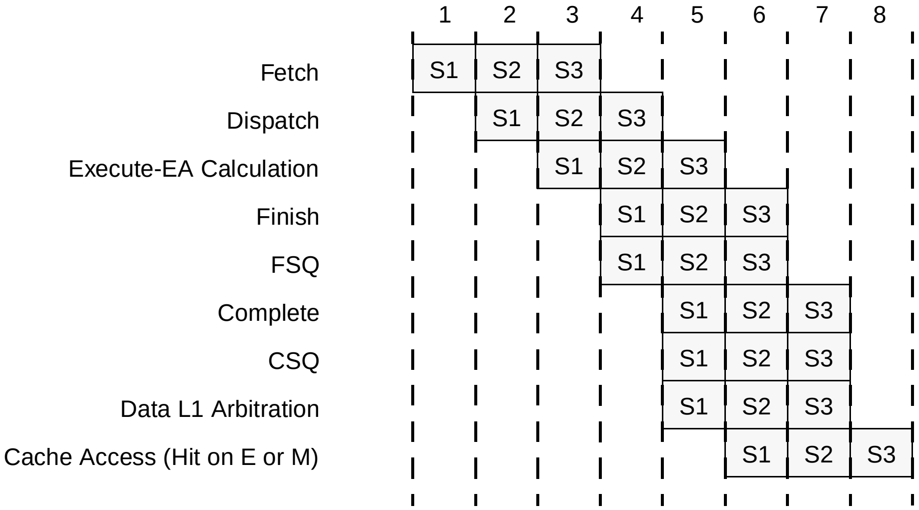


Figure 6-8. Data L1 Store Hit (No Stalls)

Figure 6-9 shows an L2 hit after an L1 miss. The L2 data queue queues operations that have accessed the L2 tags and are waiting to access the off-chip SRAMs. This example assumes an ideal case using a ÷1 clock and the fastest possible L2 response. This performance may not be available in an actual system, given SRAM timing constraints.

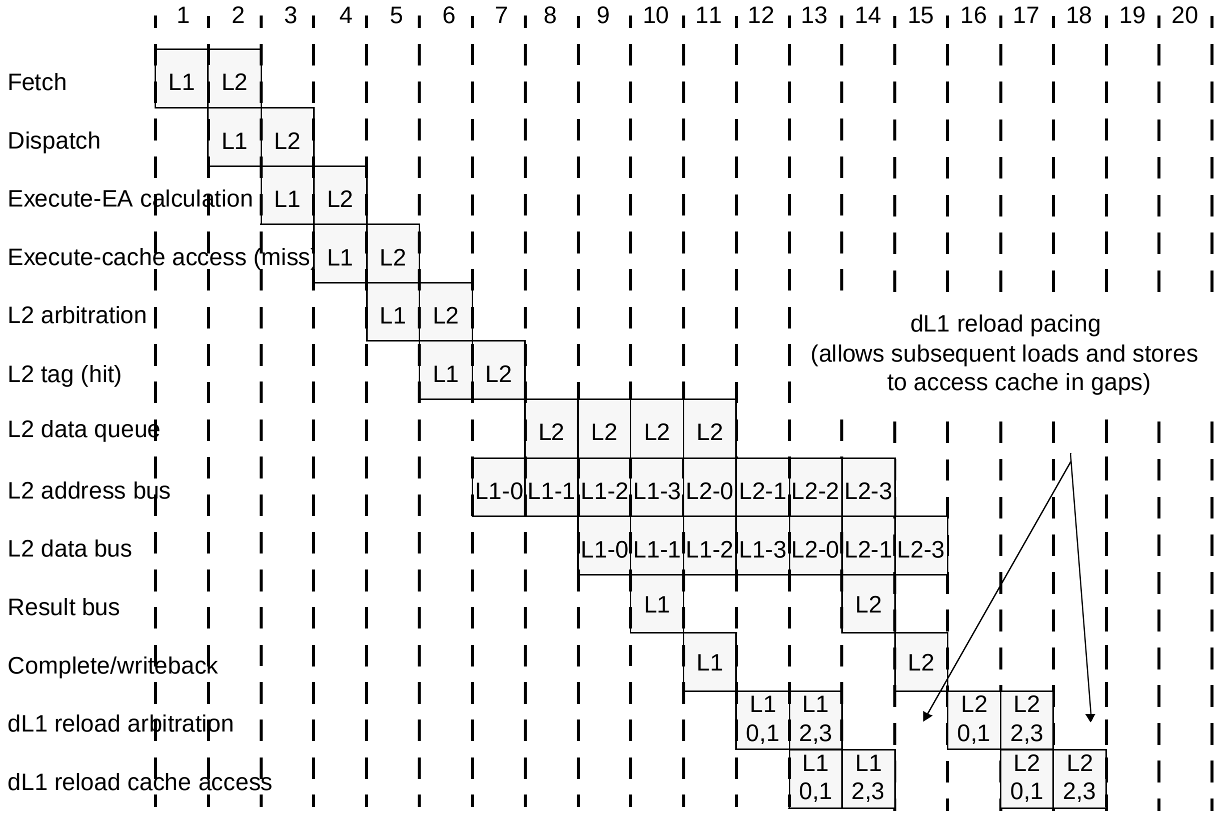


Figure 6-9. Data L1 Load Miss, L2 Hit (No Stalls)

Figure 6-10 shows a load that misses both the L1 and L2 caches. This example assumes an ideal case using a ÷2 clock and the fastest possible L2 response. This performance may not be available in an actual system, given controller and DRAM timing constraints. To illustrate the pipeline, this example shows 4-1 latency, which is unrealistic for 100-MHz SDRAM.

The L2 miss queue holds addresses that accessed the L2 tag and are waiting to access the system address bus (60x or MPX bus). The 60x/MPX bus data transaction queue queues information about system bus transactions that MPC7410 has performed on the system address bus or interventions and whose corresponding data transactions are pending.

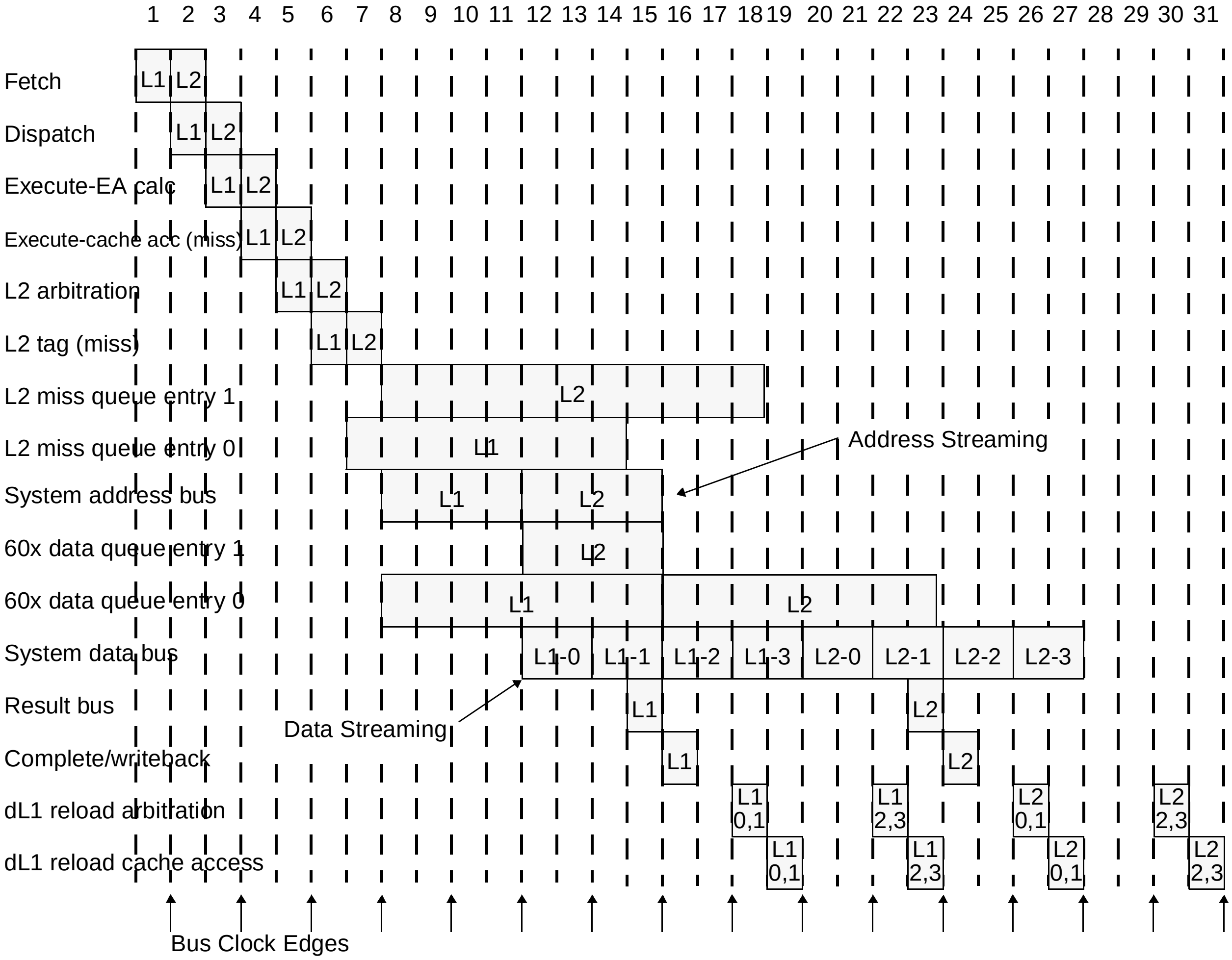


Figure 6-10. Data L1 Load Miss, L2 Miss, BIU Fetch

6.3.3.1 L2 Cache Access Timing Considerations (MPX Bus Only)

If an instruction fetch misses both the BTIC and the on-chip instruction cache, the MPC7410 next looks in the L2 cache. If the requested instructions are there, they are burst into the MPC7410 in much the same way as shown in Figure 6-6. The formula for the L2 cache latency for instruction and data accesses is as follows:

2 processor clock + 3 L2 clocks + 1 processor clock

Therefore, if the L2 is in 2:1 mode, the instruction fetch takes 8 processor clock cycles. Additional factors can also affect this latency, including the type of memory used to implement the L2 and whether the processor clock and L2 clocks are aligned immediately.

6.3.4 Instruction Dispatch and Completion Considerations

Several factors affect the MPC7410’s ability to dispatch instructions at a peak rate of two per cycle—the availability of the execution unit, destination rename registers, and CQ, as well as the handling of completion-serialized instructions. Several of these limiting factors are illustrated in the previous instruction timing examples.

To reduce dispatch unit stalls due to instruction data dependencies, the MPC7410 provides a single-entry reservation station for the FPU, SRU, VPU, VALU, and each IU, and a two-entry reservation station for the LSU. If a data dependency keeps an instruction from starting execution, that instruction is dispatched to the reservation station associated with its execution unit (and the rename registers are assigned), thereby freeing the positions in the instruction queue so instructions can be dispatched to other execution units. Execution begins during the same clock cycle that the rename buffer is updated with the data the instruction is dependent on.

If both instructions in IQ0 and IQ1 require the same execution unit, the instruction in IQ1 cannot be dispatched until the first instruction proceeds through the pipeline and provides the subsequent instruction with a vacancy in the requested execution unit.

The completion unit maintains program order after instructions are dispatched from the instruction queue, guaranteeing in-order completion and a precise exception model. Completing an instruction implies committing execution results to the architected destination registers. In-order completion ensures the correct architectural state when the MPC7410 must recover from a mispredicted branch or an exception.

Instruction state and all information required for completion is kept in the eight-entry, FIFO completion queue. A CQ entry is allocated for each instruction when it is dispatched to an execute unit; if no entry is available, the dispatch unit stalls. A maximum of two instructions per cycle may be completed and retired from the CQ, and the flow of instructions can stall when a longer-latency instruction reaches the last position in the CQ. Subsequent instructions cannot be completed and retired until that longer-latency instruction completes and retires. Examples of this are shown in Section 6.3.2.2, “Cache Hit,” and Section 6.3.2.3, “Cache Miss.”

The MPC7410 also allows an instruction to finish and complete in the same cycle. If an instruction is in CQ0 and it finishes, it completes in the same cycle. Likewise, if the instruction in CQ1 also finishes in the same cycle with the instruction in CQ0, both can also be simultaneously retired.

The MPC7410 can execute instructions out-of-order, but in-order completion by the completion unit ensures a precise exception mechanism. Program-related exceptions are signaled when the instruction causing the exception reaches the last position in the CQ. Prior instructions are allowed to complete before the exception is taken.

6.3.4.1 Rename Register Operation

To avoid contention for a given register file location in the course of out-of-order execution, the MPC7410 provides rename registers for holding instruction results before the completion commits them to the architected register. There are six GPR rename registers, six FPR rename registers, six VR rename registers, and one each for the CR, LR, and CTR.

When the dispatch unit dispatches an instruction to its execution unit, it allocates a rename register (or registers) for the results of that instruction. If an instruction is dispatched to a reservation station associated with an execution unit due to a data dependency, the dispatcher also provides a tag to the execution unit identifying the rename register that forwards the required data at completion. When the source data reaches the rename register, execution can begin.

Instruction results are transferred from the rename registers to the architected registers by the completion unit when an instruction is retired from the CQ without exceptions and after any predicted branch conditions preceding it in the CQ have been resolved correctly. If a branch prediction was incorrect, the instructions following the branch are flushed from the CQ, and any results of those instructions are flushed from the rename registers.

6.3.4.2 Instruction Serialization

Although the MPC7410 can dispatch and complete two instructions per cycle, so-called serializing instructions limit dispatch and completion to one instruction per cycle. There are five types of instruction serialization:

- Execution serialization—Execution serialized instructions are dispatched, held in the functional unit and do not execute until all prior instructions have completed. A functional unit holding an execution serialized instruction will not accept further instructions from the dispatcher. For example, execution serialization is used for instructions that modify non-renamed resources. Results from these instructions are generally not available or are forwarded to subsequent instructions until the instruction completes (using **mtspr** to write to LR or CTR provides forwarding to branch instructions).
- Store serialization (LSU only)—Store serialized instructions are dispatched, held in the LSU’s finished store queue, and are not committed for memory until all prior instructions have completed. While the store serialized instruction waits in the finished store queue, other load/store instructions can be freely executed. Store serialized instructions complete only from the bottom of the CQ. Thus, only one store-serialized instruction can complete per cycle, although non-serialized instructions can complete in the same cycle as a store serialized instruction. In general, all stores and cache operation instructions are store serialized.
- Sync serialization—Sync serialized instructions are dispatched and held in the LSU and are not performed until all prior instructions complete. Any load/store instructions dispatched behind the **sync** instruction remain in the reservation station until the **sync** serialized instruction completes. Because **sync**-serialized instructions complete only from the bottom of the CQ. Thus, only one **sync**-serialized instruction can complete in a given cycle. Non-serialized instructions can complete in the same cycle as a sync-serialized instruction.
- Completion serialization (post-dispatch or tail serialization)—Completion serialized instructions inhibit dispatching of subsequent instructions until the serialized instruction completes. Completion serialization is used for instructions that bypass the normal rename mechanism.
- Refetch serialization (flush serialization)—A subset of serialized instructions are also refetch serialized. Refetch serialized instructions inhibit dispatching of subsequent instructions and force refetching of subsequent instructions after completion.

6.4 Execution Unit Timings

The following sections describe instruction timing considerations within each of the respective execution units in the MPC7410.

6.4.1 Branch Processing Unit Execution Timing

Flow control operations (conditional branches, unconditional branches, and traps) are typically expensive to execute in most machines because they disrupt normal flow in the instruction stream. When a change in program flow occurs, the IQ must be reloaded with the target instruction stream. Previously issued instructions will continue to execute while the new instruction stream makes its way into the IQ, but depending on whether the target instruction is in the BTIC, instruction cache, L2 cache, or in system memory, some opportunities may be missed to execute instructions, as the example in Section 6.3.2.3, "Cache Miss," shows.

Performance features such as the branch folding, removal of fall-through branch instructions, BTIC, dynamic branch prediction (implemented in the BHT), two-level branch prediction, and the implementation of nonblocking caches minimize the penalties associated with flow control operations on the MPC7410. The timing for branch instruction execution is determined by many factors including the following:

- Whether the branch is taken
- Whether instructions in the target stream, typically the first two instructions in the target stream, are in the branch target instruction cache (BTIC)
- Whether the target instruction stream is in the on-chip cache
- Whether the branch is predicted
- Whether the prediction is correct

6.4.1.1 Branch Folding and Removal of Fall-Through Branch Instructions

When a branch instruction is encountered by the fetcher, the BPU immediately begins to decode it and tries to resolve it. All branch instructions except those that update either the LR or CTR are removed from the instruction flow before they would take a position in the CQ.

Branch folding occurs either when a branch is taken or is predicted as taken (as is the case with unconditional branches). When the BPU folds the branch instruction out of the instruction stream, the target instruction stream that is fetched into the instruction queue overwrites the branch instruction.

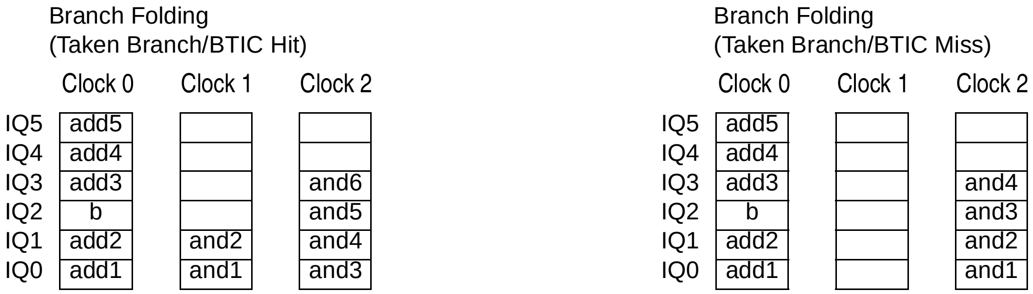
Figure 6-11 shows branch folding. Here a **b** instruction is encountered in a series of **add** instructions. The branch is resolved as taken. What happens on the next clock cycle depends on whether the target instruction stream (**add**) is in the BTIC, the instruction cache, or if it must be fetched from the L2 cache or from system memory.

Figure 6-11 shows cases where there is a BTIC hit, and when there is a BTIC miss (and instruction cache hit).

If there is a BTIC hit on the next clock cycle the **b** instruction is replaced by the target instruction, and1, that was found in the BTIC; the second and instruction is also fetched from the BTIC. On the next clock cycle, the next four and instructions from the target stream are fetched from the instruction cache.

If the target instruction is not in the BTIC, there is an idle cycle while the fetcher attempts to fetch the first four instructions from the instruction cache (on the next clock cycle). In the example in Figure 6-11, the first four target instructions are fetched on the next clock.

If it misses in the caches, an L2 cache or memory access is required, the latency of which is dependent on several factors, such as processor/bus clock ratios. In most cases, new instructions arrive in the IQ before the execution units become idle.



Branch Folding
(Taken Branch/BTIC Hit)

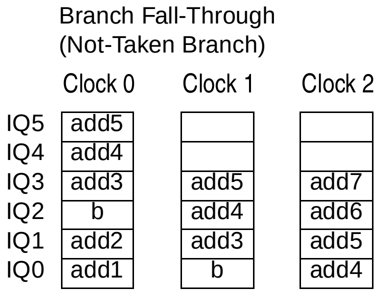
	Clock 0	Clock 1	Clock 2
IQ5	add5		
IQ4	add4		
IQ3	add3		and6
IQ2	b		and5
IQ1	add2	and2	and4
IQ0	add1	and1	and3

Branch Folding
(Taken Branch/BTIC Miss)

	Clock 0	Clock 1	Clock 2
IQ5	add5		
IQ4	add4		
IQ3	add3		and4
IQ2	b		and3
IQ1	add2		and2
IQ0	add1		and1

Figure 6-11. Branch Folding

Figure 6-12 shows the removal of fall-through branch instructions, which occurs when a branch is not taken or is predicted as not taken.



Branch Fall-Through
(Not-Taken Branch)

	Clock 0	Clock 1	Clock 2
IQ5	add5		
IQ4	add4		
IQ3	add3	add5	add7
IQ2	b	add4	add6
IQ1	add2	add3	add5
IQ0	add1	b	add4

Figure 6-12. Removal of Fall-Through Branch Instruction

In this case the branch instruction remains in the instruction queue and is removed from the instruction stream as if it were dispatched. However, it is not dispatched to an execution unit and is not assigned an entry in the CQ.

When a branch instruction is detected before it reaches a dispatch position, and if the branch is correctly predicted as taken, folding the branch instruction (and any instructions from the incorrect path) reduces the latency required for flow control to zero; instruction execution proceeds as though the branch was never there.

The advantage of removing the fall-through branch instructions at dispatch is only marginally less than that of branch folding. Because the branch is not taken, only the branch instruction needs to be discarded. The only cost of expelling the branch instruction from one of the dispatch entries rather than folding it is missing a chance to dispatch an executable instruction from that position.

6.4.1.2 Branch Instructions and Completion

As described in the previous section, instructions that do not update either the LR or CTR are removed from the instruction stream before they reach the CQ, either by branch folding (in the case of taken branches) or by removing fall-through branch instructions at dispatch (in the case of non-taken branches). However, branch instructions that update the architected LR and CTR must do so in program order and therefore must perform write-back in the completion stage, like the instructions that update the FPRs, GPRs, and VRs.

Branch instructions that update the CTR or LR pass through the instruction queue like no-branch instructions. At the point of dispatch, however, they are not sent to an execution unit, but rather are assigned a slot in the CQ, as shown in Figure 6-13.

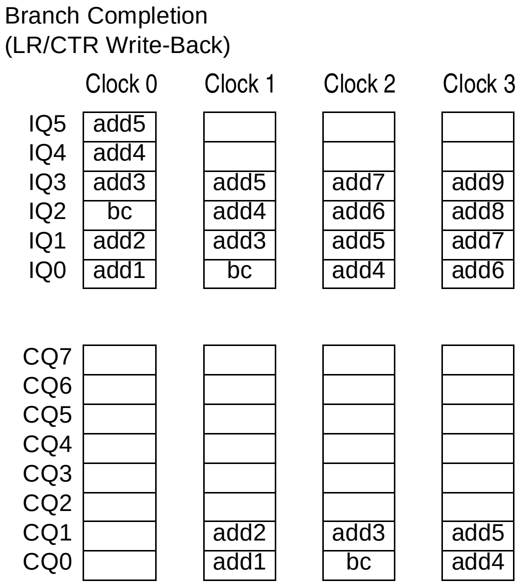


Figure 6-13. Branch Completion

In this example, the **bc** instruction is encoded to decrement the CTR. It is predicted as not-taken in clock cycle 0. In clock cycle 2, **bc** and add3 are both dispatched. In clock cycle 3, the architected CTR is updated and the **bc** instruction is retired from the CQ.

6.4.1.3 Branch Prediction and Resolution

The MPC7410 supports the following two types of branch prediction:

- Static branch prediction—This is defined by the architecture as part of the encoding of branch instructions.
- Dynamic branch prediction—This is a processor-specific mechanism implemented in hardware (in particular the branch history table, or BHT) that monitors branch instruction behavior and maintains a record from which the next occurrence of the branch instruction is predicted.

When a conditional branch cannot be resolved due to a CR data dependency, the BPU predicts whether it will be taken, and instruction fetching proceeds down the predicted path. If the branch prediction resolves as incorrect, the instruction queue and all subsequently executed instructions are purged, instructions executed prior to the predicted branch are allowed to complete, and instruction fetching resumes down the correct path.

The MPC7410 executes through two levels of prediction. Instructions from the first unresolved branch can execute, but they cannot complete until the branch is resolved. If a second branch instruction is encountered in the predicted instruction stream, it can be predicted and instructions can be fetched, but not executed, from the second branch. No action can be taken for a third branch instruction until at least one of the two previous branch instructions is resolved.

The number of instructions that can be executed after the issue of a predicted branch instruction is limited by the fact that no instruction executed after a predicted branch may actually update the register files or memory until the branch is completed. That is, instructions may be issued and executed, but cannot reach the write-back stage in the completion unit. When an instruction following a predicted branch completes execution, it does not write back its results to the architected registers, instead, it stalls in the CQ. Of course, when the CQ is full, no additional instructions can be dispatched, even if an execution unit is idle.

In the case of a misprediction, the MPC7410 can easily redirect its machine state because the programming model has not been updated. When a branch is mispredicted, all instructions that were dispatched after the predicted branch instruction are flushed from the CQ and any results are flushed from the rename registers.

The BTIC is a cache of recently used branch target instructions. If the search for the branch target hits in the cache, the first one or two branch instructions is available in the instruction queue on the next cycle (shown in Figure 6-5). Two instructions are fetched on a BTIC hit, unless the branch target is the last instruction in a cache block, in which case one instruction is fetched.

In some situations, an instruction sequence creates dependencies that keep a branch instruction from being resolved immediately, thereby delaying execution of the subsequent instruction stream based on the predicted outcome of the branch instruction. The instruction sequences and the resulting action of the branch instruction are described as follows:

- An **mtspr**(LR) followed by a **bclr**—Fetching stops and the branch waits for the **mtspr** to execute.
- An **mtspr**(CTR) followed by a **bcctr**—Fetching stops and the branch waits for the **mtspr** to execute.
- An **mtspr**(CTR) followed by a **bc** (CTR decrement)—Fetching stops and the branch waits for the **mtspr** to execute.
- A third **bc** (based-on-CR) is encountered while there are two unresolved **bc**(based-on-CR). The third **bc**(based-on-CR) is not executed and fetching stops until one of the previous **bc** (based-on-CR) is resolved. (Note that branch conditions can be a function of the CTR and the CR; if the CTR condition is sufficient to resolve the branch, then a CR-dependency is ignored.)

6.4.1.3.1 Static Branch Prediction

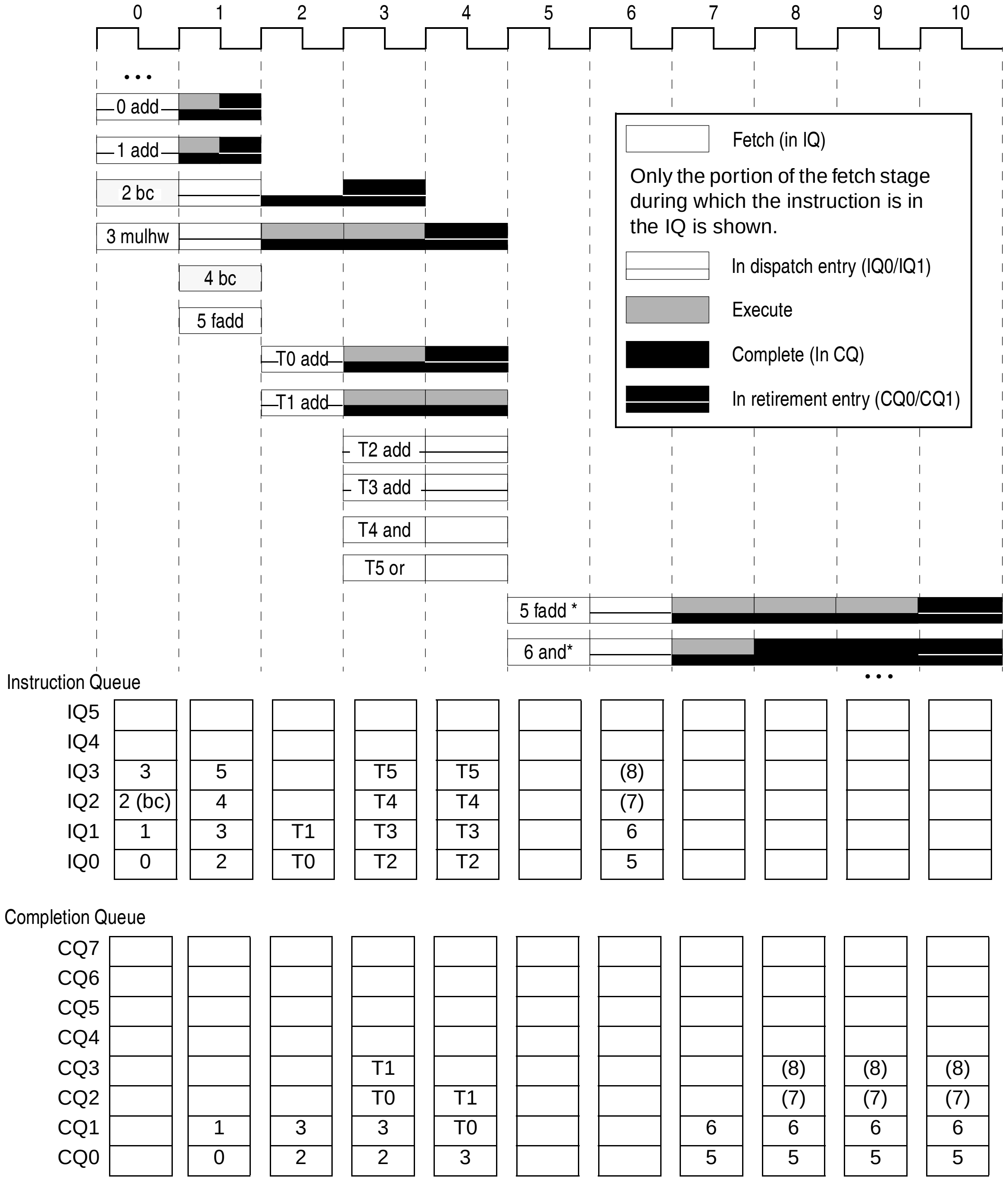
The architecture provides a field in branch instructions (the BO field) to allow software to hint whether a branch is likely to be taken. Rather than delaying instruction processing until the condition is known, the MPC7410 uses the instruction encoding to predict whether the branch is likely to be taken and begins fetching and executing along that path. When the branch condition is known, the prediction is evaluated. If the prediction was correct, program flow continues along that path; otherwise, the processor flushes any instructions and their results from the mispredicted path, and program flow resumes along the correct path.

Static branch prediction is used when HID0[BHT] is cleared. That is, the branch history table, which is used for dynamic branch prediction, is disabled. For information about static branch prediction, see “Conditional Branch Control,” in Chapter 4, “Addressing Modes and Instruction Set Summary,” in *The Programming Environments Manual*.

6.4.1.3.2 Predicted Branch Timing Examples

Figure 6-14 shows cases where branch instructions are predicted. It shows how both taken and not-taken branches are handled and how the MPC7410 handles both correct and incorrect predictions. The example shows the timing for the following instruction sequence:

```
0    add
1    add
2    bc
3    mulhw
4    bc T0
5    fadd
6    and
     add
T7   add
T8   add
T9   add
T10  add
T11  or
```


0
1
2
3
4
5
6
7
8
9
10
• • •
0 add
1 add
2 bc
3 mulhw
4 bc
5 fadd
T0 add
T1 add
T2 add
T3 add
T4 and
T5 or
5 fadd *
6 and*
• • •
Fetch (in IQ)
Only the portion of the fetch stage during which the instruction is in the IQ is shown.
In dispatch entry (IQ0/IQ1)
Execute
Complete (In CQ)
In retirement entry (CQ0/CQ1)

Instruction Queue

	0	1	2	3	4	5	6	7	8	9	10
IQ5											
IQ4											
IQ3	3	5		T5	T5		(8)				
IQ2	2 (bc)	4		T4	T4		(7)				
IQ1	1	3	T1	T3	T3		6				
IQ0	0	2	T0	T2	T2		5				

Completion Queue

	0	1	2	3	4	5	6	7	8	9	10
CQ7											
CQ6											
CQ5											
CQ4											
CQ3				T1					(8)	(8)	(8)
CQ2				T0	T1				(7)	(7)	(7)
CQ1		1	3	3	T0			6	6	6	6
CQ0		0	2	2	3			5	5	5	5

* Instructions 5 and 6 are not in the IQ in clock cycle 5. Here, the fetch stage shows cache latency.

Figure 6-14. Branch Instruction Timing

0. During clock cycle 0, instructions 0 and 1 are dispatched to their respective execution units. Instruction 2 is a branch instruction that updates the CTR. It is predicted as not taken in clock cycle 0. Instruction 3 is a **mulhw** instruction on which instruction 4 depends.

1. In clock cycle 1, instructions 0 and 1 execute and complete. Instructions 2 and 3 enter the dispatch entries in the IQ. Instruction 4 (a second **bc** instruction) and 5 are fetched. The second **bc** instruction is predicted as taken. It can be folded, but it cannot be resolved until instruction 3 writes back.
2. In clock cycle 2, instruction 4 has been folded and instruction 5 has been flushed from the IQ. The two target instructions, T0 and T1, are both in the BTIC, so they are fetched in this cycle. Note that even though the first **bc** instruction may not have resolved by this point (we can assume it has), the MPC7410 allows fetching from a second predicted branch stream. However, these instructions could not be dispatched until the previous branch has resolved.
3. In clock cycle 3, target instructions T2–T5 are fetched as T0 and T1 are dispatched.
4. In clock cycle 4, instruction 3, on which the second branch instruction depended, writes back and the branch prediction is proven incorrect. Even though T0 is in CQ1, from which it could be written back, it is not written back because the branch prediction was incorrect. All target instructions are flushed from their positions in the pipeline at the end of this clock cycle, as are any results in the rename registers.

After one clock cycle required to refetch the original instruction stream, instruction 5, the same instruction that was fetched in clock cycle 1, is brought back into the IQ from the instruction cache, along with three others (not all of which are shown).

6.4.2 Integer Unit Execution Timing

The MPC7410 has two integer units. The IU1 can execute all integer instructions; and the IU2 can execute all integer instructions except multiply and divide instructions. As shown in Figure 6-2, each integer unit has one execute pipeline stage, thus when a multicycle integer instruction is being executed, no other integer instructions can begin to execute. Table 6-6 lists integer instruction latencies.

Most integer instructions have an execution latency of one clock cycle.

6.4.3 Floating-Point Unit Execution Timing

The floating-point unit on the MPC7410 executes all floating-point instructions. Execution of most floating-point instructions is pipelined within the FPU, allowing up to three instructions to be executing in the FPU concurrently. Although most floating-point instructions execute with three-cycle latency and one-cycle throughput, three instructions (**fdivs**, **fdiv**, and **fres**) execute with latencies of 17 to 31 cycles. The **fdivs**, **fdiv**, **fres**, **mcrfs**, **mtfsb0**, **mtfsb1**, **mtfsfi**, **mffs**, and **mtfsf** instructions block the floating-point unit pipeline until they complete execution, and thereby inhibit the dispatch of additional floating-point instructions. See Table 6-7 for floating-point instruction execution timing.

6.4.4 Effect of Floating-Point Exceptions on Performance

For the highest and most predictable floating-point performance, all exceptions should be disabled in the FPSCR and MSR and FPSCR[NI] should be set.

If any exceptions are enabled (through a combination of MSR[FE] and one or more of the FPSCR enable bits), the MPC7410 FPU takes one addition cycle to complete instructions. This does not affect latency for data dependency. It may however, degrade performance by consuming limited CQ resources for 1 extra cycle per instruction.

6.4.5 Load/Store Unit Execution Timing

In addition to executing the PowerPC load and store instructions, the LSU also executes the AltiVec LRU and transient instructions. The execution of most load and store instructions is pipelined. The LSU has two pipeline stages. The first is for effective address calculation and MMU translation and the second is for accessing data in the cache. Load and store instructions have a two-cycle latency and one-cycle throughput.

If operands are misaligned, additional latency may be required either for an alignment exception to be taken or for additional bus accesses. Load instructions that miss in the cache block subsequent cache accesses during the cache line refill. Table 6-8 gives load and store instruction execution latencies.

6.4.5.1 Effect of Operand Placement on Performance

The VEA states that the placement (location and alignment) of operands in memory may affect the relative performance of memory accesses, and in some cases affect it significantly. The effects memory operand placement has on performance are shown in Table 6-1.

The best performance is guaranteed if memory operands are aligned on natural boundaries. For the best performance across the widest range of implementations, the programmer should assume the performance model described in Chapter 3, “Operand Conventions,” in *The Programming Environments Manual*.

The effect of misalignment on memory access latency is the same for big- and little-endian addressing modes except for multiple and string operations that cause an alignment exception in little-endian mode.

In Table 6-1, optimal means that one effective address (EA) calculation occurs during the memory operation. Good means that multiple EA calculations occur during the operation, which may cause additional bus activities with multiple bus transfers. Poor means that an alignment exception is generated.

Table 6-1. Performance Effects of Memory Operand Placement

Operand		Boundary Crossing [1]			
Size	**Byte Alignment**	**None**	**8 Byte**	**Cache Line**	**Protection Boundary**
		Integer			
4 Byte	4 <4	Optimal Optimal	— Good	— Good	— Good
2 Byte	2 <2	Optimal Optimal	— Good	— Good	— Good
1 Byte	1	Optimal	—	—	—
lmw, **stmw** [2]	4 <4	Good Poor	Good Poor	Good Poor	Good Poor
String [2]		Good	Good	Good	Good
		Floating-Point			

Table 6-1. Performance Effects of Memory Operand Placement (continued)

Operand		Boundary Crossing [1]			
8 Byte	8 4 <4	Optimal — —	— Good Poor	— Good Poor	— Good Poor
4 Byte	4 <4	Optimal Poor	— Poor	— Poor	— Poor

[1] Vector operands are not shown because they are always aligned.
optimal: One EA calculation occurs.
good: Multiple EA calculations occur which may cause additional bus activities with multiple bus transfers.
poor: Alignment exception occurs.

[2] These operations are not supported in little-endian mode, and would cause an alignment exception.

Note that the MPC7410 differs from the MPC750 in some aspects of little-endian operation; in little-endian mode, MPC7410 does not work with the MPC106.

6.4.5.2 Integer Store Gathering

The MPC7410 performs store gathering for write-through operations to nonguarded space. It performs cache-inhibited stores to nonguarded space for 4-byte, word-aligned stores. These stores are combined in the LSU to form a double word sent out on the 60x bus as a single-beat operation. However, stores are gathered only if the successive stores meet the criteria and are queued and pending. Store gathering occurs regardless of the address order of the stores. Store gathering is enabled by setting HID0[SGE]. Stores can be gathered in big-endian modes.

Store gathering is not done for the following:

- Stores to guarded cache-inhibited or write-through space
- Byte-reverse store operations
- **stwcx.** instructions
- **ecowx** instructions
- A store that occurs during a table search operation
- Little-endian store operations
- Floating-point store operations

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

6.4.6 System Register Unit Execution Timing

Most instructions executed by the SRU either directly access renamed registers or either access or modify nonrenamed registers. Instructions generally execute in strict order. Results from these instructions are not available to subsequent instructions until the instruction completes and is retired. See Section 6.3.4.2, “Instruction Serialization,” for more information on serializing instructions executed by the SRU. Table 6-4 and Table 6-5 show SRU instruction execution timings.

6.4.7 AltiVec Instructions Executed by the LSU

The LSU execute the AltiVec LRU and transient instructions.

6.4.7.1 LRU Instructions

The AltiVec architecture specifies that the **lvxl** and **stvxl** instructions differ from other AltiVec load and store instructions in that they leave cache entries in a least-recently-used (LRU) state instead of a most-recently-used state. This is used to identify data that is known to have little reuse and poor caching characteristics.

On the MPC7410, these instructions follow the cache allocation and replacement policies described in Chapter 3, "L1 and L2 Cache Operation," but they leave their addressed cache entries in the LRU state. In addition, all LRU instructions are also interpreted to be transient and are also treated as described in the next section. Additional discussion on LRU effects may be found in Chapter 3, "L1 and L2 Cache Operation."

6.4.7.2 Transient Instructions

The AltiVec architecture describes a difference between static and transient memory accesses.

A static memory access should have some reasonable degree of locality and 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 MPC7410 supports both static and transient memory access behavior.

If a memory access is designated as to transient, that cache block is marked not to be cast out to the L2 unless it has been modified in the L1 data cache. If it is modified in the L1, the block is not allocated in the L2 cache when it is victimized from the L1 data cache. Instead, the block is written directly to main memory, bypassing the L2 cache.

The following instructions are interpreted to be transient:

- **dstt** and **dststt** (transient forms of the two data stream touch instructions)
- **lvxl** and **stvxl**

6.4.8 AltiVec Instructions

The MPC7410 implements all instructions in the AltiVec specification. The AltiVec instruction set has no optional instructions; however, a few instructions associated with the load/store model are defined to allow significant differences between implementations. The following sections describe the MPC7410's implementation of these options.

6.4.8.1 AltiVec Permute Unit (VPU) Execution Timing

All AltiVec permute instructions are executed in a single cycle

6.4.8.2 AltiVec Arithmetic Logical Unit (VALU) Execution Timing

The AltiVec arithmetic logical unit (VALU) contains the following three independent execution units for vector computations:

- Vector simple integer unit (VSIU)
- Vector complex integer unit (VCIU)
- Vector floating-point unit (VFPU)

Execution timing for these units are described in the following sections.

6.4.8.2.1 Vector Simple Integer Unit (VSIU) Execution Timing

Except **mtvscr** and **mfvscr**, the VSIU executes all AltiVec simple integer instructions and all AltiVec floating-point compare, minimum, and maximum instructions, all of which have single-cycle latency.

6.4.8.2.2 Vector Complex Integer Unit (VCIU) Execution Timing

The VCIU executes all AltiVec complex integer instructions, which have a three-cycle latency.

6.4.8.2.3 Vector Floating-Point Unit (VFPU) Execution Timing

In non-Java mode, all AltiVec floating-point instructions (except for the floating-point compare, minimum, and maximum instructions, which are executed in the VSIU) have a four-cycle latency.

In Java mode, they have a five-cycle latency. However, similar to non-Java mode, data forwarding for instructions with dependencies can occur at the end of the fourth execution cycle as shown in the following examples.

Consider the data dependency in the following two-instruction sequence:

```
0  vaddfp V0,V1,V2
1  vaddfp V3,V0,V4
```

Figure 6-15 shows the instruction timing for the sequence in non-Java mode. Note that instruction 1 is dispatched in clock cycle 2, but remains in the reservation station until clock cycle 5 when the source operand, **v**0, is available from instruction 0. At this point, instruction 1 enters the first execute stage.

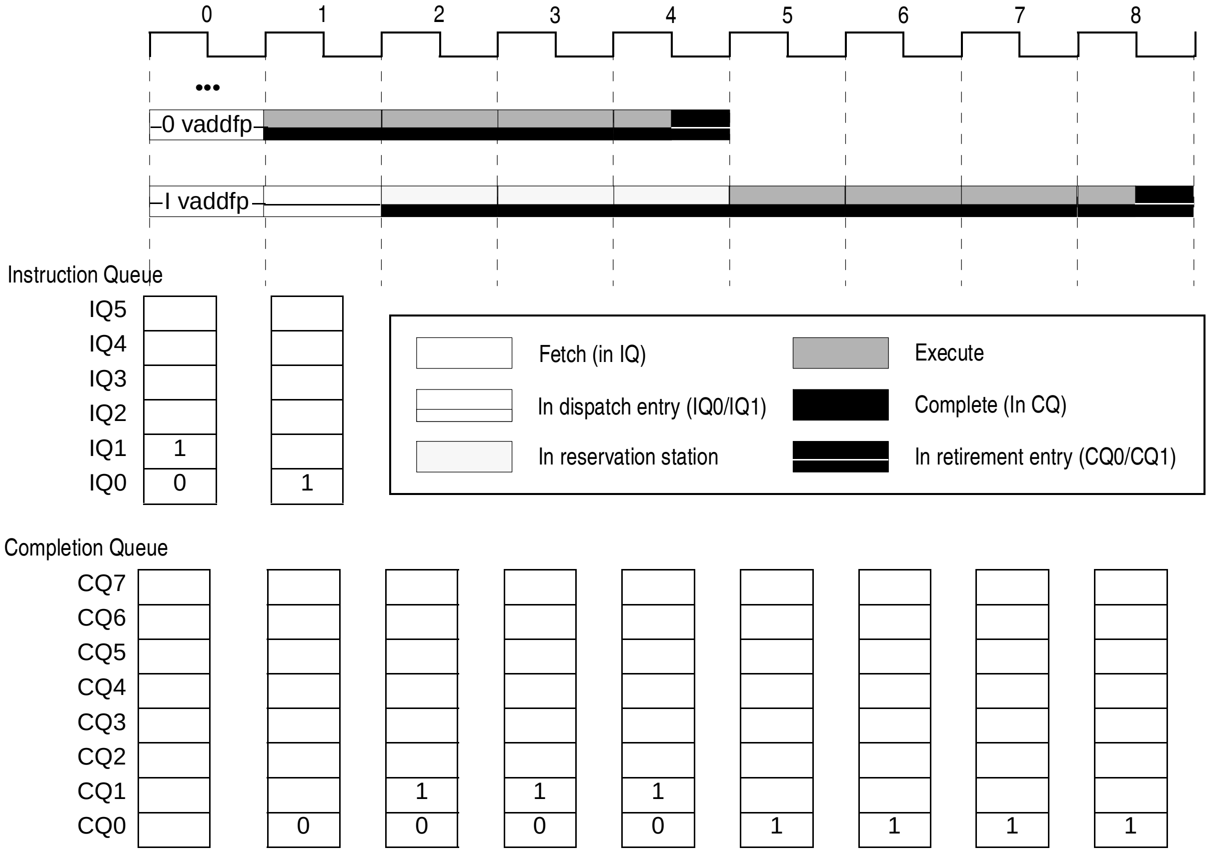


Figure 6-15. Data Dependencies in Non-Java Mode

Figure 6-16 shows that even though the execution pipeline is five stages deep in Java mode, data forwarding can still occur at the end of the fourth execution stage, just as in the non-Java mode example in Figure 6-15.

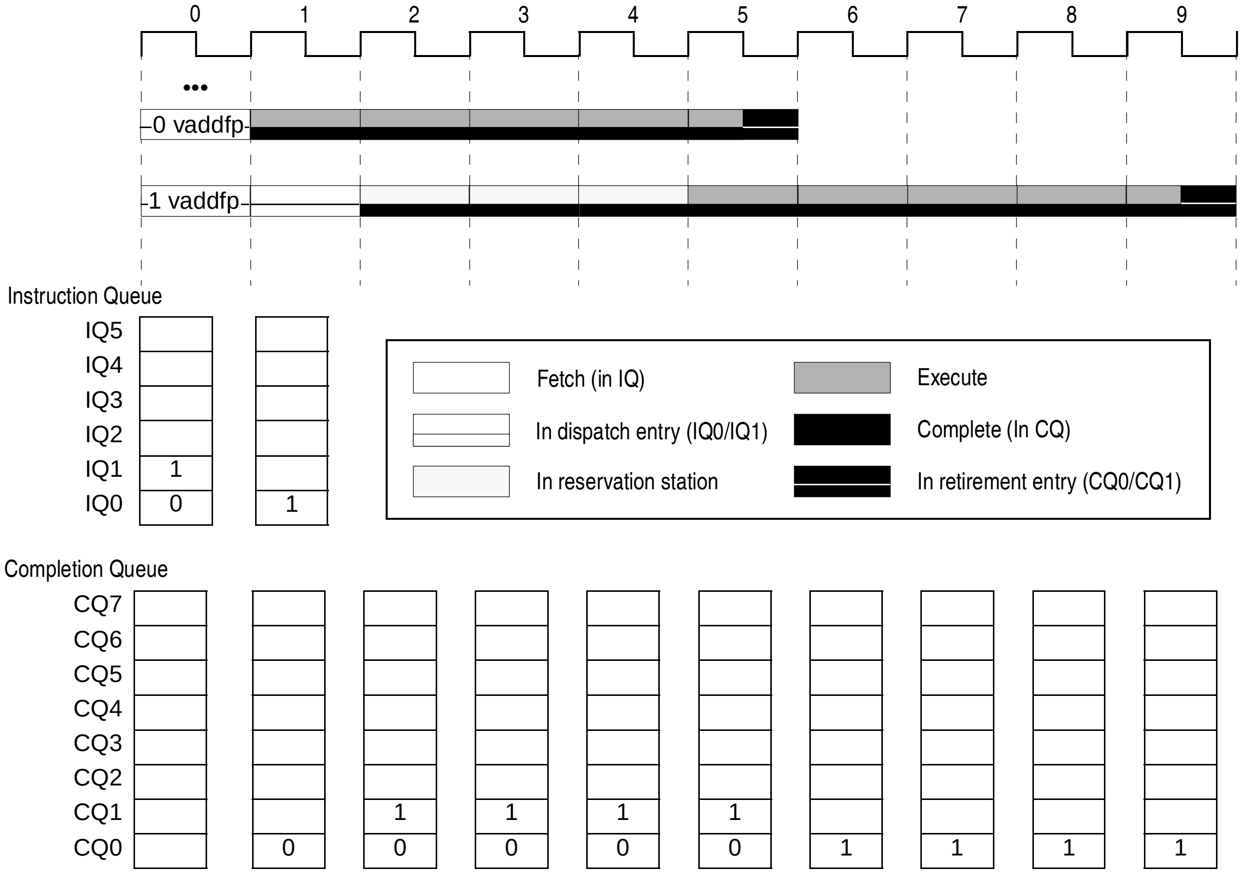


Figure 6-16. Data Forwarding in Java Mode

6.5 Memory Performance Considerations

Because the MPC7410 can have a maximum instruction throughput of three instructions per clock cycle, lack of memory bandwidth can affect performance. For the MPC7410 to maximize performance, it must be able to read and write data efficiently. If a system has multiple bus devices, one of them may experience long memory latencies while another bus master (for example, a direct-memory access controller) is using the external bus.

6.5.1 Caching and Memory Coherency

To minimize the effect of bus contention, the architecture defines WIM bits that are used to configure memory regions as caching-enforced or caching-inhibited. Accesses to such memory locations never update the on-chip cache. If a cache-inhibited access hits the on-chip cache, the cache block is invalidated. If the cache block is marked modified, it is copied back to memory before being invalidated. Where caching is permitted, memory is configured as either write-back or write-through, which are described as follows:

- Write-back—Configuring a memory region as write-back lets a processor modify data in the cache without updating system memory. For such locations, memory updates occur only on modified cache block replacements, cache flushes, or when one processor needs data that is modified in another's cache. Therefore, configuring memory as write-back can help when bus traffic could cause bottlenecks, especially for multiprocessor systems and for regions in which data, such as local variables, is used often and is coupled closely to a processor.

  If multiple devices use data in a memory region marked write-through, snooping must be enabled to allow the copyback and cache invalidation operations necessary to ensure cache coherency. The MPC7410's snooping hardware keeps other devices from accessing invalid data. For example, when snooping is enabled, the MPC7410 monitors transactions of other bus devices. For example, if another device needs data that is modified on the MPC7410's cache, the access is delayed so the MPC7410 can copy the modified data to memory.
- Write-through—Store operations to memory marked write-through always update both system memory and the on-chip cache on cache hits. Because valid cache contents always match system memory marked write-through, cache hits from other devices do not cause modified data to be copied back as they do for locations marked write-back. However, all write operations are passed to the bus, which can limit performance. Load operations that miss the on-chip cache must wait for the external store operation.

  Write-through configuration is useful when cached data must agree with external memory (for example, video memory), when shared (global) data may be needed often, or when it is undesirable to allocate a cache block on a cache miss.

Chapter 3, "L1 and L2 Cache Operation," describes the caches, memory configuration, and snooping in detail.

6.5.2 Effect of TLB Miss on Performance

TLB misses causes a hardware table search for the PTE tables and the TLB to be loaded. Table 6-2 shows some estimated latencies. These latencies are a sum of the latencies for the table search, TLB reload, and a reaccess of the TLB.

Table 6-2. Effect of TLB Miss on Performance

Cache Hit/Miss	Latency
100% L1 cache hit	9 cycles
100% L1 cache miss with 100% L2 cache hit with L2 core running at 1:1	15 cycles
100% L1 cache miss with 100% L2 cache hit with L2 core running at 1.5:1	17 cycles
100% L1 cache miss with 100% L2 cache hit with L2 core running at 2:1	18 cycles
100% L1 & L2 cache miss with bus running at 2.5:1 with 6:3:3:3 memory	28 cycles
100% L1 & L2 cache miss with bus running at 4:1 with 5:2:2:2 memory	33 cycles
100% L1 & L2 cache miss with bus running at 4:1 with 11:1:1:1 memory	57 cycles

The PTE table search assumes a hit in the first entry of the primary PTEG and no RC updates.

6.6 Instruction Scheduling Guidelines

The performance of the MPC7410 can be improved by avoiding resource conflicts and scheduling instructions to take fullest advantage of the parallel execution units. Instruction scheduling on the MPC7410 can be improved by observing the following guidelines:

- To reduce mispredictions, separate the instruction that sets CR bits from the branch instruction that evaluates them. Because there can be no more than 12 instructions in the processor (with the instruction that sets CR in CQ0 and the dependent branch instruction in IQ5), there is no advantage to having more than 10 instructions between them.
- Likewise, when branching to a location specified by the CTR or LR, separate the **mtspr** instruction that initializes the CTR or LR from the dependent branch instruction. This ensures the register values are immediately available to the branch instruction.
- Schedule instructions such that two can be dispatched at a time.
- Schedule instructions to minimize stalls due to busy execution units.
- Avoid scheduling high-latency instructions close together. Interspersing single-cycle latency instructions between longer-latency instructions minimizes the effect that instructions such as integer divide and multiply can have on throughput.
- Avoid using serializing instructions.
- Schedule instructions to avoid dispatch stalls:
  - Eight instructions can be tracked in the CQ; therefore, eight instructions can be in the execute stages at any one time
  - There are six GPR rename registers; therefore only six GPRs can be specified as destination operands at any time. If no rename registers are available, instructions cannot enter the execute stage and remain in the reservation station or instruction queue until they become available.

    Note that load with update address instructions use two destination registers
  - Similarly, there are six FPR rename registers and six VR rename registers, so only six FPR and six VR destination operands can be in the execute and complete stages at any time.

6.6.1 Branch, Dispatch, and Completion Unit Resource Requirements

This section describes the specific resources required to avoid stalls during branch resolution, instruction dispatching, and instruction completion.

6.6.1.1 Branch Resolution Resource Requirements

The following is a list of branch instructions and the resources required to avoid stalling the fetch unit in the course of branch resolution:

- The **bclr** instruction requires LR availability.
- The **bcctr** instruction requires CTR availability.
- Branch and link instructions require shadow LR availability.

- The “branch conditional on counter decrement and the CR” condition requires CTR availability or the CR condition must be false, and the MPC7410 cannot execute instructions after an unresolved predicted branch when the BPU encounters a branch.
- A branch conditional on CR condition cannot be executed following an unresolved predicted branch instruction.

6.6.1.2 Dispatch Unit Resource Requirements

The following is a list of resources required to avoid stalls in the dispatch unit. IQ[0] and IQ[1] are the two dispatch entries in the instruction queue:

- Requirements for dispatching from IQ[0] are as follows:
  - Needed execution unit available
  - Needed GPR rename registers available
  - Needed FPR rename registers available
  - Needed VR rename registers available
  - CQ is not full.
  - A completion-serialized instruction is not being executed.
- Requirements for dispatching from IQ[1] are as follows:
  - Instruction in IQ[0] must dispatch.
  - Instruction dispatched by IQ[0] is not completion- or refetch-serialized.
  - Needed execution unit is available (after dispatch from IQ[0]).
  - Needed GPR rename registers are available (after dispatch from IQ[0]).
  - Needed FPR rename register is available (after dispatch from IQ[0]).
  - Needed VR rename registers available (after dispatch from IQ[0]).
  - CQ is not full (after dispatch from IQ[0]).

6.6.1.3 Completion Unit Resource Requirements

The following is a list of resources required to avoid stalls in the completion unit; note that the two completion entries are described as CQ[0] and CQ[1], where CQ[0] is the CQ located at the end of the CQ (see Figure 6-4).

- Requirements for completing an instruction from CQ[0] are as follows:
  - Instruction in CQ[0] must be finished.
  - Instruction in CQ[0] must not follow an unresolved predicted branch.
  - Instruction in CQ[0] must not cause an exception.
- Requirements for completing an instruction from CQ[1] are as follows:
  - Instruction in CQ[0] must complete in same cycle.
  - Instruction in CQ[1] must be finished.
  - Instruction in CQ[1] must not follow an unresolved predicted branch.
  - Instruction in CQ[1] must not cause an exception.

- — Instruction in CQ[1] must be an integer, load, **dcbt**, data streaming, or AltiVec instruction.
- — Number of CR updates from both CQ[0] and CQ[1] must not exceed two.
- — Number of GPR updates from both CQ[0] and CQ[1] must not exceed two.
- — Number of FPR updates from both CQ[0] and CQ[1] must not exceed two.
- — Number of VR updates from both CQ[0] and CQ[1] must not exceed two.

6.7 Instruction Latency Summary

Instruction timing in number of processor clock cycles is shown in Table 6-3 through Table 6-9. The latency tables use the following conventions:

- Pipelined load /store instructions are shown with cycles of total latency and throughput cycles separated by a colon.
- The variable ‘b’ represents the processor/system-bus clock ratio.
- ‘Broadcast’ indicates a bus broadcast that has a minimum value of 3*b.
- Pipelined floating-point instructions are shown with number of clocks in each pipeline stage separated by dashes.
- In addition, additional cycles due to serializations are indicated in the cycles column with the following:
  - — c (completion serialization)
  - — s (store serialization)
  - — y (sync serialization)
  - — e (execution serialization)
  - — r (refetch serialization)

Table 6-3 through Table 6-9 list latencies associated with instructions executed by each execution unit. Table 6-3 describes branch instruction latencies.

Table 6-3. Branch Operation Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]
b[**l**][**a**]	18	—	I	BPU	1
bc[**l**][**a**]	16	—	B	BPU	1
bcctr[**l**]	19	528	XL	BPU	1
bclr[**l**]	19	016	XL	BPU	1

[1] Taken branches may be folded for an effective cycle time of 0.

Table 6-4 lists system register instruction latencies.

Table 6-4. SRU Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
isync	19	150	XL	SRU	2 {c,r}
mfmsr	31	083	X	SRU	1
mfspr (DBATs)	31	339	XFX	SRU	3 {e}
mfspr (IBATs)	31	339	XFX	SRU	3
mfspr (not BATs)	31	339	XFX	SRU	1 {e}
mfsr	31	595	X	SRU	3
mfsrin	31	659	X	SRU	3 {e}
mftb	31	371	X	SRU	1
mtmsr	31	146	X	SRU	1 {e}
mtspr (DBATs)	31	467	XFX	SRU	2 {e}
mtspr (IBATs)	31	467	XFX	SRU	2 {e}
mtspr (not BATs)	31	467	XFX	SRU	2 {e}
mtsr	31	210	X	SRU	2 {e}
mtsrin	31	242	X	SRU	3 {e}
mttb	31	467	XFX	SRU	1 {e}
rfi	19	050	XL	SRU	2 {c,r}
sc	17	- -1	SC	SRU	2 {c,r}

Table 6-5 lists condition register logical instruction latencies.

Table 6-5. Condition Register Logical Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
mcrf	19	000	XL	SRU	1 {e}
crand	19	257	XL	SRU	1 {e}
crandc	19	129	XL	SRU	1 {e}
creqv	19	289	XL	SRU	1 {e}
crnand	19	225	XL	SRU	1 {e}
crnor	19	033	XL	SRU	1 {e}
cror	19	449	XL	SRU	1 {e}
crorc	19	417	XL	SRU	1 {e}
crxor	19	193	XL	SRU	1 {e}
mcrxr	31	512	X	SRU	1 {e}

Table 6-5. Condition Register Logical Execution Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles
mfcr	31	019	X	SRU	1 {e}
mtcrf	31	144	XFX	SRU	1 {e}

Table 6-6 shows integer instruction latencies. Note that the IU1 executes all integer arithmetic instructions—multiply, divide, shift, rotate, add, subtract, and compare. The IU2 executes all integer instructions except multiply and divide (that is, shift, rotate, add, subtract, and compare).

Table 6-6. Integer Unit Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
addc[**o**][.]	31	010	XO	IU	1
adde[**o**][.]	31	138	XO	IU	1 {e}
addi	14	—	D	IU	1
addic	12	—	D	IU	1
addic.	13	—	D	IU	1
addis	15	—	D	IU	1
addme[**o**][.]	31	234	XO	IU	1 {e}
addze[**o**][.]	31	202	XO	IU	1 {e}
add[**o**][.]	31	266	XO	IU	1
andc[.]	31	060	X	IU	1
andi.	28	—	D	IU	1
andis.	29	—	D	IU	1
and[.]	31	028	X	IU	1
cmp	31	000	X	IU	1
cmpi	11	—	D	IU	1
cmpl	31	032	X	IU	1
cmpli	10	—	D	IU	1
cntlzw[.]	31	026	X	IU	1
divwu[**o**][.]	31	459	XO	IU	19
divw[**o**][.]	31	491	XO	IU	19
eqv[.]	31	284	X	IU	1
extsb[.]	31	954	X	IU	1
extsh[.]	31	922	X	IU	1
mulhwu[.]	31	011	XO	IU	2,3,4,5,6[1]
mulhw[.]	31	075	XO	IU	2,3,4,5[1]

Table 6-6. Integer Unit Execution Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles
mulli	07	—	D	IU	2,3[1]
mull[**o**][.]	31	235	XO	IU	2,3,4,5[1]
nand[.]	31	476	X	IU	1
neg[**o**][.]	31	104	XO	IU	1
nor[.]	31	124	X	IU	1
orc[.]	31	412	X	IU	1
ori	24	—	D	IU	1
oris	25	—	D	IU	1
or[.]	31	444	X	IU	1
rlwimi[.]	20	—	M	IU	1
rlwinm[.]	21	—	M	IU	1
rlwnm[.]	23	—	M	IU	1
slw[.]	31	024	X	IU	1
srawi[.]	31	824	X	IU	1
sraw[.]	31	792	X	IU	1
srw[.]	31	536	X	IU	1
subfc[**o**][.]	31	008	XO	IU	1
subfe[**o**][.]	31	136	XO	IU	1 {e}
subfic	08	—	D	IU	1
subfme[**o**][.]	31	232	XO	IU	1 {e}
subfze[**o**][.]	31	200	XO	IU	1 {e}
subf[.]	31	040	XO	IU	1
tw	31	004	X	IU	2
twi	03	—	D	IU	2
xori	26	—	D	IU	1
xoris	27	—	D	IU	1
xor[.]	31	316	X	IU	1

[1] The number of cycles depends on the operands: The instruction takes two cycles if one of the operands is zero. The instruction takes three cycles if only 8 bits are being multiplied—that is, the high order bits are either all zeros or all ones (for negative operands). The instuction takes four cycles if only 16 bits are being multiplied. The instuction takes five cycles if only 24 bits are being multiplied. The instuction takes six cycles if all 32 bits are being multiplied.

Table 6-7 shows latencies for floating-point instructions. Floating-point instructions with a single entry in the cycles column are not pipelined. Thus, the unit executing these nonpipelined instructions is busy for the full duration of the instruction execution and is not available for additional instruction execution.

Pipelined floating-point instructions are shown with number of clocks in each pipeline stage separated by dashes.

Table 6-7. Floating-Point Unit Execution Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles
fabs[.]	63	264	X	FPU	1-1-1
fadds[.]	59	021	A	FPU	1-1-1
fadd[.]	63	021	A	FPU	1-1-1
fcmpo	63	032	X	FPU	1-1-1
fcmpu	63	000	X	FPU	1-1-1
fctiwz[.]	63	015	X	FPU	1-1-1
fctiw[.]	63	014	X	FPU	1-1-1
fdivs[.]	59	018	A	FPU	17
fdiv[.]	63	018	A	FPU	31
fmadds[.]	59	029	A	FPU	1-1-1
fmadd[.]	63	029	A	FPU	1-1-1
fmr[.]	63	072	X	FPU	1-1-1
fmsubs[.]	59	028	A	FPU	1-1-1
fmsub[.]	63	028	A	FPU	1-1-1
fmuls[.]	59	025	A	FPU	1-1-1
fmul[.]	63	025	A	FPU	1-1-1
fnabs[.]	63	136	X	FPU	1-1-1
fneg[.]	63	040	X	FPU	1-1-1
fnmadds[.]	59	031	A	FPU	1-1-1
fnmadd[.]	63	031	A	FPU	1-1-1
fnmsubs[.]	59	030	A	FPU	1-1-1
fnmsub[.]	63	030	A	FPU	1-1-1
fres[.]	59	024	A	FPU	10
frsp[.]	63	012	X	FPU	1-1-1
frsqrte[.]	63	026	A	FPU	1-1-1
fsel[.]	63	023	A	FPU	1-1-1
fsubs[.]	59	020	A	FPU	1-1-1
fsub[.]	63	020	A	FPU	1-1-1

Table 6-7. Floating-Point Unit Execution Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles
mcrfs	63	064	X	FPU	3 {e}
mffs[.]	63	583	X	FPU	3 {e}
mtfsb0[.]	63	070	X	FPU	3{e}
mtfsb1[.]	63	038	X	FPU	3{e}
mtfsfi[.]	63	134	X	FPU	3{e}
mtfsf[.]	63	711	XFL	FPU	3 {e}

Table 6-8 shows load and store instruction latencies. Load/store multiple and string instruction cycles are represented as a fixed number of cycles plus a variable number of cycles, where *n* = the number of words accessed by the instruction. Pipelined load/store instructions are shown with cycles of total latency and throughput cycles separated by a colon.

Table 6-8. Load/Store Instruction Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Updates	Speculatively Executed
dcba	31	758	X	LSU	2:3* {s}	R, C	No
dcbf	31	086	X	LSU	2:3*b {s}	R	No
dcbi	31	470	X	LSU	2:3*b {s}	R, C	No
dcbst	31	054	X	LSU	2:3*b {s}	R	No
dcbt	31	278	X	LSU	2:1	R	Yes
dcbtst	31	246	X	LSU	2:1	R	Yes
dcbz	31	1014	X	LSU	2:3* {s}	R, C	No
eciwx	31	310	X	LSU	2:1	R	Yes
ecowx	31	438	X	LSU	2:1 {s}	R, C	No
eieio	31	854	X	LSU	2:3*b {y}	None	No
icbi	31	982	X	LSU	2:3*b {s}	R	No
lbz	34	—	D	LSU	2:1	R	Yes
lbzu	35	—	D	LSU	2:1	R	Yes
lbzux	31	119	X	LSU	2:1	R	Yes
lbzx	31	087	X	LSU	2:1	R	Yes
lfd	50	—	D	LSU	2:1	R	Yes
lfdu	51	—	D	LSU	2:1	R	Yes
lfdux	31	631	X	LSU	2:1	R	Yes
lfdx	31	599	X	LSU	2:1	R	Yes
lfs	48	—	D	LSU	2:1	R	Yes

Table 6-8. Load/Store Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Updates	Speculatively Executed
lfsu	49	—	D	LSU	2:1	R	Yes
lfsux	31	567	X	LSU	2:1	R	Yes
lfsx	31	535	X	LSU	2:1	R	Yes
lha	42	—	D	LSU	2:1	R	Yes
lhau	43	—	D	LSU	2:1	R	Yes
lhaux	31	375	X	LSU	2:1	R	Yes
lhax	31	343	X	LSU	2:1	R	Yes
lhbrx	31	790	X	LSU	2:1	R	Yes
lhz	40	—	D	LSU	2:1	R	Yes
lhzu	41	—	D	LSU	2:1	R	Yes
lhzux	31	311	X	LSU	2:1	R	Yes
lhzx	31	279	X	LSU	2:1	R	Yes
lmw	46	—	D	LSU	2 + n {c,e}	R	No
lswi	31	597	X	LSU	2 + n {c,e}	R	No
lswx	31	533	X	LSU	2 + n {c,e}	R	No
lwarx	31	020	X	LSU	3:3 {e}	R	No
lwbrx	31	534	X	LSU	2:1	R	Yes
lwz	32	—	D	LSU	2:1	R	Yes
lwzu	33	—	D	LSU	2:1	R	Yes
lwzux	31	055	X	LSU	2:1	R	Yes
lwzx	31	023	X	LSU	2:1	R	Yes
stb	38	—	D	LSU	2:1 {s}	R, C	No
stbu	39	—	D	LSU	2:1 {s}	R, C	No
stbux	31	247	X	LSU	2:1 {s}	R, C	No
stbx	31	215	X	LSU	2:1 {s}	R, C	No
stfd	54	—	D	LSU	2:1	R, C	No
stfdu	55	—	D	LSU	2:1	R, C	No
stfdux	31	759	X	LSU	2:1 {s}	R, C	No
stfdx	31	727	X	LSU	2:1 {s}	R, C	No
stfiwx	31	983	X	LSU	2:1 {s}	R, C	No
stfs	52	—	D	LSU	2:1	R, C	No
stfsu	53	—	D	LSU	2:1	R, C	No
stfsux	31	695	X	LSU	2:1 {s}	R, C	No

Table 6-8. Load/Store Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Updates	Speculatively Executed
stfsx	31	663	X	LSU	2:1 {s}	R, C	No
sth	44	—	D	LSU	2:1 {s}	R, C	No
sthbrx	31	918	X	LSU	2:1 {s}	R, C	No
sthu	45	—	D	LSU	2:1 {s}	R, C	No
sthux	31	439	X	LSU	2:1 {s}	R, C	No
sthx	31	407	X	LSU	2:1 {s}	R, C	No
stmw	47	—	D	LSU	2 + n {e}	R, C	No
stswi	31	725	X	LSU	2 + n {e}	R, C	No
stswx	31	661	X	LSU	2 + n {e}	R, C	No
stw	36	—	D	LSU	2:1 {s}	R, C	No
stwbrx	31	662	X	LSU	2:1 {s}	R, C	No
stwcx.	31	150	X	LSU	5:5 {s}	R, C	No
stwu	37	—	D	LSU	2:1 {s}	R, C	No
stwux	31	183	X	LSU	2:1 {s}	R, C	No
stwx	31	151	X	LSU	2:1 {s}	R, C	No
sync	31	598	X	LSU	8+broadcast {y}	None	No
tlbie	31	306	X	LSU	2:3*b {s}	None	No
tlbsync	31	566	X	LSU	8+broadcast {y}	None	No

[1] For cache-ops, the first number indicates the latency in finishing a single instruction, and the second number denotes the throughput for back to back cache-ops. The throughput cycle may be larger than the initial latency due to the fact that more cycles may be needed to complete the instruction to the cache which remains busy preventing subsequent cache-ops from executing. These numbers also assume that there is a bus broadcast (i.e. M = 1). For M = 0, the number will be a minimum of 3 cycles.

Table 6-9 describes AltiVec instruction latencies.

Table 6-9. AltiVec Instruction Latencies

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Update
dss	31	—	X	LSU	2:1	—
dssall	31	—	X	LSU	2:1	—
dst	31	—	X	LSU	2:2 [2]	R
dstst	31	—	X	LSU	2:2 [2]	R
dststt	31	—	X	LSU	2:2 [2]	R
dstt	31	—	X	LSU	2:2 [2]	R
lvebx	31	—	X	LSU	2:1	R

Table 6-9. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Update
lvehx	31	—	X	LSU	2:1	R
lvewx	31	—	X	LSU	2:1	R
lvsl	31	—	X	LSU	2:1	—
lvsr	31	—	X	LSU	2:1	—
lvx	31	—	X	LSU	2:1	R
lvxl	31	—	X	LSU	2:1	R
mfvscr	04	—	VX	VALU(VSIU)	1{e}	—
mtvscr	04	—	VX	VALU(VSIU)	1 {e}	—
stvebx	31	—	X	LSU	2:1	R, C
stvehx	31	—	X	LSU	2:1	R, C
stvewx	31	—	X	LSU	2:1	R, C
stvx	31	—	X	LSU	2:1	R, C
stvxl	31	—	X	LSU	2:1	R, C
vaddcuw	04	—	VX	VALU(VSIU)	1	—
vaddfp	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vaddsbs	04	—	VX	VALU(VSIU)	1	—
vaddshs	04	—	VX	VALU(VSIU)	1	—
vaddsws	04	—	VX	VALU(VSIU)	1	—
vaddubm	04	—	VX	VALU(VSIU)	1	—
vaddubs	04	—	VX	VALU(VSIU)	1	—
vadduhm	04	—	VX	VALU(VSIU)	1	—
vadduhs	04	—	VX	VALU(VSIU)	1	—
vadduwm	04	—	VX	VALU(VSIU)	1	—
vadduws	04	—	VX	VALU(VSIU)	1	—
vand	04	—	VX	VALU(VSIU)	1	—
vandc	04	—	VX	VALU(VSIU)	1	—
vavgsb	04	—	VX	VALU(VSIU)	1	—
vavgsh	04	—	VX	VALU(VSIU)	1	—
vavgsw	04	—	VX	VALU(VSIU)	1	—
vavgub	04	—	VX	VALU(VSIU)	1	—
vavguh	04	—	VX	VALU(VSIU)	1	—
vavguw	04	—	VX	VALU(VSIU)	1	—

Table 6-9. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Update
vcfsx	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vcfux	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vcmpbfp	04	—	VX	VALU(VSIU)	1	—
vcmpeqfp	04	—	VX	VALU(VSIU)	1	—
vcmpequb	04	—	VX	VALU(VSIU)	1	—
vcmpequh	04	—	VX	VALU(VSIU)	1	—
vcmpequw	04	—	VX	VALU(VSIU)	1	—
vcmpgefp	04	—	VX	VALU(VSIU)	1	—
vcmpgtfp	04	—	VX	VALU(VSIU)	1	—
vcmpgtsb	04	—	VX	VALU(VSIU)	1	—
vcmpgtsh	04	—	VX	VALU(VSIU)	1	—
vcmpgtsw	04	—	VX	VALU(VSIU)	1	—
vcmpgtub	04	—	VX	VALU(VSIU)	1	—
vcmpgtuh	04	—	VX	VALU(VSIU)	1	—
vcmpgtuw	04	—	VX	VALU(VSIU)	1	—
vctsxs	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vctuxs	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vexptefp	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vlogefp	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vmaddfp	04	—	VA	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vmaxfp	04	—	VX	VALU(VSIU)	1	—
vmaxsb	04	—	VX	VALU(VSIU)	1	—
vmaxsh	04	—	VX	VALU(VSIU)	1	—
vmaxsw	04	—	VX	VALU(VSIU)	1	—
vmaxub	04	—	VX	VALU(VSIU)	1	—
vmaxuh	04	—	VX	VALU(VSIU)	1	—
vmaxuw	04	—	VX	VALU(VSIU)	1	—
vmhaddshs	04	—	VA	VALU(VCIU)	3:1	—

Table 6-9. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Update
vmhraddshs	04	—	VA	VALU(VCIU)	3:1	—
vminfp	04	—	VX	VALU(VSIU)	1	—
vminsb	04	—	VX	VALU(VSIU)	1	—
vminsh	04	—	VX	VALU(VSIU)	1	—
vminsw	04	—	VX	VALU(VSIU)	1	—
vminub	04	—	VX	VALU(VSIU)	1	—
vminuh	04	—	VX	VALU(VSIU)	1	—
vminuw	04	—	VX	VALU(VSIU)	1	—
vmladduhm	04	—	VA	VALU(VCIU)	3:1	—
vmrghb	04	—	VX	VPU	1	—
vmrghh	04	—	VX	VPU	1	—
vmrghw	04	—	VX	VPU	1	—
vmrglb	04	—	VX	VPU	1	—
vmrglh	04	—	VX	VPU	1	—
vmrglw	04	—	VX	VPU	1	—
vmsummbm	04	—	VA	VALU(VCIU)	3:1	—
vmsumshm	04	—	VA	VALU(VCIU)	3:1	—
vmsumshs	04	—	VA	VALU(VCIU)	3:1	—
vmsumubm	04	—	VA	VALU(VCIU)	3:1	—
vmsumuhm	04	—	VA	VALU(VCIU)	3:1	—
vmsumuhs	04	—	VA	VALU(VCIU)	3:1	—
vmulesb	04	—	VX	VALU(VCIU)	3:1	—
vmulesh	04	—	VX	VALU(VCIU)	3:1	—
vmuleub	04	—	VX	VALU(VCIU)	3:1	—
vmuleuh	04	—	VX	VALU(VCIU)	3:1	—
vmulosb	04	—	VX	VALU(VCIU)	3:1	—
vmulosh	04	—	VX	VALU(VCIU)	3:1	—
vmuloub	04	—	VX	VALU(VCIU)	3:1	—
vmulouh	04	—	VX	VALU(VCIU)	3:1	—
vnmsubfp	04	—	VA	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vnor	04	—	VX	VALU(VSIU)	1	—
vor	04	—	VX	VALU(VSIU)	1	—

Table 6-9. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Update
vperm	04	—	VA	VPU	1	—
vpkpx	04	—	VX	VPU	1	—
vpkshss	04	—	VX	VPU	1	—
vpkshus	04	—	VX	VPU	1	—
vpkswss	04	—	VX	VPU	1	—
vpkswus	04	—	VX	VPU	1	—
vpkuhum	04	—	VX	VPU	1	—
vpkuhus	04	—	VX	VPU	1	—
vpkuwum	04	—	VX	VPU	1	—
vpkuwus	04	—	VX	VPU	1	—
vrefp	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vrfim	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vrfin	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vrfip	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vrfiz	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vrlb	04	—	VX	VALU(VSIU)	1	—
vrlh	04	—	VX	VALU(VSIU)	1	—
vrlw	04	—	VX	VALU(VSIU)	1	—
vrsqrtefp	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vsel	04	—	VA	VALU(VSIU)	1	—
vsl	04	—	VX	VALU(VSIU)	1	—
vslb	04	—	VX	VALU(VSIU)	1	—
vsldoi	04	—	VA	VPU	1	—
vslh	04	—	VX	VALU(VSIU)	1	—
vslo	04	—	VX	VPU	1	—
vslw	04	—	VX	VALU(VSIU)	1	—
vspltb	04	—	VX	VPU	1	—
vsplth	04	—	VX	VPU	1	—
vspltisb	04	—	VX	VPU	1	—

Table 6-9. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Update
vspltish	04	—	VX	VPU	1	—
vspltisw	04	—	VX	VPU	1	—
vspltw	04	—	VX	VPU	1	—
vsr	04	—	VX	VALU(VSIU)	1	—
vsrab	04	—	VX	VALU(VSIU)	1	—
vsrah	04	—	VX	VALU(VSIU)	1	—
vsraw	04	—	VX	VALU(VSIU)	1	—
vsrb	04	—	VX	VALU(VSIU)	1	—
vsrh	04	—	VX	VALU(VSIU)	1	—
vsro	04	—	VX	VPU	1	—
vsrw	04	—	VX	VALU(VSIU)	1	—
vsubcuw	04	—	VX	VALU(VSIU)	1	—
vsubfp	04	—	VX	VALU(VFPU)	4:1 (non-Java)/ 5:1 (Java)	—
vsubsbs	04	—	VX	VALU(VSIU)	1	—
vsubshs	04	—	VX	VALU(VSIU)	1	—
vsubsws	04	—	VX	VALU(VSIU)	1	—
vsububm	04	—	VX	VALU(VSIU)	1	—
vsububs	04	—	VX	VALU(VSIU)	1	—
vsubuhm	04	—	VX	VALU(VSIU)	1	—
vsubuhs	04	—	VX	VALU(VSIU)	1	—
vsubuwm	04	—	VX	VALU(VSIU)	1	—
vsubuws	04	—	VX	VALU(VSIU)	1	—
vsum2sws	04	—	VX	VALU(VCIU)	3:1	—
vsum4sbs	04	—	VX	VALU(VCIU)	3:1	—
vsum4shs	04	—	VX	VALU(VCIU)	3:1	—
vsum4ubs	04	—	VX	VALU(VCIU)	3:1	—
vsumsws	04	—	VX	VALU(VCIU)	3:1	—
vupkhpx	04	—	VX	VPU	1	—
vupkhsb	04	—	VX	VPU	1	—
vupkhsh	04	—	VX	VPU	1	—
vupklpx	04	—	VX	VPU	1	—
vupklsb	04	—	VX	VPU	1	—

Table 6-9. AltiVec Instruction Latencies (continued)

Mnemonic	Primary	Extend	Form	Unit	Cycles [1]	MMU Update
vupklsh	04	—	VX	VPU	1	—
vxor	04	—	VX	VALU(VSIU)	1	—

[1] In Java mode, all VFPU instructions need a fifth execution cycle; however, data forwarding for instruction depedency can still occur at the end of the fourth execution cycle as in non-Java mode.

[2] Data streaming instructions can request a maximum of one line fetch at the L1 data cache every 2 cycles.

Chapter 7
AltiVec Technology Implementation

The AltiVec technology, a short vector parallel architecture, extends the instruction set architecture (ISA) of the architecture. The AltiVec ISA is based on separate vector/SIMD-style (single instruction stream, multiple data streams) execution units that have high-data parallelism. That is, the AltiVec technology operations can perform on multiple data elements in a single instruction. The term ‘vector’ in this document refers to the spatial parallel processing of short, fixed-length, one-dimensional matrices performed by an execution unit. It should not be confused with the temporal parallel (pipelined) processing of long, variable-length vectors performed by classical vector machines. High degrees of parallelism are achievable with simple, in-order instruction dispatch and low instruction bandwidth. However, the ISA is designed to not impede additional parallelism through superscalar dispatch in multiple execution units or multithreaded execution unit pipelines.

The AltiVec specification is defined in the *AltiVec Technology Programming Environments Manual*. That document describes but does not require many aspects of a preferred implementation. The MPC7410 implements the following key features of preferred implementation:

- All data paths and execution units are 128 bits wide.
- There are two independent AltiVec subunits, one for permute (VPU) and one for all arithmetic and logical (VALU) instructions.
- The memory subsystem is redesigned to provide very high bandwidth.
- The data stream touch instructions, **dst**(**t**) (for loads) and **dstst**(**t**) (for stores) are implemented in their full, four-tag form.

The AltiVec instruction set both defines entirely new resources and extends the functionality of the architecture. These changes are described in the following sections.

7.1 AltiVec Technology and the Programming Model

The following sections describe how the AltiVec technology affects features of the programming model as described in Chapter 2, “Programming Model.” Although the AltiVec specification describes four optional user-mode SPRs for thread management, the MPC7410 does not implement these registers.

7.1.1 Register Set

The incorporation of AltiVec technology affects the register set of the MPC7410 as described in the following sections. These features are detailed in the *AltiVec Programming Environments Manual*.

7.1.1.1 Changes to the Condition Register

AltiVec vector-compare operations with Rc set can update condition register field 6 (CR[6]) in user mode.

7.1.1.2 Addition to the Machine State Register

The AltiVec available bit, MSR[VEC], indicates the availability of the AltiVec instruction set. Its default state for the MPC7410 is a zero (not available). It can be set by the supervisor-level **mtmsr** instruction.

7.1.1.3 Vector Registers (VRs)

The AltiVec programming model defines vector registers (VRs) that are used as source and destination operands for AltiVec load, store, and computational instructions.

Figure 7-1 shows the 32 registers of the vector register file (VRF). Each is 128 bits wide and can hold sixteen 8-bit elements, eight 16-bit elements, or four 32-bit elements.

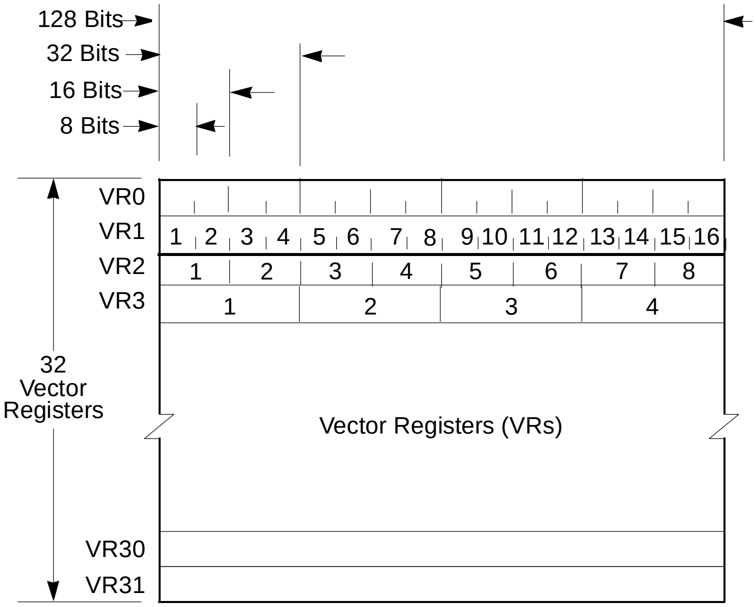


Figure 7-1. Vector Registers (VRs)

7.1.1.4 Vector Status and Control Register (VSCR)

The vector status and control register (VSCR) is a 32-bit vector register (not an SPR) that functions similarly to the FPSCR and is accessed by AltiVec instructions. The Move from Vector Status and Control Register (**mfvscr**) and Move to Vector Status and Control Register (**mtvscr**) instructions are provided to move the contents of the VSCR from and to the least-significant bits of a vector register. The VSCR is shown in Figure 7-2.

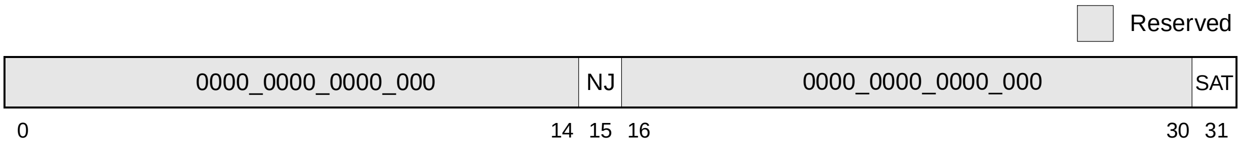


Figure 7-2. Vector Status and Control Register (VSCR)

The VSCR has two defined bits, the AltiVec non-Java mode bit (VSCR[NJ]) and the AltiVec saturation bit (VSCR[SAT]). The remaining bits are reserved.

VSCR bits are described in Table 7-1.

Table 7-1. VSCR Field Descriptions

Bits	Name	Description
0–14	—	Reserved. The handling of reserved bits is the same as that for other PowerPC registers. Software is permitted to write any value to such a bit. A subsequent reading of the bit returns 0 if the value last written to the bit was 0 and returns an undefined value (0 or 1) otherwise.
15	NJ	Non-Java. This bit determines whether AltiVec floating-point operations are performed in a Java-compliant mode or a possibly faster non-Java mode. 0 Java–compliant mode (defaultIn this mode, the AltiVec assist exception is enabled. The AltiVec assist exception allows software to handle denormalized values as specified in the Java standard. 1 Non-Java mode. (This is the default mode)If an element in a source vector register contains a denormalized value, the value 0 is used instead. If an instruction causes an underflow condition, the corresponding element in the target VR is cleared to 0. In both cases the 0 has the same sign as the denormalized or underflowing value.
16–30	—	Reserved. The handling of reserved bits is the same as that for other PowerPC registers. Software is permitted to write any value to such a bit. A subsequent reading of the bit returns 0 if the value last written to the bit was 0 and returns an undefined value (0 or 1) otherwise.
31	SAT	Saturation. This sticky status bit indicates that a field in a saturating instruction saturated since the last time SAT was cleared. It is sticky in that when SAT = 1, it remains set to 1 until it is cleared to 0 by an **mtvscr** instruction. 0 Indicates no saturation occurred; **mtvscr** can explicitly clear this bit. 1 The AltiVec saturate instruction is set when saturation occurs for the results of one of the AltiVec instructions having ‘saturation’ in its name, as follows: Move To VSCR (**mtvscr**) Vector Add Integer with Saturation (**vaddubs**, **vadduhs**, **vadduws**, **vaddsbs**, **vaddshs**, **vaddsws**) Vector Subtract Integer with Saturation (**vsububs**, **vsubuhs**, **vsubuws**, **vsubsbs**, **vsubshs**, **vsubsws**) Vector Multiply-Add Integer with Saturation (**vmhaddshs**, **vmhraddshs**) Vector Multiply-Sum with Saturation (**vmsumuhs**, **vmsumshs**, **vsumsws**) Vector Sum-Across with Saturation (**vsumsws**, **vsum2sws**, **vsum4sbs**, **vsum4shs**, **vsum4ubs**) Vector Pack with Saturation (**vpkuhus**, **vpkuwus**, **vpkshus**, **vpkswus**, **vpkshss**, **vpkswss**) Vector Convert to Fixed-Point with Saturation (**vctuxs**, **vctsxs**)

7.1.1.5 Vector Save/Restore Register (VRSAVE)

The vector save/restore register (VRSAVE) is a user-mode register used to assist application and operating system software in saving and restoring the architectural state across process context-switched events. VRSAVE is a 32-bit special-purpose register (SPR 256). VRSAVE is entirely maintained and managed by software.

VR0	VR1	VR2	VR3	VR4	VR5	VR6	VR7	VR8	VR9	VR10	VR11	VR12	VR13	VR14	VR15	VR16	VR17	VR18	VR19	VR20	VR21	VR22	VR23	VR24	VR25	VR26	VR27	VR28	VR29	VR30	VR31
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Figure 7-3. Vector Save/Restore Register (VRSAVE)

VRSAVE bit settings are shown in Table 7-2.

Table 7-2. VRSAVE Bit Settings

Bits	Name	Description
0–31	VR*n*	Determine which VRs are used in the current process. 0 Not being used for the current process 1 Used for the current process

7.1.2 AltiVec Instruction Set

The MPC7410 implements all of the defined AltiVec instructions. The AltiVec instruction set has no optional instructions; however, a few instructions associated with the load/store model are defined to allow significant differences between implementations. The following sections describe the MPC7410's implementation of these options.

AltiVec instructions are primarily user level and are divided into the following categories:

- Vector integer arithmetic instructions—These include arithmetic, logical, compare, rotate, and shift instructions.
- Vector floating-point arithmetic instructions
- Vector load and store instructions
- Vector permutation and formatting instructions—These include pack, unpack, merge, splat, permute, select, and shift instructions.
- Processor control instructions—These instructions are used to read and write from the VSCR.
- Memory control instructions—These instructions are used for managing caches (user- and supervisor-level).

7.1.2.1 LRU Instructions

The AltiVec architecture suggests that the **lvxl** and **stvxl** instructions differ from other AltiVec load and store instructions in that they leave data cache entries in a least recently used (LRU) state instead of a most recently used state (MRU). This is used to identify data known to have little reuse and poor caching characteristics.

On the MPC7410, these instructions follow the cache allocation and replacement policies described in Section 3.6, "Cache Operations," but they leave their addressed data cache entries in the LRU state. In addition, all LRU instructions are also interpreted to be transient and are treated as described in Section 7.1.2.2, "Transient Instructions and Caches."

7.1.2.2 Transient Instructions and Caches

The MPC7410 supports both static and transient memory access behavior as defined by the AltiVec technology. A static memory access assumes a reasonable degree of locality and that the data will be needed several times over a relatively long period. A transient memory reference has poor locality and is likely to be referenced few times or over a relatively short period of time.

If a memory access is designated as transient, that cache block is marked to not be cast out to the L2 unless it has been modified in the L1 data cache. If it is modified in the L1, the block is not allocated in the L2 cache when it is cast out from the L1 data cache. Instead, the block is written directly to main memory, bypassing the L2 cache.

The following instructions are interpreted to be transient:

- **lvxl** and **stvxl**
- **dstt** and **dststt** (transient forms of the two data stream touch instructions). These are described in detail in the following section.

The AltiVec architecture specifies the data stream touch instructions **dst(t)** and **dstst(t)**, and it specifies two data stream stop (**dss(all)**) instructions. The MPC7410 implements all of them. The term **dst***x* used below refers to all of the data stream touch instructions. The T field in the **dst***x* instruction is used as the transient hint bit indicator.

The instructions summarized in this section provide user-level programs the ability to manage on-chip caches; see Chapter 5, “Cache Model and Memory Coherency,” in *The Programming Environments Manual* for more information about cache topics.

Bandwidth between the processor and memory is managed explicitly through the use of cache management instructions that provide a way to indicate to the cache hardware how it should prefetch and prioritize the writeback of data. The principal instruction for this purpose is the software-directed cache prefetch Data Stream Touch (**dst**). Other related instructions are provided for complete control of the software-directed cache prefetch mechanism.

Table 7-3 summarizes the directed prefetch cache instructions defined by the AltiVec VEA. Note that these instructions are accessible to user-level programs.

Table 7-3. AltiVec User-Level Cache Instructions

Name	Mnemonic	Syntax	Implementation Notes
Data Stream Touch (non-transient)	**dst**	**r**A,**r**B,STRM	—
Data Stream Touch (transient)	**dstt**	**r**A,**r**B,STRM	Used for last access
Data Stream Touch for Store (non-transient)	**dstst**	**r**A,**r**B,STRM	Not recommended for use in the MPC7410
Data Stream Touch for Store (transient)	**dststt**	**r**A,**r**B,STRM	Not recommended for use in the MPC7410
Data Stream Stop (one stream)	**dss**	STRM	—
Data Stream Stop (all streams)	**dssall**	STRM	—

7.1.2.3 Data Stream Touch Instructions

Prefetching data to which the program is performing only store instructions does not help and can sometimes hinder performance. User-level programs should not use the touch-for-store prefetches (**dstt**, **dstst**, and **dststt**) unless the program is performing loads and stores to the data that is being prefetched. If the user is performing only stores to the data, then performance is almost certainly better if the data is not prefetched and the stores are performed independently. In this case, a **dcbz** instruction is often the best method to initialize the cache block without creating an external memory access request.

In general, touch-for-store instructions (**dstt**, **dstst**, and **dststt**) should only be used when prefetching data that is going to be both loaded and then stored. Otherwise, programmers should use the normal touch-for-load instruction (**dst**) to prefetch data that the program is loading.

If HID0[NOPDST] = 1, all subsequent **dst***x* instructions are treated as no-ops, and all previously executed **dst** streams are canceled. This no-op means that the touch does not cause a load operation and cannot perform address translation. Therefore, no table search operations are initiated, and no page table entry (PTE) referenced bits are set.

The **dst***x* instructions are broken into one or more self-initiated **dcbt**-like touch line fetches by the memory subsystem. When the **dst***x* instruction is dispatched to the LSU and all of its operands are available, the **dst***x* is queued in a vector-touch queue (VTQ) in the next cycle. There are four data stream engines within the VTQ—data stream 0 uses engine VT0 within the VTQ, data stream 1 uses VT1, and so forth.

The operation of a VT data stream engine does not consume any dispatch or completion resources. A VT is an asynchronous line-fetch or line-touch engine that can prefetch data in units of 32-byte cache blocks by inserting touch requests into the normal load/store pipeline.

After the **dst***x* is queued in the VTQ, the VTQ begins to unroll the stream into 32-byte line touches. As early as the second cycle after the LSU sends its request to the VTQ, the VTQ could make its first line-fetch touch request to the data cache.

Note that a data stream engine bases its accesses on effective addresses. This means that each line fetch within a stream accesses the data MMU simultaneously with the L1 data cache and performs a normal translation. There are no arbitrary address boundaries that affect the progress of a given stream.

In addition, if a VTQ line touch accesses a page whose translation does not reside in the data MMU, a table search operation is performed to load that PTE into the data TLB. The TLB is non-blocking during a VTQ-initiated table search operation, meaning that normal loads and stores can hit in the TLB (and in the data cache) during the table search. For details on a table search operation see Section 5.4.5.1, “Conditions for a Page Table Search Operation.”

7.1.2.3.1 Stream Engine Tags

The opcodes for the **dst***x* instructions is shown in Table 7-4.

Table 7-4. Opcodes for dstx Instructions

Name	0 5		7 8	9 10	11 15	16 20	21 30	31
dst	0111_11	0	00	STRM	A	B	01_0101_0110	0
dstst	0111_11	0	00	STRM	A	B	01_0111_0110	0
dststt	0111_11	1	00	STRM	A	B	010_111_0110	0
dstt	0111_11	1	00	STRM	A	B	01_0101_0110	0

The STRM field in the **dst***x* instruction designates which of the four data stream engines (VT0, VT1, VT2, or VT3) is used by a given instruction, as described in Table 7-5.

Table 7-5. DST[STRM] Description

Value of STRM Field in dst*x* Instruction	Data Stream Engines (VTs)
00	VT0
01	VT1
10	VT2
11	VT3

Bits 7 and 8 of the **dst***x* opcode are reserved. If bit 7 is set, it is ignored. If bit 8 is set, the VTQ does not queue up the stream and that **dst***x* instruction is ignored.

7.1.2.3.2 Speculative Execution and Pipeline Stalls for Data Stream Instructions

Like a load miss instruction or a **dcbt**/**dcbtst** instruction, a **dst***x* instruction is executed speculatively. If the target of a particular **dst***x* line fetch is mapped with G = 1 (guarded), any reload for that line fetch is under the same constraints as a guarded load. If any of the four data stream engines encounter a TLB miss, all four pause until the **dst***x* access that caused the TLB miss is retired from the completion queue or is the oldest instruction in the queue. The **dst***x* then initiates a table search operation and completes its current cache access.

If a **dst***x* instruction to a given data stream is dispatched and the VTQ is processing a previous **dst***x* to the same data stream, the second **dst** to that tag supersedes the first one, but only after the second **dst***x* becomes non-branch-speculative; it can still be speculative with respect to exceptions. If a third **dst***x* is ready for dispatch while the second is waiting for branch speculation to resolve, instruction dispatch stalls.

7.1.2.3.3 Static/Transient Data Stream Touch Instructions

Static data is likely to have a reasonable degree of locality and is referenced several times or over a reasonably long period of time. Transient data is assumed to have poor locality and is likely to be referenced only a few times over a short period of time.

The MPC7410 supports both static and transient memory-access behavior. The AltiVec ISA defines two of the **dst***x* instructions as static (**dst** and **dstst**) and two as transient (**dstt** and **dststt**).

7.1.2.3.4 Relationship with the sync/tblsync Instructions

If a **sync** instruction is executed while a **dst***x* is in progress, the following happens for each of the four VTs:

- Any cache line fetch in progress continues until that single cache line refill has completed.
- The VTQ pauses and does not continue to its next line-fetch location.
- When all other necessary conditions are met in the machine, the **sync** instruction is completed.
- The **dst***x* resumes with cache accesses/reloads to the next line-fetch location.

The effect of the **sync** is a short pause in **dst***x* operation. Code sequences that are truly intended to quiet the machine, like those used to enter reduced-power states, must use **dss**/**dssall** followed by a **sync** instruction to kill outstanding transactions initiated by **dst***x* instructions. Refer to Section 7.1.2.3.8, "Differences Between dst/dstt and dstst/dststt Instructions," for more details on the **dst***x* and **dss**/**dssall** instructions.

Note that a **tlbsync** instruction affects the VTQ identically to a **sync** instruction with the additional effect that an outstanding VTQ-initiated table search operation is canceled when a **tlbsync** is dispatched to the LSU.

7.1.2.3.5 Data Stream Termination

If one of the conditions in Table 7-6 is determined to be true when a given line fetch of a **dst***x* stream is translated, the entire **dst***x* stream is terminated. Note that this can occur in the middle of many line fetches for a **dst***x* stream.

If the condition involves address translation and the **dst***x* stream specifies an access that would cross into another page, the processor does not attempt to continue the **dst***x* stream at those new pages if it had an opportunity to fully translate the access.

Table 7-6. The dst*x* Stream Termination Conditions

Conditions
Successfully reached end of stream
The **dst***x* stream is still speculative with respect to program flow, and the control unit issues a cancel due to a mispredicted branch or exception.
Another **dst***x* instruction to this stream tag is executed, and this new **dst***x* is non-speculative with respect to branch prediction.
A **dss** instruction to this stream tag is completed.
Current line fetch caused a table search operation that did not find a matching entry in the page table.
Current line fetch is translated as cache-inhibited.
Current line fetch is translated as write-through and the stream is a touch-for-store.
Current line fetch is translated to direct-store space (SR[T] = 1).
Current line fetch is to a protected page.

Table 7-6. The dst*x* Stream Termination Conditions (continued)

Conditions
L1 data cache is locked or disabled.
The processor has encountered a condition that causes a machine check exception.

Note that asserting $\overline{\text{SRESET}}$ does not terminate a **dst***x* stream.

7.1.2.3.6 Line Fetch Skipping

When an exception condition occurs, the MPC7410 terminates any **dst***x*-initiated table search operations and pauses the stream engine that initiated the table search. In this situation, the line fetch of the **dst** that caused the table search is effectively dropped and any translation exception that would have terminated the stream had the table search operation completed does not occur. Instead, the engine attempts the next line fetch when the stream resumes. This, in effect, causes a skip of one line fetch in the stream engine.

Also note that the execution of a **tlbsync** instruction cancels any **dst***x*-initiated table search operations in progress, which can cause a line fetch skip.

7.1.2.3.7 Context Awareness and Stream Pausing

Stream accesses can take place only when data translation is enabled (MSR[DR] = 1), and when the processor is in the same privilege state as it was when the **dst***x* instruction was executed.

If the privilege level setting changes or if data translation is disabled, the stream engine suspends generation of new accesses. Any outstanding transactions initiated before the pause (like cache refills and bus activity) finish normally. The stream engine resumes when translation is again enabled and the privilege level again matches the level in place when the **dst***x* instruction for that stream was executed.

7.1.2.3.8 Differences Between dst/dstt and dstst/dststt Instructions

The only difference between touch-for-load (**dst**/**dstt**) and touch-for-store (**dstst**/**dststt)** streams is that touch-for-load streams are subdivided into line fetches that are treated identically to individual **dcbt** fetches, while touch-for-store streams are subdivided into line fetches that are treated identically to individual **dcbtst** fetches.

Note that if a touch-for-store stream instruction is mapped to a write-through page, that stream is terminated. The use of the touch-for-store streams is not recommended when store-miss merging is enabled, which is the default case. See Section 3.6.5, “Store Miss Merging,” for further details on store-miss merging.

Although the MPC7410 implements touch-for-store stream instructions, their use is discouraged. If **dstst** is used to prefetch a 32-byte a cache block that would eventually be fully consumed by 32 bytes worth of stores (that is, two back-to-back **stvx** instructions), the inclusion of touch-for-store can reduce performance for systems with limited bandwidth. This is because a touch-for-store must perform both a 32-byte coherency operation on the address bus (two or more bus cycles) and 32-bytes of data transfer (four or more 64-bit bus cycles). On the other hand, cacheable write-back stores that merge to 32 bytes require only a 32-byte coherency operation (two or more bus cycles) because of the store-miss-merging mechanism.

Because these store misses are already fully pipelined on the MPC7410, placing a touch-for-store before a series of adjacent stores that merge naturally can degrade performance.

7.1.2.3.9 Data Stream Stop (dss) and Data Stream Stop All (dssall) Instructions

The **dss** instruction is never executed speculatively. Instead, **dss** instructions flow into a four-entry **dss** queue (DSSQ) in which one entry is dedicated to each possible tag. If another **dss** is dispatched with a tag that matches a non-completed but valid DSSQ entry, that new **dss** remains in a hold queue and waits for the previous **dss** in the DSSQ to be completed.

If a subsequent **dst***x* is queued in the VTQ, it cancels an older **dss** entry in the DSSQ (for the same tag). When a given DSSQ entry completes, the valid bit for the VTQ entry corresponding to that tag is immediately cleared.

If a **dssall** instruction is executed, the DSSQ queues all four queue entries in order to terminate all four VT streams when the **dssall** instruction is the oldest. The **dssall** opcode differs from **dss** in that bit 6 (the A field) is set and bits 7–10 are ignored.

Note that line fetches in progress for a given **dst***x* stream are not canceled by the **dss** instruction. Only subsequent line fetches are prevented. To ensure that all line fetches from a **dst***x* are completed, a **sync** instruction must be issued after the **dss** instruction.

7.1.3 Vector Floating Point Data Considerations

This section describes the MPC7410 floating-point behavior for various special-case data types. The descriptions cover both Java and non-Java modes (see Section 7.1.1.4, “Vector Status and Control Register (VSCR)” for setting Java/non-Java mode), including the following:

- Denormalization for all instructions
- NaNs, denormalized numbers, and zeros for compare, min, and max MPC7410 operations
- Zero and NaN data for round-to-float integral operations

Note the following:

- The MPC7410 defaults to non-Java mode.
- TheMPC7410 handles NaNs the same way regardless of Java or non-Java mode.
- The MPC7410 handles most denormalized numbers in Java mode by taking a trap to exception 0x01600 (AltiVec assist exception) but, for some instructions the MPC7410 can produce the exact result without trapping.

Table 7-7 describes denormalization instructions.

Table 7-7. Denormalization for AltiVec Instructions

Instruction	Input Denormalization Detected		Output Denormalization Detected	
	Java	Non-Java	Java	Non–Java
vaddfp, **vsubfp**, **vmaddfp**, **vnmsubfp**	Trap (unlessanother input is a NaN) [1]	Input treated as correctly signed zero	Trap	Result squashed to correctly signed zero
vrefp	Trap	Denormalized number squashed to zero, returning +/-∞	Trap	Result squashed to zero
vrsqrtefp	Trap	Denormalized number squashed to zero, returning +/-∞	Never produces a denormalized number	Never produces a denormalized number
vlogefp	Trap	Denormalized number squashed to zero, returning -∞	Never produces a denormalized number	Never produces a denormalized number
vexptefp	Result is +1.0	Input squashed to zero, output result is +1.0	Trap	Result squashed to zero
vcfux, vcfsx	Never detects denormalized numbers			
vctsxs, vctuxs	Trap [1]	Output result is 0x0	Never produces a denormalized number	Never produces a denormalized number

[1] May change in the future to produce an IEEE default result in hardware instead of trapping.

Table 7-8 describes the behavior of the vector floating-point compare, min, and max instructions in non-Java mode.

Table 7-8. Vector Floating-Point Compare, Min, and Max in Non-Java Mode

vA	vB	vminfp	vmaxfp	vcmpgtfp	vcmpgefp	vcmpeqfp	vcmpbfp	
							LE	GE
NaN_A	—	QNaN_A	QNaN_A	False	False	False	0	0
—	NaN_B	QNaN_B	QNaN_B	False	False	False	0	0
+Den_A	-B	-B	+Zero	True	True	False	0	0
-Den_A	-B	-B	-Zero	True	True	False	0	0
+Den_A	+B	+Zero	+B	False	False	False	1	1
-Den_A	+B	-Zero	+B	False	False	False	1	1
-A	+Den_B	-A	+Zero	False	False	False	1	0
-A	-Den_B	-A	-Zero	False	False	False	1	0
+A	+Den_B	+Zero	+A	True	True	False	0	1
+A	-Den_B	-Zero	+A	True	True	False	0	1

Table 7-8. Vector Floating-Point Compare, Min, and Max in Non-Java Mode (continued)

vA	vB	vminfp	vmaxfp	vcmpgtfp	vcmpgefp	vcmpeqfp	vcmpbfp	
							LE	GE
+Den_A/+Zero	+Den_B/+Zero	+Zero	+Zero	False	True	True	1	1
+Den_A/+Zero	-Den_B/-Zero	-Zero	+Zero	False	True	True	1	1
-Den_A/-Zero	+Den_B/+Zero	-Zero	+Zero	False	True	True	1	1
-Den_A/-Zero	-Den_B/-Zero	-Zero	-Zero	False	True	True	1	1

Table 7-9 describes the behavior of the same instructions in Java mode.

Table 7-9. Vector Floating-Point Compare, Min, and Max in Java Mode

vA	vB	vminfp	vmaxfp	vcmpgtfp	vcmpgefp	vcmpeqfp	vcmpbfp	
							LE	GE
NaN_A	—	QNaN_A	QNaN_A	False	False	False	0	0
—	NaN_B	QNaN_B	QNaN_B	False	False	False	0	0
+Den_A	-B	-B	+Den_A	True	True	False	0	0
-Den_A	-B	-B	-Den_A	True	True	False	0	0
+Den_A	+B	+Den_A	+B	False	False	False	1	1
-Den_A	+B	-Den_A	+B	False	False	False	1	1
-A	+Den_B	-A	+Den_B	False	False	False	1	0
-A	-Den_B	-A	-Den_B	False	False	False	1	0
+A	+Den_B	+Den_B	+A	True	True	False	0	1
+A	-Den_B	-Den_B	+A	True	True	False	0	1
+Den_A	±Zero	±Zero	+Den_A	True	True	False	0	1
-Den_A	±Zero	-Den_A	±Zero	False	False	False	1	0
±Zero	+Den_B	±Zero	+Den_B	False	False	False	1	1
±Zero	-Den_B	-Den_B	±Zero	True	True	False	0	0
-Den_A	+Den_B	-Den_A	+Den_B	False	False	False	1	Result depends on input operands
+Den_A	-Den_B	-Den_B	+Den_A	True	True	False	0	
-Den_A	-Den_B	Result depends on input operands						0
+Den_A	+Den_B							1

Table 7-10 describes the behavior of round-to-integer instructions in non-Java mode.

Table 7-10. Round-to-Integer Instructions in Non-Java Mode

vB Sign	vB exponent	Instruction			
		vrfin	**vrfiz**	**vrfip**	**vrfim**
neg	127 > exp > 24	vB	vB	vB	vB
	23 > exp > 0	Round towards nearest	Truncate fraction	Round towards +∞	Round towards -∞
	Exp = -1	Round to nearest	-Zero	-Zero	-1.0
	-2 > exp > -126	-Zero	-Zero	-Zero	-1.0
	Input is denormalized	-Zero	-Zero	-Zero	-Zero
	Input is zero	-Zero	-Zero	-Zero	-Zero
pos	input is zero	+Zero	+Zero	+Zero	+Zero
	Input is denormalized	+Zero	+Zero	+Zero	+Zero
	-126 < exp < -2	+Zero	+Zero	+1.0	+Zero
	exp = -1	Round towards nearest	+Zero	+1.0	+Zero
	0 < exp < 23	Round towards nearest	Truncate fraction	Round towards +∞	Round towards -∞
	24 < exp < 126	vB	vB	vB	vB

Table 7-11 describes round-to-integer instructions in Java mode. Note that round-to-integer instructions never produce denormalized numbers.

Table 7-11. Round-to-Integer Instructions in Java Mode

vB Sign	vB Exponent	Instruction			
		vrfin	**vrfiz**	**vrfip**	**vrfim**
neg	127 > exp > 24	vB	vB	vB	vB
	23 > exp > 0	Round towards nearest	Truncate fraction	Round towards +∞	Round towards -∞
	Exp = -1	Round to nearest	-Zero	-Zero	-1.0
	-2 > exp > -126	-Zero	-Zero	-Zero	-1.0
	Input is denormalized	Trap	Trap	Trap	Trap
	Input is zero	-Zero	-Zero	-Zero	-Zero

Table 7-11. Round-to-Integer Instructions in Java Mode (continued)

vB Sign	vB Exponent	Instruction			
		vrfin	**vrfiz**	**vrfip**	**vrfim**
pos	Input is zero	+Zero	+Zero	+Zero	+Zero
	Input is denormalized	Trap	Trap	Trap	Trap
	-126 < exp < -2	+Zero	+Zero	+1.0	+Zero
	Exp = -1	Round towards nearest	+Zero	+1.0	+Zero
	0 < exp < 23	Round to nearest	Truncate fraction	Round To +∞	Round To -∞
	24 < exp < 126	vB	vB	vB	vB

The MPC7410 detects underflows and production of denormalized numbers on vector float results before rounding, not after. Future versions of the *AltiVec Technology Programming Environments Manual* may reflect this ordering.

7.2 AltiVec Technology and the Cache Model

The MPC7410 uses a unified LSU to load and store operands into the GPRs, FPRs, and VRs. The MPC7410's high-bandwidth memory subsystem supports anticipated AltiVec workloads.

The memory subsystem features summarized in the following sections combine to provide high bandwidth while maintaining latencies and cache capacities similar to the MPC750.

The following list summarizes features of the MPC7410 L1 cache implementation that affect the AltiVec implementation:

- The 32-Kbyte, 8-way set associative L1 data cache is fully non-blocking.
  - The 128-bit interface is designed to support AltiVec load/store operations.
  - It supports both MRU (most recently used) and LRU (least recently used) vector loads.
  - New castout and modified bits support **lvx**/**stvx** LRU operations
- Pseudo LRU (PLRU) replacement algorithm for L1 cache
- Support for AltiVec LRU instructions. LRU instructions are described in Section 7.1.2.1, "LRU Instructions."
- Support for AltiVec transient instructions. Transient instructions are described in Section 7.1.2.2, "Transient Instructions and Caches."

7.3 AltiVec and the Exception Model

Only the three following exceptions can result from execution of an AltiVec instruction:

- An AltiVec unavailable exception occurs when executing any non-stream AltiVec instruction with MSR[VEC] = 0. After this exception occurs, execution resumes at offset 0x00F20 from the base physical address indicated by MSR[IP]. This exception does not occur for data streaming instructions (**dst(t)**, **dstst(t)**, and **dss**). Also note that VRSAVE is not protected by this exception which is consistent with the *AltiVec Programming Environments Manual*. Thus, any access to the VRSAVE register does not cause an exception when MSR[VEC] = 0.

- A DSI exception occurs only if an AltiVec load or store operation encounters a protection violation or a page fault (does not find a valid PTE during a table search operation).
- An AltiVec assist exception may occur if an AltiVec floating-point instruction detects denormalized data as an input or output in Java mode.

7.4 AltiVec and the Memory Management Model

The AltiVec functionality in the MPC7410 affects the MMU model in the following ways:

- A data stream instruction (**dst**(**t**) or **dstst**(**t**)) can cause table search operations to occur after the instruction is retired.
- MMU exception conditions can cause a data stream operation to abort.
- Aborted VTQ-initiated table search operations can cause a line fetch skip.
- Execution of a **tlbsync** instruction can cancel an outstanding table search operation for a VTQ.

Data stream touch instructions may use either of the two translation mechanisms as specified by the architecture—segment/page or BAT. For more information, see Chapter 5, “Memory Management.”

7.5 AltiVec Technology and Instruction Timing

AltiVec computational instructions are executed in the four independent pipelined AltiVec execution units. The VPU has a single-stage pipeline, the VSIU has a single-stage pipeline, the VCIU has a three-stage pipeline, and the VFPU has a four-stage pipeline. The AltiVec technology defines additional data streaming instructions to help improve throughput. Those instructions are described in Section 7.1.2.3, “Data Stream Touch Instructions.” A complete description of the AltiVec instruction timing is provided in Chapter 6, “Instruction Timing.”

Chapter 8
Signal Descriptions

This chapter describes the MPC7410 microprocessor's external signals. It contains a concise description of individual signals, showing behavior when the signal is asserted, negated, or tristated, and when the signal is an input or an output.

NOTE

> A bar over a signal name indicates that the signal is active low—for example, $\overline{\text{ARTRY}}$ (address retry) and $\overline{\text{TS}}$ (transfer start). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as AP[0:3] (address bus parity signals) and TT[0:4] (transfer type signals) are referred to as asserted when they are high and negated when they are low.

The MPC7410 provides a mode switch (via the $\overline{\text{EMODE}}$ signal) that enables either the 60x bus protocol or MPX bus protocol operation. The 60x bus interface implements the protocol described in the *PowerPC Microprocessor Family: The Bus Interface for 32-Bit Microprocessors*; note that although this protocol is implemented by the MPC603e, MPC604 and MPC740/750 processors, it is referenced as the 60x bus interface. The MPX bus mode includes several additional features that allow it to provide higher memory bandwidth than the 60x bus.

Refer to the MPC7410 hardware specification for detailed electrical and mechanical information for each signal.

8.1 Signal Groupings

The MPC7410 60x bus and MPX bus interface protocol signals are grouped as follows:

- Address arbitration—The MPC7410 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 MPC7410 uses these signals to arbitrate for data bus mastership.

- Data transfer—These signals, which consist of the data bus and data parity, 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, the data termination signals also indicate the end of the tenure. In burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat. The data termination signals also indicate whether a condition exists that requires the data phase to be repeated.

In addition there are many other signals on the MPC7410 that control and affect other aspects of the device, aside from the bus protocol as follows:

- L2 cache address/data—The MPC7410 has separate address and data buses for accessing the L2 cache.
- L2 cache clock/control—These signals provide clocking and control for the L2 cache.
- Interrupts/resets—These signals include the external interrupt signal, checkstop signals, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the processor.
- Processor status and control—These signals are used to set the reservation coherency bit, and enable the time base and other functions. They are also used in conjunction with such resources as secondary caches and the time base facility.
- Clock control—These signals determine the system clock frequency. They are also used to synchronize multiprocessor systems.
- Test interface—The JTAG (IEEE Std. 1149.1a™) 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 select— These signals control the voltages of the L2 interface and the rest of the device.

8.1.1 Signal Summary

Table 8-1 lists all the MPC7410 signals in alphabetical order and provides a cross-reference to the section of this chapter that contains the detailed description for each. The table also shows which signals provide multiple functions and are multiplexed on the MPC7410.

Table 8-1. MPC7410 Signal Cross Reference

Signal	Signal Name	Interface	Alternate Function	Pins	I/O	Section #
A[0:31]	Address	60x, MPX	—	32	I/O	8.2.3.1 8.4.3 (MPX)
$\overline{\text{AACK}}$	Address acknowledge	60x, MPX	—	1	I	8.2.5.1 8.4.5.1 (MPX)
$\overline{\text{ABB}}$	Address bus busy	60x	$\overline{\text{AMON}}$	1	O	8.2.2.3
$\overline{\text{AMON}}$	Address bus monitor	MPX	$\overline{\text{ABB}}$	1	O	8.4.2.3
AP[0:3]	Address Parity	60x, MPX	—	4	I/O	8.2.3.2 8.4.3(MPX)
$\overline{\text{ARTRY}}$	Address retry	60x, MPX	—	1	I/O	8.2.5.2 8.4.5.2 (MPX)
$\overline{\text{BG}}$	Bus grant	60x, MPX	—	1	I	8.2.2.2 8.4.2.2 (MPX)
$\overline{\text{BR}}$	Bus request	60x, MPX	—	1	O	8.2.2.1 8.4.2.1 (MPX)
BVSEL	Bus voltage select	60x, MPX	—	1	I	8.5.7.1
$\overline{\text{CI}}$	Cache-inhibited	60x, MPX	—	1	I/O	8.2.4.7 8.4.4.8 (MPX)
$\overline{\text{CHK}}$	Check	60x, MPX	—	1	I	8.5.3.7
$\overline{\text{CKSTP\_IN}}$	Checkstop in	60x, MPX	—	1	I	8.5.3.5
$\overline{\text{CKSTP\_OUT}}$	Checkstop out	60x, MPX	—	1	O	8.5.3.6
CLK_OUT	Clock out	60x, MPX	—	1	O	8.5.5.3
$\overline{\text{DBB}}$	Data bus busy	60x	$\overline{\text{DMON}}$	1	O	8.2.6.3
$\overline{\text{DBG}}$	Data bus grant	60x, MPX	—	1	I	8.2.2.2 8.4.6.1 (MPX)
$\overline{\text{DBWO}}$	Data bus write only	60x	DTI0	1	I	8.2.6.2
DH[0:31]	Data bus high 0:31	60x, MPX	—	32	I/O	8.2.7.1 8.4.7.1 (MPX)
DL[0:31]	Data bus low 0:31	60x, MPX	—	32	I/O	8.2.7.1 8.4.7.1 (MPX)
$\overline{\text{DMON}}$	Data bus monitor	MPX	$\overline{\text{DBB}}$	1	O	8.4.6.4
DP[0:7]	Data parity	60x, MPX	—	8	I/O	8.2.7.2 8.4.3 (MPX)

Table 8-1. MPC7410 Signal Cross Reference (continued)

Signal	Signal Name	Interface	Alternate Function	Pins	I/O	Section #
$\overline{\text{DRDY}}$	Data ready	MPX	—	1	O	8.4.6.3
DTI0	Data transaction index	MPX	$\overline{\text{DBWO}}$	1	I	8.4.6.2
DTI[1:2]	Data transaction index	MPX	—	2	I	8.4.6.2
$\overline{\text{EMODE}}$	Enhanced mode	60x, MPX	—	1	I	8.5.4.5
GBL	Global	60x, MPX	—	1	I/O	8.2.4.5 8.4.4.6 (MPX)
HIT	Snoop hit	MPX	—	1	O	8.4.5.4
$\overline{\text{HRESET}}$	Hard reset	60x, MPX	—	1	I	8.5.3.4.2
INT	Interrupt request	60x, MPX	—	1	I	8.5.3.1
L2ADDR[18:0]	L2 address	60x, MPX	—	19	O	8.5.1.1
$\overline{\text{L2CE}}$	L2 chip enable	60x, MPX	—	1	O	8.5.2.1
L2CLK_OUT[A:B]	L2	60x, MPX	—	2	O	8.5.2.3
L2DATA[0:63]	L2 data	60x, MPX	—	64	I/O	8.5.1.2
L2DP[0:7]	L2 data parity	60x, MPX	—	8	I/O	8.5.1.3
L2SYNC_IN	L2 sync in	60x, MPX	—	1	I	8.5.2.6
L2SYNC_OUT	L2 sync out	60x, MPX	—	1	O	8.5.2.5
L2VSEL	L2 voltage select	60x, MPX	—	1	I	8.5.7.2
$\overline{\text{L2WE}}$	L2 write enable	60x, MPX	—	1	O	8.5.2.2
L2ZZ	L2 low-power mode enable	60x, MPX	—	1	O	8.5.2.7
$\overline{\text{MCP}}$	Machine check	60x, MPX	—	1	I	8.5.3.3
PLL_CFG[0:3]	PLL configuration	60x, MPX	—	4	I	8.5.5.2
$\overline{\text{QACK}}$	Quiesce acknowledge	60x, MPX	—	1	I	8.5.4.4
$\overline{\text{QREQ}}$	Quiesce request	60x, MPX	—	1	O	8.5.4.3
$\overline{\text{RSRV}}$	Reservation	60x, MPX	—	1	O	8.5.4.1
$\overline{\text{SRESET}}$	Soft reset	60x, MPX	—	1	I	8.5.3.4.1
$\overline{\text{SHD}}$	Shared	60x	$\overline{\text{SHD0}}$	1	I/O	8.2.5.3
$\overline{\text{SHD0}}$	Shared 0	MPX	$\overline{\text{SHD}}$	1	I/O	8.4.5.3
$\overline{\text{SHD1}}$	Shared 1	MPX	—	1	I/O	8.4.5.3
$\overline{\text{SMI}}$	System management interrupt	60x, MPX	—	1	I	8.5.3.2
SYSCLK	System clock	60x, MPX	—	1	I	8.5.5.1
$\overline{\text{TA}}$	Transfer acknowledge	60x, MPX	—	1	I	8.2.8.1 8.4.8.1 (MPX)

Table 8-1. MPC7410 Signal Cross Reference (continued)

Signal	Signal Name	Interface	Alternate Function	Pins	I/O	Section #
TBEN	Time base enable	60x, MPX	—	1	I	8.5.4.2
$\overline{\text{TBST}}$	Transfer burst	60x, MPX	—	1	O	8.2.4.3 8.4.4.4 (MPX)
TCK	Scan clock	60x, MPX	—	1	I	8.5.6.1
TDI	Serial scan input	60x, MPX	—	1	I	8.5.6.2
TDO	Serial scan output	60x, MPX	—	1	O	8.5.6.3
$\overline{\text{TEA}}$	Transfer error acknowledge	60x, MPX	—	1	I	8.2.8.2 8.4.8.2 (MPX)
TMS	Test mode select	60x, MPX	—	1	I	8.5.6.4
$\overline{\text{TS}}$	Transfer start	60x, MPX	—	1	I/O	Figure 8.2.4.1 8.4.4 (MPX)
$\overline{\text{TRST}}$	Test reset	60x, MPX	—	1	I	8.5.6.5
TSIZ[0:2]	Transfer size	60x, MPX	—	3	O	8.2.4.4 8.4.4.5 (MPX)
TT[0:4]	Transfer type	60x, MPX	—	5	I/O	8.2.4.2 8.4.4.3 (MPX)
$\overline{\text{WT}}$	Write-through	60x, MPX	—	1	I/O	8.2.4.6 8.4.4.7 (MPX)

8.1.2 60x Bus and MPX Bus Output Signal States During Reset

The assertion of $\overline{\text{HRESET}}$ causes all bi-directional signals to be in the input state. Table 8-2 shows the state of MPC7410 output signals during $\overline{\text{HRESET}}$ assertion.

Table 8-2. Output Signal States During System Reset

Signal Group	Signals	State During System Reset
Address Arbitration	$\overline{\text{BR}}$ $\overline{\text{ABB}}$/$\overline{\text{AMON}}$	High impedance
Address Bus	A[0:31] AP[0:3]	High impedance
Address Transfer Attributes	$\overline{\text{TBST}}$ TSIZ[0:2]	High impedance
Address Termination	$\overline{\text{HIT}}$	High impedance
Data Arbitration	$\overline{\text{DRDY}}$ $\overline{\text{DBB}}$/$\overline{\text{DMON}}$	High impedance
L2 Cache Address/Data	L2ADD[17:0]	High impedance

Table 8-2. Output Signal States During System Reset (continued)

Signal Group	Signals	State During System Reset
L2 Cache Clock/Control	$\overline{\text{L2CE}}$ $\overline{\text{L2WE}}$ L2CLK_OUT[A:B] L2SYNC_OUT L2ZZ	Driven negated Driven negated Driven low Driven low Driven negated
Interrupts/resets	$\overline{\text{CKSTP\_OUT}}$	Driven negated
Processor Status/Control	$\overline{\text{RSRV}}$ $\overline{\text{QREQ}}$	High impedance
Clock Control	CLK_OUT	Bus clock (SYSCLK)
Test interface	TDO	High impedance

8.2 60x Bus Signal Configuration

The following sections describe the signals that implement the 60x bus protocol on the MPC7410. The MPX bus protocol signals start in Section 8.4, “MPX Bus Signal Configuration,” on page 8-23.

8.2.1 60x Bus Functional Groupings

Figure 8-1 illustrates the MPC7410’s signal configuration in 60x bus mode, showing how the signals are grouped. A pinout showing pin numbers is included in the MPC7410 hardware specification. Note that the left side of the figure depicts the signals that implement the 60x bus protocol and the right side of the figure shows the remaining signals on the MPC7410 (not part of the bus protocol).

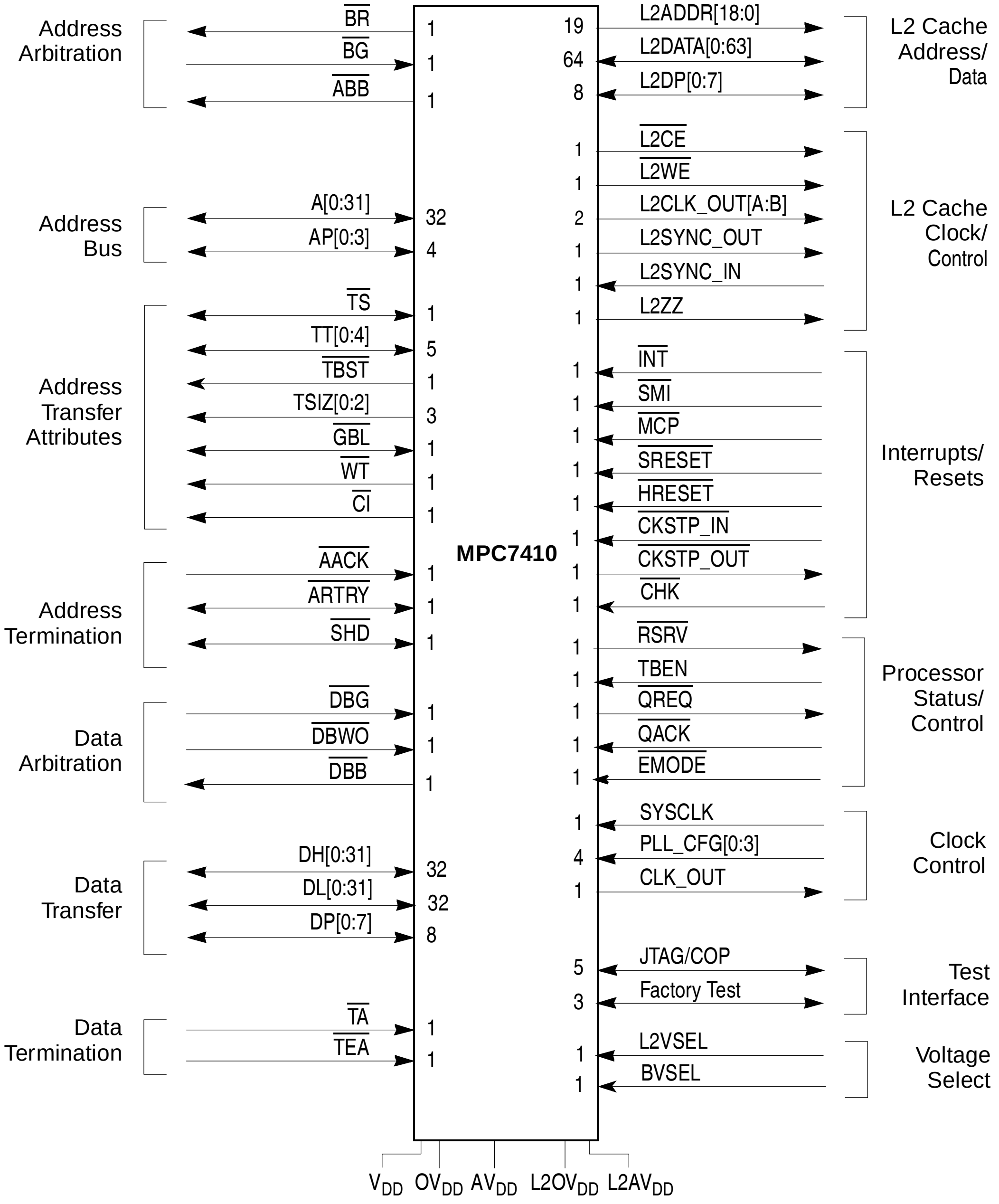


Figure 8-1. 60x Bus Signal Groups

Note that the following sections summarize signal functions. Chapter 9, “System Interface Operation,” describes many of these signals in greater detail, both with respect to how individual signals function and how groups of signals interact.

8.2.2 Address Bus Arbitration Signals

The address arbitration signals are input and output signals the MPC7410 uses to request the address bus, recognize when the request is granted, and indicate to other devices when mastership is granted. For a detailed description of how these signals interact, see Section 9.3.1, “Address Bus Arbitration.”

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

Following are the state meaning and timing comments for the $\overline{\text{BR}}$ output signal.

State Meaning Asserted—Indicates that the MPC7410 is requesting mastership of the address bus. Note that $\overline{\text{BR}}$ may be asserted for one or more cycles, and then negated due to an internal cancellation of the bus request. See Section 9.3.1, “Address Bus Arbitration,” for more information.

Negated—Indicates that the MPC7410 is not requesting the address bus. The MPC7410 may have no bus operation pending, the address bus may be parked, or the $\overline{\text{ARTRY}}$ input was asserted on the previous bus clock cycle.

Timing Comments Assertion—Occurs when the MPC7410 is not parked and a bus transaction is needed.

Negation—Occurs for at least one bus clock cycle after an accepted, qualified bus grant (see $\overline{\text{BG}}$), even if another transaction is pending. It is also negated for at least one bus clock cycle when the assertion of $\overline{\text{ARTRY}}$ is detected on the bus.

High Impedance—Occurs during a hard reset or checkstop condition.

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

Following are the state meaning and timing comments for the $\overline{\text{BG}}$ input signal.

State Meaning Asserted—Indicates that the MPC7410 may, with proper qualification, assume mastership of the address bus. The conditions for a qualified bus grant are described in Section 9.3.1, “Address Bus Arbitration.”

Negated— Indicates that the MPC7410 is not the next potential address bus master.

Timing Comments Assertion—May occur at any time to indicate the MPC7410 can use the address bus. In 60x bus mode, the MPC7410 does not accept a $\overline{\text{BG}}$ in the cycles between the assertion of any $\overline{\text{TS}}$ and $\overline{\text{AACK}}$.

Negation—May occur at any time to indicate the MPC7410 cannot use the bus. The MPC7410 may still assume bus mastership on the bus clock cycle of the negation of $\overline{\text{BG}}$ because during the previous cycle $\overline{\text{BG}}$ indicated to the MPC7410 that it could take mastership (if qualified).

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

Unlike other processors that implement the 60x bus protocol, the address bus busy ($\overline{\text{ABB}}$) signal is strictly an output signal on the MPC7410. Use of this signal is optional in the 60x bus protocol. See Section 9.3.1, "Address Bus Arbitration," for a detailed description of the operation of $\overline{\text{ABB}}$ in the MPC7410. Following are the state meaning and timing comments for $\overline{\text{ABB}}$.

State Meaning Asserted—Indicates that the MPC7410 is the address bus master. See Section 9.3.1, "Address Bus Arbitration."

Negated—Indicates that the MPC7410 is not using the address bus. If $\overline{\text{ABB}}$ is negated during the bus clock cycle following a qualified bus grant, the MPC7410 did not accept mastership even if $\overline{\text{BR}}$ was asserted. This can occur if a potential transaction is aborted internally before the transaction begins.

Timing Comments Assertion—Occurs on the bus clock cycle following a qualified $\overline{\text{BG}}$ that is accepted by the processor.

Negation—Occurs for a minimum of one-half bus clock cycle following the assertion of $\overline{\text{AACK}}$. If $\overline{\text{ABB}}$ is negated during the bus clock cycle after a qualified bus grant, the MPC7410 did not accept mastership, even if $\overline{\text{BR}}$ was asserted.

High Impedance—Occurs after $\overline{\text{ABB}}$ is negated.

8.2.3 Address Transfer Signals

The address transfer signals are used to transmit the address and to generate and monitor parity for the address transfer. For a detailed description of how these signals interact, refer to Section 9.3.2, "Address Transfer."

8.2.3.1 Address Bus (A[0:31])

The address bus (A[0:31]) consists of 32 signals that are both input and output signals.

8.2.3.1.1 Address Bus (A[0:31])—Output

Following are the state meaning and timing comments for the A[0:31] output signals.

State Meaning Asserted/Negated—Represents the physical address (real address in the architecture specification) of the data to be transferred. On burst transfers, the address bus presents the double-word-aligned address containing the critical code/data that missed the cache on a read operation, or the first double word of the cache line on a write operation. Note that the address output during burst operations is not incremented. See Section 9.3.2, "Address Transfer."

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

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

8.2.3.1.2 Address Bus (A[0:31])—Input

Following are the state meaning and timing comments for the A[0:31] input signals.

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

Timing Comments Assertion/Negation—Must be valid on the same bus clock cycle as the assertion of $\overline{\text{TS}}$; it is sampled by MPC7410 only on this cycle.

8.2.3.2 Address Bus Parity (AP[0:3])

The address bus parity (AP[0:3]) signals, both input and output, reflect one bit of odd-byte parity for each of the 4 bytes of address when a valid address is on the bus.

8.2.3.2.1 Address Bus Parity (AP[0:3])—Output

Following are the state meaning and timing comments for the AP[0:3] output signals on the MPC7410.

State Meaning Asserted/Negated—Represents odd parity for each of the 4 bytes of the physical address for a transaction. Odd parity means that an odd number of bits, including the parity bit, are driven high. Table 8-3 shows the address parity signal assignments. For more information, see Section 9.3.2.1, “Address Bus Parity.”

Table 8-3. Address Parity Bit Assignments

Address Parity Bit	Address Bus Signals
AP0	A[0:7]
AP1	A[8:15]
AP2	A[16:23]
AP3	A[24:31]

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

High Impedance—The same as A[0:31]

8.2.3.2.2 Address Bus Parity (AP[0:3])—Input

Following are the state meaning and timing comments for the AP[0:3] input signal on the MPC7410.

State Meaning Asserted/Negated—Represents odd parity for each of the 4 bytes of the physical address for snooping operations. Detected even parity causes the processor to take a machine check exception or enter the checkstop state if address parity checking is enabled (HID0[EBA] = 1); see Section 2.1.5.1, “Hardware Implementation-Dependent Register 0 (HID0).”

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

8.2.4 Address Transfer Attribute Signals

The transfer attribute signals are a set of signals that characterize the following:

- The size of the transfer
- Whether it is a read or write operation.
- Whether it is a burst or single-beat transfer.

For a detailed description of how these signals interact, see Section 9.3.2, “Address Transfer.”

8.2.4.1 Transfer Start ($\overline{\text{TS}}$)

The address transfer start ($\overline{\text{TS}}$) signal is both an input and an output signal on the MPC7410, and indicates that an address bus transfer has begun.

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

Following are the state meaning and timing comments for the $\overline{\text{TS}}$ output signal.

State Meaning Asserted—Indicates that the MPC7410 has begun a bus transaction and that the address bus and transfer attribute signals are valid. When asserted with the appropriate TT[0:4] signals; it is also an implied data bus request for a memory transaction (unless it is an address-only operation).

Negated—Indicates that no bus transaction is occurring during normal operation.

Timing Comments Assertion—May occur on any cycle following a qualified $\overline{\text{BG}}$.

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

High Impedance—Occurs two bus clock cycles after $\overline{\text{TS}}$ is asserted.

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

Following are the state meaning and timing comments for the $\overline{\text{TS}}$ input signal.

State Meaning Asserted—Indicates that another master has begun a bus transaction and that the address bus and transfer attribute signals are valid for snooping; see Section 8.2.4.5, “Global (GBL).”

Negated—Indicates that no bus transaction is occurring.

Timing Comments Assertion—May occur on any cycle following a qualified $\overline{\text{BG}}$.

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

8.2.4.2 Transfer Type (TT[0:4])

The transfer type (TT[0:4]) signals consist of five input/output signals on the MPC7410. For a complete description of TT[0:4] signals and for transfer type encodings, see Section 9.3.2.2.1, “Transfer Type (TT[0:4]) Signals in 60x Bus Mode.”

8.2.4.2.1 Transfer Type (TT[0:4])—Output

Following are the state meaning and timing comments for the TT[0:4] output signals on the MPC7410.

State Meaning Asserted/Negated—Indicates the type of transfer in progress.

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

High Impedance—The same as A[0:31]

8.2.4.2.2 Transfer Type (TT[0:4])—Input

Following are the state meaning and timing comments for the TT[0:4] input signals on the MPC7410.

State Meaning Asserted/Negated—Indicates the type of transfer in progress.

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

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

Unlike other processors that implement the 60x bus protocol, the transfer burst ($\overline{\text{TBST}}$) signal is an output-only signal on the MPC7410.

Following are the state meaning and timing comments for the $\overline{\text{TBST}}$ output signal.

State Meaning Asserted—Indicates that a burst transfer is in progress.

For transactions initiated by external control instructions (**eciwx** and **ecowx**), $\overline{\text{TBST}}$ forms part of the 4-bit Resource ID field on the bus as follows:

$\overline{\text{TBST}}$ || TSIZ(0:2) ← EAR(28:31)

Negated—Indicates that a burst transfer is not in progress.

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

High Impedance—The same as A[0:31]

8.2.4.4 Transfer Size (TSIZ[0:2])—Output

Following are the state meaning and timing comments for the transfer size (TSIZ[0:2]) output signals on the MPC7410.

State Meaning Asserted/Negated—For memory accesses, these signals along with $\overline{\text{TBST}}$, indicate the data transfer size for the current bus operation. See Section 9.3.2.2.2, "Transfer Size (TSIZ[0:2]) Signals." Also, Section 9.3.2.4, "Effect of Alignment in Data Transfers," shows how the transfer size signals are used with the address signals for aligned and misaligned transfers. Note that the MPC7410 does not generate all possible TSIZ[0:2] encodings.

For transactions initiated by external control instructions (**eciwx** and **ecowx**), TSIZ[0:2] signals form part of the 4-bit resource ID field (they are used to output bits 29–31 of the external access register (EAR)) on the bus as follows:

$\overline{\text{TBST}}$ || TSIZ(0:2) ← EAR(28:31)

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

High Impedance—The same as A[0:31]

8.2.4.5 Global ($\overline{\text{GBL}}$)

The global ($\overline{\text{GBL}}$) signal is an input/output signal on the MPC7410.

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

Following are the state meaning and timing comments for the $\overline{\text{GBL}}$ output signal.

State Meaning Asserted—Indicates that a transaction is global, reflecting the setting of the M bit for the block or page that contains the address of the current transaction (except during certain data cache, memory synchronization, TLB management, and external control operations as described in Table 3-16 on page 3-74). Thus, this transaction must be snooped.

Negated—Indicates that a transaction is not global and does not need to be snooped by other masters.

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

High Impedance—The same as A[0:31]

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

Following are the state meaning and timing comments for the $\overline{\text{GBL}}$ input signal.

State Meaning Asserted—Indicates that a transaction must be snooped by the MPC7410.

Negated—Indicates that a transaction must not be snooped by the MPC7410.

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

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

The write-through ($\overline{\text{WT}}$) signal is an output signal on the MPC7410 in 60x bus mode. Following are the state meaning and timing comments for the $\overline{\text{WT}}$ signal in 60x bus mode.

State Meaning Asserted—Indicates that a single-beat write transaction is write-through, reflecting the value of the W bit for the block or page that contains the address of the current transaction (except during certain data cache, memory synchronization, TLB management, and external control operations as described in Table 3-16).

Note that on the MPC750, $\overline{\text{WT}}$ assertion during a read operation indicates an instruction fetch. The MPC7410 does not use $\overline{\text{WT}}$ to indicate instruction fetches. Instead, the MPC7410 uses the TT0 signal (if HID0[IFFT] = 1) to indicate an instruction fetch.

Negated—Indicates that a write transaction is not write-through.

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

High Impedance—The same as A[0:31]

8.2.4.7 Cache Inhibit ($\overline{\text{CI}}$)—Output

The cache inhibit ($\overline{\text{CI}}$) signal is an output signal on the MPC7410 in 60x bus mode. Following are the state meaning and timing comments for the $\overline{\text{CI}}$ signal in 60x bus mode.

State Meaning Asserted—Indicates that a single-beat transfer is not cached, reflecting the setting of the I bit for the block or page that contains the address of the current transaction (except during certain data cache, memory synchronization, TLB management, and external control operations as described in Table 3-16).

Negated—Indicates that a burst transfer allocates an MPC7410 data cache block.

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

High Impedance—The same as A[0:31]

8.2.5 Address Transfer Termination Signals

The address transfer termination signals are used to indicate either that the address phase of the transaction has completed successfully or must be repeated, and when it must be terminated. For detailed information about how these signals interact, see Section 9.3.3, “Address Transfer Termination.”

8.2.5.1 Address Acknowledge ($\overline{\text{AACK}}$)—Input

The address acknowledge ($\overline{\text{AACK}}$) signal is an input-only signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{AACK}}$ signal.

State Meaning Asserted—Indicates that the address phase of a transaction is complete; the address bus is released to high-impedance on the next bus clock cycle.

Note that the address tenure does not terminate until the assertion of $\overline{\text{AACK}}$, even if the associated data tenure has completed. As a snooping device, the MPC7410 requires that $\overline{\text{AACK}}$ be asserted for every assertion of $\overline{\text{TS}}$ that it detects.

Negated—(During an address tenure) indicates that the address bus and the transfer attributes must remain driven.

Timing Comments Assertion—May occur as early as the bus clock cycle after $\overline{\text{TS}}$ is asserted; assertion can be delayed to allow adequate address access time for slow devices. For example, if an implementation supports slow snooping devices, an external arbiter can postpone the assertion of $\overline{\text{AACK}}$.

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

8.2.5.2 Address Retry ($\overline{\text{ARTRY}}$)

The address retry ($\overline{\text{ARTRY}}$) signal is both an input and output signal on the MPC7410.

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

Following are the state meaning and timing comments for the $\overline{\text{ARTRY}}$ output signal.

State Meaning Asserted—Indicates that the MPC7410, as a snooping device, detects a condition in which a snooped address tenure must be retried. If the MPC7410 needs to update memory as a result of the snoop that caused the retry, the MPC7410 asserts $\overline{\text{BR}}$ in the bus clock cycle following the assertion of $\overline{\text{ARTRY}}$. Note that the MPC7410 is self-snooping and may assert $\overline{\text{ARTRY}}$ for its own transaction. See Section 3.9.3, “Snooping,” for more information.

High Impedance—Indicates that the MPC7410 does not need the snooped address tenure to be retried.

Timing Comments Assertion—Asserted the second bus cycle following the assertion of $\overline{\text{TS}}$ if a retry is required.

Negation/High Impedance—Driven asserted until the bus clock cycle following the assertion of $\overline{\text{AACK}}$. Because this signal may be simultaneously driven by multiple devices, it negates in a unique fashion. First the output buffer goes to high impedance for a fraction of a bus clock cycle (dependent on the clock mode—minimum of one-half of a bus clock cycle), then it is driven negated for one bus clock cycle before returning to high impedance.

This special method of negation may be disabled by setting the precharge disable bit in HID0.

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

Following are the state meaning and timing comments for the $\overline{\text{ARTRY}}$ input signal.

State Meaning Asserted—If the MPC7410 is the address bus master, $\overline{\text{ARTRY}}$ indicates that the MPC7410 must retry the preceding address tenure and immediately negate $\overline{\text{BR}}$ (if asserted). If the associated data tenure has already started, the MPC7410 also aborts the data tenure immediately, even if data has been received.

If the MPC7410 is not the address bus master, this input indicates that the MPC7410 must immediately negate $\overline{\text{BR}}$ to allow an opportunity for a copyback operation to main memory after a snooping bus master asserts $\overline{\text{ARTRY}}$. Note that the subsequent address presented on the address bus may not be the same one associated with the assertion of the $\overline{\text{ARTRY}}$ signal.

Note that the MPC7410 ignores the $\overline{\text{BG}}$ signal on the cycle in which $\overline{\text{ARTRY}}$ is detected and the cycle following the assertion of $\overline{\text{ARTRY}}$.

Negated/High Impedance—Indicates that the MPC7410 does not need to retry the last address tenure.

Timing Comments Assertion—May occur as early as the second cycle following the assertion of $\overline{\text{TS}}$ and must occur by the bus clock cycle immediately following the assertion of $\overline{\text{AACK}}$ if an address retry is required; must remain asserted until the clock cycle following the assertion of $\overline{\text{AACK}}$.

Negation/High Impedance—Must occur two bus clock cycles after the assertion of $\overline{\text{AACK}}$.

Note that during the second bus clock cycle after the assertion of $\overline{\text{AACK}}$, masters release $\overline{\text{ARTRY}}$ to high impedance and then negate it. Thus, care must be taken when sampling $\overline{\text{ARTRY}}$ during this clock period as it could be sampled in an indeterminate state.

8.2.5.3 Shared ($\overline{\text{SHD}}$)

The shared, $\overline{\text{SHD}}$ signal is both an input and an output on the MPC7410 in 60x bus mode. In the MPX bus mode, this signal is used as the $\overline{\text{SHD0}}$ signal. The shared state is enabled with the SHDEN bit in the memory subsystem control register, MSSCR0. See Section 2.1.5.3, “Memory Subsystem Control Register (MSSCR0).”

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

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

State Meaning Asserted—If $\overline{\text{ARTRY}}$ is negated, indicates that after this transaction completes successfully, the MPC7410 will keep a valid shared copy of the address or that a reservation exists on this address. If $\overline{\text{SHD}}$ and $\overline{\text{ARTRY}}$ are asserted for a snooping master, the snoop hit modified data is pushed as the master's next address transaction.

Negated/High Impedance—After this address is transferred, the processor no longer has a valid copy of the snooped address.

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

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

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

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

State Meaning Asserted—If $\overline{\text{ARTRY}}$ is negated, the MPC7410 allocates the incoming cache block as shared (S) for a self-generated transaction. Applies only to read and read atomic transactions.

If $\overline{\text{ARTRY}}$ is asserted, $\overline{\text{SHD}}$ is ignored as an input.

Negated—If $\overline{\text{ARTRY}}$ is negated and $\overline{\text{SHD}}$ is negated, the MPC7410 allocates the incoming cache block as exclusive (E) for a self-generated read or read-atomic transaction.

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

8.2.6 Data Bus Arbitration Signals

Like the address bus arbitration signals, data bus arbitration signals maintain an orderly process for determining data bus mastership. Note that there is no data bus arbitration signal equivalent to the address bus arbitration signal $\overline{\text{BR}}$ (bus request), because, except for address-only transactions, $\overline{\text{TS}}$ implies data bus requests. For a detailed description on how these signals interact, see Section 9.4.1, "Data Bus Arbitration."

One special signal, $\overline{\text{DBWO}}$, allows the MPC7410 to be configured dynamically to write data out of order with respect to read data. For detailed information about using $\overline{\text{DBWO}}$, see Section 9.4.4, "Using Data Bus Write Only (DBWO)."

8.2.6.1 Data Bus Grant ($\overline{\text{DBG}}$)—Input

The data bus grant ($\overline{\text{DBG}}$) signal is an input-only signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{DBG}}$ signal.

State Meaning Asserted—Indicates that the MPC7410 may, with the proper qualification, assume ownership of the data bus.

QDBG = $\overline{\text{DBG}}$ & ¬($\overline{\text{ARTRY}}$ & retriable) & ¬(state_variables)

where retriable indicates whether or not the current transaction can still be retried; and state variables include whether or not:

- The data bus is being used by this master
- Whether or not the master has back-to-back burst accesses in progress

- The processor has already received the next-to-last TA for the current burst.

Thus, a qualified data bus grant occurs when:

- DBG is asserted.
- ARTRY was not asserted in the address retry window for the address phase of this transaction.
- The MPC7410 is ready to begin a data transaction.

Note that data streaming is not supported in 60x bus mode.

Negated—Indicates that the MPC7410 must hold off its data tenures.

Timing Comments Assertion—May occur any time to indicate the MPC7410 is free to take data bus mastership. It is not sampled until TS is asserted.

Negation—May occur at any time to indicate the MPC7410 cannot assume data bus mastership.

8.2.6.2 Data Bus Write Only (DBWO)—Input

The data bus write only (DBWO) signal is an input-only signal on the MPC7410. Following are the state meaning and timing comments for the DBWO signal. See Section 9.4.4, “Using Data Bus Write Only (DBWO),” for a detailed description of the use of this signal.

Note that DBWO functions as DTI0 in the MPX bus mode.

State Meaning Asserted—Indicates that the MPC7410 may run the data bus tenure for an outstanding write address even if a read address is pipelined before the write address.

Negated—Indicates that the MPC7410 must run the data bus tenures in the same order as the address tenures.

Timing Comments Assertion—Must occur no later than a qualified DBG for an outstanding write tenure. DBWO is sampled by the MPC7410 on the clock of a qualified DBG. If no write requests are pending, the MPC7410 ignores DBWO and assumes data bus mastership for the next pending read request.

Negation—May occur any time after a qualified DBG and before the next assertion of DBG.

8.2.6.3 Data Bus Busy (DBB)—Output

The data bus busy (DBB) signal is strictly an output signal on the MPC7410. See Section 9.4.1.2, “Using the DBB Signal,” for more information. Following are the state meaning and timing comments for DBB.

State Meaning Asserted—Indicates that the MPC7410 is the data bus master. The MPC7410 always assumes data bus mastership if it needs the data bus and is given a qualified data bus grant (see DBG).

Negated—Indicates that the MPC7410 is not using the data bus.

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

Negation—Occurs for a minimum of one-half bus clock cycle (dependent on clock mode) following the assertion of the final $\overline{\text{TA}}$.

High Impedance—Occurs after $\overline{\text{DBB}}$ is negated.

8.2.7 Data Transfer Signals

Like the address transfer signals, the data transfer signals are used to transmit data and to generate and monitor parity for the data transfer. For a detailed description of how the data transfer signals interact, see Section 9.4.2, “Data Transfer Signals and Protocol.”

8.2.7.1 Data Bus (DH[0:31], DL[0:31])

The data bus (DH[0:31] and DL[0:31]) consists of 64 signals that are both inputs and outputs on the MPC7410. The data bus is driven once for single-beat transactions and four times for burst transactions. See Table 8-4 for the data bus lane assignments.

Table 8-4. Data Bus Lane Assignments

Data Bus Signals	Byte Lane
DH[0:7]	0
DH[8:15]	1
DH[16:23]	2
DH[24:31]	3
DL[0:7]	4
DL[8:15]	5
DL[16:23]	6
DL[24:31]	7

8.2.7.1.1 Data Bus (DH[0:31], DL[0:31])—Output

Following are the state meaning and timing comments for DH[0:31] and DL[0:31] as output signals.

State Meaning Asserted/Negated—Represent the state of data during a data write. Byte lanes not selected for data transfer do not supply valid data.

Timing Comments Assertion/Negation—Initial beat coincides with $\overline{\text{DBB}}$ and, for bursts, transitions on the bus clock cycle following each assertion of $\overline{\text{TA}}$.

High Impedance—Occurs on the bus clock cycle after the final assertion of $\overline{\text{TA}}$, following the assertion of $\overline{\text{TEA}}$, or in certain $\overline{\text{ARTRY}}$ cases.

8.2.7.1.2 Data Bus (DH[0:31], DL[0:31])—Input

Following are the state meaning and timing comments for DH[0:31] and DL[0:31] as input signals.

State Meaning Asserted/Negated—Represent the state of data during a data read transaction.

Timing Comments Assertion/Negation—Data must be valid on the same bus clock cycle that $\overline{\text{TA}}$ is asserted.

8.2.7.2 Data Bus Parity (DP[0:7])

The eight data bus parity (DP[0:7]) signals on the MPC7410 are both output and input.

8.2.7.2.1 Data Bus Parity (DP[0:7])—Output

Following are the state meaning and timing comments for DP[0:7] as output signals.

State Meaning Asserted/Negated—Represents odd parity for each of the eight bytes during data write transactions. Odd parity means that an odd number of bits, including the parity bit, are driven high. The generation of parity is enabled through HID0. The signal assignments are listed in Table 8-5.

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

Table 8-5. DP[0:7] Signal Assignments

Signal Name	Signal Assignments
DP0	DH[0:7]
DP1	DH[8:15]
DP2	DH[16:23]
DP3	DH[24:31]
DP4	DL[0:7]
DP5	DL[8:15]
DP6	DL[16:23]
DP7	DL[24:31]

8.2.7.2.2 Data Bus Parity (DP[0:7])—Input

Following are the state meaning and timing comments for DP[0:7] as input signals.

State Meaning Asserted/Negated—Represents odd parity for each byte of read data. Parity is checked on all data byte lanes, regardless of the size of the transfer. Detected even parity causes a checkstop if data parity errors are enabled in the HID0 register.

Timing Comments Assertion/Negation—The same as DL[0:31]

8.2.8 Data Transfer Termination Signals

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

For a detailed description of how these signals interact, see Section 9.4.3, “Data Transfer Termination.”

8.2.8.1 Transfer Acknowledge ($\overline{\mathrm{TA}}$)—Input

Following are the state meaning and timing comments for the $\overline{\mathrm{TA}}$ signal.

State Meaning

Asserted—Indicates that a single-beat data transfer or a data beat in a burst transfer completed successfully. Note that $\overline{\mathrm{TA}}$ must be asserted for each data beat in a burst transaction.

Negated—(During a data tenure) indicates that, until $\overline{\mathrm{TA}}$ is asserted, the MPC7410 must continue to drive the data for the current write or must wait to sample the data for reads.

Timing Comments

Assertion—Must not occur before $\overline{\mathrm{ARTRY}}$ for the current transaction (if the address retry mechanism is to be used to prevent invalid data from being used by the processor); otherwise, assertion may occur at any time during a data tenure. The system can withhold assertion of $\overline{\mathrm{TA}}$ to indicate that the MPC7410 should insert wait states to extend the duration of the data beat.

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

8.2.8.2 Transfer Error Acknowledge ($\overline{\mathrm{TEA}}$)—Input

Following are the state meaning and timing comments for the $\overline{\mathrm{TEA}}$ signal.

State Meaning

Asserted—Indicates that a bus error occurred. Causes a machine check exception (and possibly causes the processor to enter checkstop state if machine check enable bit is cleared (MSR[ME] = 0)). For more information, see Section 4.6.2.2, "Checkstop State (MSR[ME] = 0)." Assertion terminates the current transaction; that is, assertion of $\overline{\mathrm{TA}}$ is ignored. The assertion of $\overline{\mathrm{TEA}}$ causes the negation/high impedance of $\overline{\mathrm{DBB}}$ in the next clock cycle. However, data entering the GPR or the cache is not invalidated.

Negated—Indicates that no bus error was detected.

Timing Comments

Assertion—May be asserted while $\overline{\mathrm{DBB}}$ is asserted, up to and including the cycle of the final $\overline{\mathrm{TA}}$. $\overline{\mathrm{TEA}}$ should be asserted for one cycle only.

Negation—$\overline{\mathrm{TEA}}$ must be negated one cycle after it is asserted.

8.3 60x/MPX Bus Protocol Signal Compatibility

The MPX bus mode protocol defines several new signals not present in the 60x bus protocol. Also, there are 60x signals not supported by the MPC7410. These signal differences are summarized in Table 8-6. Note that a few 60x signals have expanded or modified functionality in the MPX bus mode.

Table 8-6. Signal Compatibility Summary

60x Bus Signals not in MPC7410	60x Bus Signals Multiplexed with new MPX Bus Mode Signals	New MPX Bus Mode Signals
Address bus busy (as input) $\overline{\text{ABB}}$in Data bus busy (as input) $\overline{\text{DBB}}$in Data retry $\overline{\text{DRTRY}}$ Extended transfer protocol $\overline{\text{XATS}}$ Transfer code TC[0:1] Cache set element CSE[0:1] Address parity error $\overline{\text{APE}}$ Data parity error $\overline{\text{DPE}}$	Data bus write only $\overline{\text{DBWO}}$⇒ Data transfer index DTI[0:2] (Shared) $\overline{\text{SHD}}$ ⇒ $\overline{\text{SHD0}}$ Address bus busy (as output) $\overline{\text{ABB}}$⇒AMON Data bus busy (as output) $\overline{\text{DBB}}$⇒DMON	Hit $\overline{\text{HIT}}$ Data ready $\overline{\text{DRDY}}$ (Shared) $\overline{\text{SHD1}}$

The three types of signals in Table 8-6 (shown in the column headings) are described in the following three sections.

8.3.1 60x Bus Signals Not in the MPC7410

Several signals defined in the 60x bus protocol are not implemented in the MPC7410; however, new signals provide similar functionality for compatibility reasons.

8.3.1.1 Address Bus Busy and Data Bus Busy (ABB and DBB)

The MPC7410 does not use the $\overline{\text{ABB}}$ or $\overline{\text{DBB}}$ signals as inputs. The MPC7410 tracks its own outstanding transactions and relies on the system arbiter to provide grants for the address and data buses only when the bus is available and the grant may be accepted.

8.3.1.2 Data Retry ($\overline{\text{DRTRY}}$)

The data retry input signal is not implemented on the MPC7410. Only the no-$\overline{\text{DRTRY}}$ mode defined in the 60x bus protocol is supported.

8.3.1.3 Extended Transfer Protocol ($\overline{\text{XATS}}$)

The extended transfer protocol signal, used for accesses to direct-store segments, is not supported by the MPC7410 processor interface.

8.3.1.4 Transfer Code (TC[0:1])

The transfer code signals are not implemented on the MPC7410. Other processors that implement the 60x bus provided an indication of whether a read access was instruction or data by the encoding of these signals. This information is now provided on TT0 (driven high for instruction fetches if HID0[IFFT] = 1).

8.3.1.5 Cache Set Element (CSE[0:1])

These signals are not implemented in the MPC7410 as the MPC7410 does not support snoop-filtering devices.

8.3.1.6 Address Parity Error and Data Parity Error (APE, DPE)

The address parity and data parity error signals are not implemented in the MPC7410.

8.3.2 60x Signals Multiplexed with New MPX Bus Mode Signals

The DBWO signal is implemented similarly on the MPC7410 as in the MPC750. In MPX bus mode, this signal is multiplexed with the new DTI0 signal, and together with the new DTI[1:2] signals, implements more extensive data reordering functionality. See Section 8.4.6.2, “Data Transaction Index (DTI[0:2])—Input.”

The SHD signal is implemented similarly on the MPC7410 as in the MPC604e. In MPX bus mode, this signal is multiplexed with the new SHD0 signal, and together with the new SHD1 signal, provides the cache coherency shared indication in a multiprocessor system. See Section 8.2.5.3, “Shared (SHD).”

As described in Section 8.3.1.1, the ABB and DBB signals are implemented only as outputs on the MPC7410 in 60x bus mode. In MPX bus mode, these signals are multiplexed with the new AMON and DMON signals that provide essentially the same functionality as the ABB and DBB outputs. However, these signals are strictly optional and may not be implemented in subsequent products that support the MPX bus protocol.

8.3.3 New MPX Bus Mode Signals

The MPX bus mode’s support for data intervention and full data streaming for burst reads and writes is realized through the addition of two new signals—HIT and DRDY. See Section 9.6.2, “Data Tenure in MPX Bus Mode,” for a complete description of this functionality.

The HIT signal is a point-to-point signal output from the processor or local bus slave to the system arbiter. This signal is a snoop response valid in the address retry (ARTRY) window (the cycle after an address acknowledge (AACK) that indicates that the MPC7410 will supply intervention data. That is, the MPC7410 has found the data in its L1 or L2 cache that has been requested by another master’s bus transaction. Instead of asserting ARTRY and flushing the data to memory, the MPC7410 may assert HIT to indicate that it can supply the data directly to the other master. This functionality is enabled separately for the L1 and L2 caches by fields in the MSSCR0 register.

The DRDY signal is also used by the MPX bus protocol to implement data intervention in the case of a cache hit. See Section 8.4.6.3, “Data Ready (DRDY)—Output.”

The SHD1 signal operates in conjunction with the SHD0 signal to indicate that a cached item is shared. See Section 8.4.5.3, “MPX Bus Shared (SHD0, SHD1) Signals.”

8.4 MPX Bus Signal Configuration

The MPC7410 has a new bus interface that is derived from the 60x bus. This new interface, the MPX bus, includes several additional features that provide higher memory bandwidth than the 60x bus and more efficient use of the system bus in a multiprocessing environment.

The value of the $\overline{\text{EMODE}}$ signal at the negation of $\overline{\text{HRESET}}$ determines whether the MPC7410 operates with the 60x bus or the MPX bus. This value is stored in and readable from the EMODE bit in MSSCR0. The state of MSSR0[EMODE] is active high, meaning that if $\overline{\text{EMODE}}$ is detected as asserted at the negation of $\overline{\text{HRESET}}$, MSSR0[EMODE] = 1 and MPX bus mode is selected; otherwise, MSSR0[EMODE] = 0 and 60x bus mode is selected.

8.4.1 MPX Bus Mode Functional Groupings

Figure 8-2 illustrates the MPC7410’s signal configuration, showing how the signals are grouped in MPX bus mode. A pinout showing pin numbers is included in the MPC7410 hardware specifications. See Section 9.6, “MPX Bus Protocol,” for a complete functional description of the MPX bus protocol.

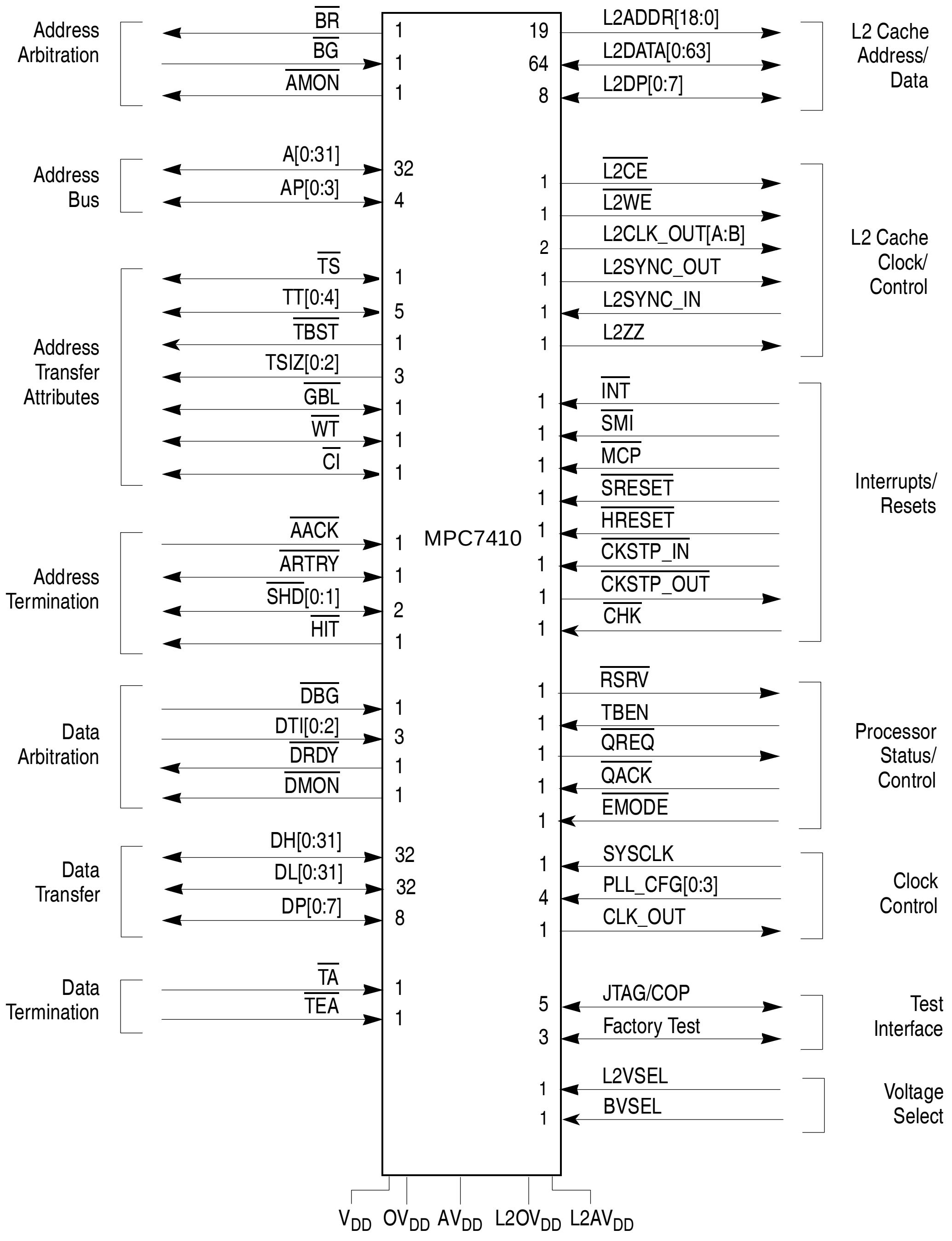


Figure 8-2. MPX Bus Signal Groups

8.4.2 MPX Address Bus Arbitration Signals

The address arbitration signals are the input and output signals the MPC7410 uses to request the address bus, recognize when the request is granted, and indicate to other devices when mastership is granted. For a detailed description of how these signals interact, see Section 9.3.1, “Address Bus Arbitration.”

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

Following are the state meaning and timing comments for the $\overline{\text{BR}}$ output signal on the MPC7410 in MPX bus mode.

State Meaning Asserted—Same as 60x bus interface

Negated—Same as 60x bus interface

Timing Comments Assertion—Same as 60x bus interface

Negation—Note that $\overline{\text{BR}}$ is negated during the cycle in which the processor is asserting $\overline{\text{TS}}$ unless the processor has another pending transaction to perform in MPX bus mode.

High Impedance—Same as 60x bus interface

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

Following are the state meaning and timing comments for the $\overline{\text{BG}}$ output signal on the MPC7410 in MPX bus mode.

State Meaning Asserted—Indicates that the MPC7410 may, with the proper qualification, begin a bus transaction. A qualified bus grant is determined from the bus state as follows:

QBG = $\overline{\text{BG}}$ • ¬$\overline{\text{ARTRY}}$ • ¬$\overline{\text{TS}}$ • ¬(latched state variables)

Negated—Indicates that the MPC7410 is not granted next address bus ownership.

Timing Comments Assertion—May occur on any cycle.

Negation—May occur whenever the MPC7410 must be prevented from starting a bus transaction. The MPC7410 may still assume address bus ownership on the cycle $\overline{\text{BG}}$ is negated if $\overline{\text{BG}}$ was asserted the previous cycle with other bus grant qualifications. Negation must occur in every cycle the arbiter delays $\overline{\text{AACK}}$. Since $\overline{\text{AACK}}$ is not in the qualified bus grant equation and $\overline{\text{ABB}}$ is not generated by the MPC7410 in MPX bus mode, the bus arbiter must negate $\overline{\text{BG}}$ in every cycle the arbiter is delaying $\overline{\text{AACK}}$ in order to prevent a qualified bus grant.

8.4.2.3 Address Bus Monitor ($\overline{\text{AMON}}$)—Output

The address bus monitor ($\overline{\text{AMON}}$) signal is strictly optional in the MPX bus protocol. Following are the state meaning and timing comments for $\overline{\text{AMON}}$.

State Meaning Asserted—Same as 60x bus interface $\overline{\text{ABB}}$ signal

Negated—Same as 60x bus interface $\overline{\text{ABB}}$ signal

Timing Comments Assertion—Same as 60x bus interface $\overline{\text{ABB}}$ signal

Negation—Same as 60x bus interface $\overline{\text{ABB}}$ signal

High Impedance—Same as 60x bus interface $\overline{\text{ABB}}$ signal

8.4.3 Address Bus and Parity in MPX Bus Mode

The following sections describe the address bus and parity signals in MPX bus mode. The address bus driven mode is enabled with the assertion of $\overline{\text{EMODE}}$ after $\overline{\text{HRESET}}$ negation. Note that this selection is reflected in the read-only ABD bit in the memory subsystem control register, MSSCR0. See Section 2.1.5.3, "Memory Subsystem Control Register (MSSCR0)."

8.4.3.1 Address Bus (A[0:31])—Output

Following are the state meaning and timing comments for the address bus A[0:31] output signals on the MPC7410 in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as 60x bus interface

High Impedance—Occurs one bus clock cycle following the assertion of $\overline{\text{AACK}}$ unless address bus streaming is occurring and the MPC7410 qualified a $\overline{\text{BG}}$ on the previous cycle.

Note that if MSSCR0[ABD] is set and MSSCR0[EMODE] is set, the address bus is always driven on the bus clock cycle after $\overline{\text{BG}}$ is asserted to the processor, regardless of whether the MPC7410 has a queued transaction.

8.4.3.2 Address Bus (A[0:31])—Input

Following are the state meaning and timing comments for the address bus A[0:31] input signals on the MPC7410 in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as 60x bus interface

High Impedance—Occurs on the bus clock cycle after the assertion of $\overline{\text{AACK}}$ unless address bus streaming is occurring and the MPC7410 qualified a $\overline{\text{BG}}$ on the previous cycle.

8.4.3.3 Address Parity (AP[0:3])—Output

Following are the state meaning and timing comments for the AP[0:3] output signals on the MPC7410.

State Meaning Asserted/Negated—Same as A[0:31]

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.3.4 Address Parity (AP[0:3])—Input

Following are the state meaning and timing comments for the AP[0:3] input signals on the MPC7410.

State Meaning Asserted/Negated—Same as A[0:31]

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4 Address Transfer Attribute Signals in MPX Bus Mode

The transfer attribute signal functions in MPX bus mode are very similar to that of 60x bus mode, with the exceptions noted in the following subsections.

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

Following are the state meaning and timing comments for the transfer start $\overline{\text{TS}}$ output signal on the MPC7410.

State Meaning Asserted—Same as 60x bus interface

Negated—Same as 60x bus interface

Timing Comments Assertion—Same as 60x bus interface

Negation—Same as 60x bus interface

High Impedance—Occurs two bus clock cycles after $\overline{\text{TS}}$ is asserted, unless address bus streaming is occurring and the MPC7410 qualified a $\overline{\text{BG}}$ on the previous cycle.

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

Following are the state meaning and timing comments for the transfer start $\overline{\text{TS}}$ input signal on the MPC7410.

State Meaning Asserted—Same as 60x bus interface

Negated—Same as 60x bus interface

Timing Comments Assertion—Can occur on any bus clock cycle following a qualified $\overline{\text{BG}}$ that is accepted by the processor.

Negation— Must occur one bus clock cycle after assertion.

8.4.4.3 Transfer Type (TT[0:4])

The transfer type (TT[0:4]) signals consist of five input/output signals on the MPC7410.

8.4.4.3.1 Transfer Type (TT[0:4])—Output

Following are the state meaning and timing comments for the transfer type TT[0:4] output signals on the MPC7410 in MPX bus mode. Note that there is a new transfer type called read claim (RCLAIM; TT[0:4] = 0b0111) defined for MPX bus mode that is used for accesses generated by touch-for-store instructions.

State Meaning Asserted/Negated–Same as 60x bus interface except for RCLAIM as defined in Section 9.6.1.3.1, “Transfer Type 0–4 (TT[0:4]) in MPX Bus Mode.”

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4.3.2 Transfer Type (TT[0:4])—Input

Following are the state meaning and timing comments for the transfer type TT[0:4] input signals on the MPC7410 in MPX bus mode.

State Meaning Asserted/Negated–Same as 60x bus interface except for RCLAIM as defined in Section 9.6.1.3.1, “Transfer Type 0–4 (TT[0:4]) in MPX Bus Mode.”

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4.4 Transfer Burst (TBST)—Output

The transfer burst (TBST) signal is an output signal on the MPC7410.

Following are the state meaning and timing comments for the transfer burst TBST output signal on the MPC7410 in MPX bus mode.

State Meaning Asserted—Same as 60x bus interface

Negated—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as A[0:31]

High Impedance—Same as A[0:31]

8.4.4.5 Transfer Size (TSIZ[0:2])—Output

Following are the state meaning and timing comments for the transfer size TSIZ[0:2] output signals on the MPC7410 in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as A[0:31]

High Impedance—Same as A[0:31]

8.4.4.6 Global (GBL)

The global (GBL) signal is an input/output signal on the MPC7410.

8.4.4.6.1 Global (GBL)—Output

Following are the state meaning and timing comments for the global GBL output signal on the MPC7410 in MPX bus mode.

State Meaning Asserted—Same as 60x bus interface

Negated—Same as 60x bus interface.

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4.6.2 Global (GBL)—Input

Following are the state meaning and timing comments for the global GBL input signal on the MPC7410 in MPX bus mode.

State Meaning Asserted—Same as 60x bus interface

Negated—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4.7 Write-Through ($\overline{\text{WT}}$)

The $\overline{\text{WT}}$ signal is both an input and output signal on the MPC7410 in MPX bus mode (note that it is output-only in 60x bus mode).

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

Following are the state meaning and timing comments for the write-through $\overline{\text{WT}}$ output signal on the MPC7410 in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4.7.2 Write-Through ($\overline{\text{WT}}$)—Input

Following are the state meaning and timing comments for the write-through $\overline{\text{WT}}$ signal as an input on the MPC7410 in MPX bus mode.

State Meaning Asserted—Indicates that the MPC7410 should not assert $\overline{\text{HIT}}$ to provide data intervention in response to a snoop because data intervention is not allowed for write-through accesses.

Negated—Indicates that the MPC7410 may assert $\overline{\text{HIT}}$ to provide data intervention in response to a snoop, provided $\overline{\text{CI}}$ is not asserted.

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4.8 Cache Inhibit ($\overline{\text{CI}}$)

The $\overline{\text{CI}}$ signal is both an input and output signal on the MPC7410 in MPX bus mode (note that it is output-only in 60x bus mode).

8.4.4.8.1 Cache Inhibit ($\overline{\text{CI}}$)—Output

The cache inhibit ($\overline{\text{CI}}$) signal is an output signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{CI}}$ signal in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as A[0:31]

8.4.4.8.2 Cache Inhibit ($\overline{\text{CI}}$)—Input

The cache inhibit ($\overline{\text{CI}}$) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{CI}}$ signal as an input in MPX bus mode.

State Meaning Asserted—Indicates that the MPC7410 should not assert $\overline{\text{HIT}}$ to provide data intervention in response to a snoop because data intervention is not allowed for cache-inhibited accesses.

Negated—Indicates that the MPC7410 may assert HIT to provide data intervention in response to a snoop, provided WT is not asserted

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

8.4.5 MPX Address Transfer Termination Signals

The address transfer termination signal functions in MPX bus mode are very similar to that of 60x bus mode, with the exceptions noted in the following subsections. For detailed information about how these signals interact, see Section 9.6.1.4, “Address Termination Phase in MPX Bus Mode.”

8.4.5.1 Address Acknowledge (AACK)—Input

The address acknowledge (AACK) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the AACK signal in MPX bus mode.

State Meaning Asserted—The same as 60x bus interface except the MPC7410, as address bus master, does not release the address and transfer attribute signals to high impedance in response to AACK when the following conditions are met:

- Address bus driven mode is enabled (MSSCR0[ABD] = 1) and there was a BG to the MPC7410 on the previous clock cycle.
- Address bus streaming is occurring and the MPC7410 qualified a BG in the previous clock cycle.

Negated—Same as 60x bus interface

Timing Comments Assertion—The same as 60x bus interface except that because AACK is not in the qualified bus grant equation and a synthesized ABB is not generated in MPX bus mode, the bus arbiter must negate BG in every cycle the arbiter is delaying AACK to prevent a qualified bus grant in those cases.

Negation—Same as 60x bus interface

8.4.5.2 Address Retry (ARTRY)

The address retry (ARTRY) signal is both an input and output signal on the MPC7410 in MPX bus mode.

8.4.5.2.1 Address Retry (ARTRY)—Output

The address retry (ARTRY) signal is an output signal on the MPC7410. Following are the state meaning and timing comments for the ARTRY signal in MPX bus mode.

State Meaning Asserted—Same as 60x bus interface

Negation/High Impedance—Same as 60x bus interface

Timing Comments Assertion—Same as 60x bus interface

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

The address retry ($\overline{\text{ARTRY}}$) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{ARTRY}}$ signal in MPX bus mode.

State Meaning Asserted—The same as 60x bus interface except that if address bus streaming is occurring and a $\overline{\text{TS}}$ from this MPC7410 coincides with the bus cycle of the $\overline{\text{ARTRY}}$ input, the MPC7410 also aborts subsequent transactions that may have already begun as an additional response to the assertion of $\overline{\text{ARTRY}}$.

Negated—Same as 60x bus interface

Timing Comments Assertion—Same as 60x bus interface

Negation/High Impedance—Same as 60x bus interface

8.4.5.3 MPX Bus Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$) Signals

The $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ signals act together to indicate a shared snoop response. In 60x bus mode, the $\overline{\text{SHD0}}$ signal is used as the $\overline{\text{SHD}}$ signal, analogous to the $\overline{\text{SHD}}$ signal of the MPC604e. The MPX bus mode interface allows a given master to drive a new address tenure every other cycle, so the shared signal must be able to be driven every other cycle. But, because it must be actively negated and might be driven by multiple masters at any given time, in MPX mode, electrical requirements dictate that two versions of the $\overline{\text{SHD}}$ signal be implemented. When signaling a snoop response of shared, the MPC7410 must assert $\overline{\text{SHD0}}$ unless $\overline{\text{SHD0}}$ was asserted in any of the three cycles prior to the snoop response window for the current transaction. In that case, the MPC7410 asserts $\overline{\text{SHD1}}$. Thus, each of $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ can be released to high-impedance, driven negated, then released to high-impedance again before it needs to be reasserted. When the MPC7410 is a bus master, the MPC7410 considers the snoop response to be shared if either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ is asserted.

In MEI mode (MSSCR[SHDEN] = 0), the shared signals are enabled with MSSCR0[SHDPEN3]. In MESI or MERSI mode (MSSCR[SHDEN] = 1), the SHDPEN3 bit in MSSCR0 is ignored. See Section 2.1.5.3, “Memory Subsystem Control Register (MSSCR0).”

8.4.5.3.1 Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$)—Output

If $\overline{\text{SHD0}}$ was asserted in any of the three cycles before the snoop response window for the current transaction, then $\overline{\text{SHD1}}$ is used to indicate a shared response in this cycle. Following are the state meaning and timing comments for the $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ output signals.

State Meaning Asserted—If $\overline{\text{ARTRY}}$ is not asserted, it indicates that the MPC7410 had a cache hit on a shared block or the reservation address.

If $\overline{\text{ARTRY}}$ is asserted, a snoop push of modified data is required.

Negated/High Impedance—Indicates that the processor did not contain the data or has invalidated the snooped address.

Timing Comments Assertion/Negation—Same as $\overline{\text{SHD}}$ in 60x bus interface (same as $\overline{\text{ARTRY}}$).

High Impedance—Same as $\overline{\text{SHD}}$ in 60x bus interface (same as $\overline{\text{ARTRY}}$).

8.4.5.3.2 Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$)—Input

Following are the state meaning and timing comments for the $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ input signals.

State Meaning Asserted—Same as $\overline{\text{SHD}}$ in 60x bus interface.

Negated—Same as $\overline{\text{SHD}}$ in 60x bus interface.

Timing Comments Assertion/Negation—Same as $\overline{\text{SHD}}$ in 60x bus interface (same as $\overline{\text{ARTRY}}$).

8.4.5.4 Snoop Hit ($\overline{\text{HIT}}$)—Output

The snoop response in MPX mode of the MPC7410 uses the $\overline{\text{HIT}}$ output signal to communicate to the system whether or not data intervention occurs for the current transaction. See Section 9.6.1, “Address Tenure in MPX Bus Mode,” and Section 9.6.2, “Data Tenure in MPX Bus Mode,” for more detailed information about the data-only transactions used by the MPC7410 in MPX bus mode for data intervention.

Additionally, if the MPC7410 intervenes with shared or exclusive data rather than modified data, the $\overline{\text{HIT}}$ signal is asserted for a second cycle after $\overline{\text{AACK}}$. This second $\overline{\text{HIT}}$ cycle signals to the memory controller that the copy of data in memory is up-to-date, and snarfing is not required. (Snarfing is when a device provides data specifically for another device and a third device reads the data also). L1 and L2 data cache hit intervention (and the assertion of $\overline{\text{HIT}}$) is enabled individually with the L1_INTVEN and L2_INTVEN bits in the memory subsystem control register, MSSCR0. See Section 2.1.5.3, “Memory Subsystem Control Register (MSSCR0).”

It is possible for the MPC7410 to assert both $\overline{\text{ARTRY}}$ and $\overline{\text{HIT}}$ simultaneously for the same snoop response. When simultaneously asserted, $\overline{\text{ARTRY}}$ supersedes $\overline{\text{HIT}}$ and $\overline{\text{HIT}}$ should be ignored by the system.

Following are the state meaning and timing comments for the $\overline{\text{HIT}}$ signal.

State Meaning Asserted—The MPC7410 has the requested data in its cache and will supply it through a data-only transaction. $\overline{\text{HIT}}$ is asserted for a second cycle if the snoop data does not need to be forwarded to memory because it was not modified.

Negated—The MPC7410 cannot provide data for a snoop request through the $\overline{\text{HIT}}$ intervention protocol.

Timing Comments Asserted—Like other snoop responses, $\overline{\text{HIT}}$ can be driven as soon as the second cycle after $\overline{\text{TS}}$. If $\overline{\text{AACK}}$ is delayed, the response needs to be held until the cycle after $\overline{\text{AACK}}$ (for example, $\overline{\text{HIT}}$ is asserted with the same timing as ARTRY).

$\overline{\text{HIT}}$ is held asserted for one cycle beyond the assertion of $\overline{\text{AACK}}$ if the snoop hit data is modified and must be forwarded to memory. $\overline{\text{HIT}}$ is asserted for two cycles beyond the assertion of $\overline{\text{AACK}}$ if the snoop hit data is not modified and does not need to be forwarded to memory.

Negated—$\overline{\text{HIT}}$ is negated the cycle after the appropriate response window (one cycle after $\overline{\text{AACK}}$ for modified data and two cycles after $\overline{\text{AACK}}$ for non-modified data) unless it must be asserted again immediately for another snoop response.

8.4.6 Data Bus Arbitration Signals

The data bus arbitration signals for MPX bus mode operate similarly to 60x bus mode except as noted in the following subsections. See Section 9.6.2.1, “Data Bus Arbitration Phase in MPX Bus Mode,” for more information about data bus arbitration in MPX bus mode.

8.4.6.1 Data Bus Grant ($\overline{\text{DBG}}$)—Input

The data bus grant ($\overline{\text{DBG}}$) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{DBG}}$ signal in MPX bus mode.

State Meaning Asserted—Same as 60x bus interface, except that data streaming is allowed in MPX bus mode.

Negated—Same as 60x bus interface

Timing Comments Assertion—Same as 60x bus interface

Negation—Same as 60x bus interface

8.4.6.2 Data Transaction Index (DTI[0:2])—Input

The 60x bus transaction reordering scheme is implemented with the $\overline{\text{DBWO}}$ signal. The MPX bus mode can be configured to support a generalized reordering scheme using the new 3-bit data transfer index (DTI[0:2]) input signals.

The DTI signals can be bused or point-to-point. They must be driven valid by the system arbiter on the cycle before a data bus grant ($\overline{\text{DBG}}$). They are sampled on each bus clock cycle by the MPC7410 and are qualified by the assertion of $\overline{\text{DBG}}$ on the following cycle.

The data transfer index is a pointer into the MPC7410’s queue of outstanding transactions, indicating which transaction is to be serviced by the subsequent data tenure. Note that this protocol is a generalization of the $\overline{\text{DBWO}}$ protocol in which the assertion of $\overline{\text{DBWO}}$ indicated that the first write operation in the queue was to be serviced. For example, DTI = 0b000 means that the oldest transaction is to be serviced, DTI = 0b001 means the second oldest transaction is to be serviced up to DTI = 0b101 meaning the 6th oldest transaction is to be serviced. Note that because the MPC7410 only supports six outstanding data transactions, the maximum setting for DTI is 0b101.

Data tenure reordering can be disabled by setting DTI[0:2] to 0b000. This setting causes the MPC7410 to select always the oldest transaction in the outstanding transaction queue. See Section 9.6.2.2.8, “Data Tenure Reordering in MPX Bus Only.”

Following are the state meaning and timing comments for the DTI[0:2] signals.

State Meaning Asserted—The DTI[0:2] signals act as a pointer into the queue of outstanding transactions for the MPC7410, indicating which transaction is to be served by the subsequent data tenure. For example, DTI = 0b000 means that the oldest transaction is to be serviced, DTI = 0b001 means the second oldest transaction is to be serviced up to DTI = 0b101 meaning the 6th oldest transaction is to be serviced.

Negated—DTI = 0b000 indicates that the MPC7410 must run the data bus tenures in the same order as the address tenures

Timing Comments Assertion/Negation—Sampled each cycle and qualified by a qualified DBG in the following cycle.

8.4.6.3 Data Ready (DRDY)—Output

The data ready (DRDY) signal is an output signal on the MPC7410 used in conjunction with HIT to perform data intervention in MPX bus mode. Note that the L1_INTVEN and L2_INTVEN fields of MSSCR0 control the way that the MPC7410 uses data intervention for the L1 and L2 caches. See Section 2.1.5.3, “Memory Subsystem Control Register (MSSCR0).” Also, see Section 9.6.2, “Data Tenure in MPX Bus Mode,” for more information about the data intervention functionality. Following are the state meaning and timing comments for the DRDY signal.

State Meaning Asserted—The MPC7410 has data ready for a pending bus operation initiated elsewhere in the system (for which the MPC7410 has previously signaled HIT during the snoop response window), and the MPC7410 is requesting the data bus in order to service that bus operation.

Negated—The MPC7410 is not requesting the data bus to service an outstanding bus request.

Timing Comments Asserted—DRDY is asserted no earlier than HIT and no earlier than two cycles before the MPC7410 is able to drive the data (since DRDY may be followed immediately by DBG and then TA).

Negated—DRDY is negated on the cycle after it is asserted unless another DRDY is asserted for the next transaction. DRDY may be fully pipelined on back-to-back cycles when multiple hits are outstanding.

8.4.6.4 Data Bus Monitor (DMON)—Output

The data bus monitor (DMON) signal is strictly optional in the MPX bus protocol. Following are the state meaning and timing comments for DMON.

State Meaning Asserted—Same as 60x bus interface DBB signal

Negated—Same as 60x bus interface DBB signal

Timing Comments Assertion—Same as 60x bus interface DBB signal

Negation—Same as 60x bus interface DBB signal

High Impedance—Same as 60x bus interface DBB signal

8.4.7 Data Transfer Signals in MPX Bus Mode

The data transfer signals in MPX bus mode transmit data and generate and monitor parity for the data transfer similarly to that in 60x bus mode, except that they are also used for data-only transactions. For a detailed description of how the data transfer signals interact in MPX bus mode, see Section 9.6.2, “Data Tenure in MPX Bus Mode.”

8.4.7.1 Data Bus (DH[0:31], DL[0:31])

The following subsections describe the operation of the data bus signals as inputs and outputs in MPX bus mode.

8.4.7.1.1 Data Bus (DH[0:31], DL[0:31])—Output

Following are the state meaning and timing comments for the DH[0:31], DL[0:31] signals as outputs in MPX bus mode.

State Meaning Asserted/Negated—Represent the state of data during a data write transaction or a data-only (data intervention) transaction. Byte lanes not selected for data transfer do not supply valid data.

Timing Comments Assertion/Negation—Initial beat occurs one bus clock cycle after a qualified $\overline{\text{DBG}}$ is sampled, and, for bursts, transitions on the bus in the clock cycle following each assertion of $\overline{\text{TA}}$.

High Impedance—Same as 60x bus interface

8.4.7.1.2 Data Bus (DH[0:31], DL[0:31])—Input

Following are the state meaning and timing comments for the DH[0:31], DL[0:31] signals as inputs in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface, except that these signals are also used for data-only transactions in MPX bus mode.

Timing Comments Assertion/Negation—Same as 60x bus interface

8.4.7.2 Data Bus Parity (DP[0:7])—Output

Following are the state meaning and timing comments for the DP[0:7] signals as outputs in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface, except that they are also driven for data-only transactions in MPX bus mode.

High Impedance—Same as 60x bus interface

Timing Comments Assertion/Negation—Same as DH[0:31], DL[0:31]

High Impedance—Same as DH[0:31], DL[0:31]

8.4.7.3 Data Bus Parity (DP[0:7])—Input

Following are the state meaning and timing comments for the DP[0:7] signals as inputs in MPX bus mode.

State Meaning Asserted/Negated—Same as 60x bus interface., except that these signals are also used for data-only transactions in MPX bus mode.

Timing Comments Assertion/Negation—Same as DH[0:31], DL[0:31]

8.4.8 Data Transfer Termination Signals in MPX Bus Mode

The function of the data termination signals in MPX bus mode is similar to that in 60x bus mode. The differences are described in the following subsections. For a detailed description of how these signals interact in MPX bus mode, see Section 9.6.2.3, “Data Termination Phase in MPX Bus Mode.”

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

Following are the state meaning and timing comments for the $\overline{\text{TA}}$ signal.

State Meaning Asserted—Same as 60x bus interface

Negated—Same as 60x bus interface

Timing Comments Assertion—Same as 60x bus interface

Negation—Same as 60x bus interface

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

Following are the state meaning and timing comments for the $\overline{\text{TEA}}$ signal.

State Meaning Asserted—The same as the 60x bus interface except for the comment about the assertion of $\overline{\text{TA}}$ causing $\overline{\text{DBB}}$ to negate (because the MPX bus mode does not use $\overline{\text{DBB}}$, although similar functionality is provided by the DMON signal in MPX bus mode).

Negated—Same as 60x bus interface

Timing Comments Assertion—May be asserted on any bus clock cycle during a normal data tenure, from the cycle following a qualified data bus grant to the cycle of the final $\overline{\text{TA}}$.

Negation—Same as 60x bus interface

8.5 Non-Protocol Signal Descriptions

The following sections describe the signals on the MPC7410 that do not specifically implement the 60x or MPX bus protocols. These signals include the L2 interface signals, the interrupt and reset signals, processor status and control signals, clock control signals, and JTAG test signals.

8.5.1 L2 Cache Address/Data

The MPC7410’s dedicated L2 cache interface provides all the signals required for the support of up to 2 Mbytes of synchronous SRAM for data storage. The use of the L2 data parity (L2DP[0:7]) and L2 low-power mode enable (L2ZZ) signals is optional, and depends on the SRAMs selected for use with the MPC7410. Note that the least-significant bit of the L2 address (L2ADDR[18:0]) is identified as bit 0, and the most-significant bit is identified as bit 18. See Section 3.7, “L2 Cache Interface,” for more information on the operation of the L2 interface and the interactions of these signals.

8.5.1.1 L2 Address (L2ADDR[18:0])—Output

Following are the state meaning and timing comments for the L2 address output signals.

State Meaning Asserted/Negated—Represents the address of the data to be transferred to the L2 cache. The L2 address bus is configured with bit 0 as the least-significant bit. The L2 address signals reflect the real address for various L2 cache sizes and data bus widths as shown in Table 8-7. Note that the L2 address does not correspond bit-for-bit with the real address.

Table 8-7. L2 Cache Address Signal Mappings

L2ADDR	Real Address Bit							
	64-bit Data Bus				32-bit Data Bus			
	2 Mbyte	1 Mbyte	512 Kbyte	256 Kbyte	2 Mbyte	1 Mbyte	512 Kbyte	256 Kbyte
18	Low (0b0)				12	PM[1]	Low (0b0)	
17	12	PM[1]	Low (0b0)		13		PM[1]	Low (0b0)
16	13		PM[1]	Low (0b0)	14			PM[1]
15	14			PM[1]	15			
14	15				16			
13	Way/11[2]	Way/12[2]	Way/13[2]	Way/14[2]	Way/11[2]	Way/12[2]	Way/13[2]	Way/14[2]
12	17				17			
11	18				18			
10	19				19			
9	20				20			
8	21				21			
7	22				22			
6	23				23			
5	24				24			
4	25				25			
3	26				26			
2	16				27			
1	27				28			
0	28				29			

[1] PM is high (0b1) if the transaction is to private memory space or low (0b0) otherwise.

[2] Way/*nn* is the way associated with the L2 cache tag if the transaction hits in the L2 cache or bit *nn* of the real address if the transaction is to private memory space.

Timing Comments Assertion/Negation—Driven valid by the MPC7410 during read and write operations; driven with static data when the L2 cache memory is not being accessed.

Note that L2ADDR[18] is only required for addressing 2 Mbytes of L2 cache SRAM in 32-bit L2 data bus mode. The MPC7400 does not support 32-bit L2 data bus mode and therefore does not include the L2ADDR[18] signal.

8.5.1.2 L2 Data (L2DATA[0:63])

The data bus (L2DATA[0:63]) consists of 64 signals that are both input and output on the MPC7410.

8.5.1.2.1 L2 Data (L2DATA[0:63])—Output

Following are the state meaning and timing comments for the L2 data output signals.

State Meaning Asserted/Negated—Represents the state of data during a data write transaction. Data is always transferred in full data bus widths—that is, double words in 64-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b00), and words in 32-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b10).

In 32-bit L2 data bus mode, the MPC7410 uses the high-order L2 data signals (L2DATA[0:31]) for L2 data; the low-order data signals (L2DATA[32:63]) are driven low for writes.

Note that the MPC7400 does not support 32-bit L2 data bus mode.

Timing Comments Assertion/Negation—Driven valid by MPC7410 during write operations; driven with static data when the L2 cache memory is not being accessed by a read operation.

High Impedance—Occurs for at least one cycle when transitioning between read and write operations to the L2 cache memory.

8.5.1.2.2 L2 Data (L2DATA[0:63])—Input

Following are the state meaning and timing comments for the L2 data input signals.

State Meaning Asserted/Negated—Represents the state of data during a data read transaction. Data is always transferred in full data bus widths—that is, double words in 64-bit L2 data bus mode, and words in 32-bit L2 data bus mode.

In 32-bit L2 data bus mode, the MPC7410 uses the high-order L2 data signals (L2DATA[0:31]) for L2 data; the low-order data signals (L2DATA[32:63]) are not sampled for reads.

Note that the MPC7400 does not support 32-bit L2 data bus mode.

Timing Comments Assertion/Negation—Driven valid by L2 cache memory during read operations.

8.5.1.3 L2 Data Parity (L2DP[0:7])

The eight data bus parity (L2DP[0:7]) signals on the MPC7410 are both output and input signals.

8.5.1.3.1 L2 Data Parity (L2DP[0:7])—Output

Following are the state meaning and timing comments for the L2 data parity output signals.

State Meaning Asserted/Negated—Represents odd parity for each of the 8 bytes of L2 cache data during write transactions. Odd parity means that an odd number of bits, including the parity bit, are driven high. For 32-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b10), the L2 interface drives parity on the L2DP[0:3] signals and drives the L2DP[4:7] signals low. L2DP0 is associated with bits 0:7 (byte lane 0) of the L2DATA bus.

Timing Comments Assertion/Negation—The same as L2DATA[0:63].
High Impedance—The same as L2DATA[0:63].

8.5.1.3.2 L2 Data Parity (L2DP[0:7])—Input

Following are the state meaning and timing comments for the L2 parity input signals.

State Meaning Asserted/Negated—Represents odd parity for each byte of L2 cache read data. For 32-bit L2 data bus mode, the L2 interface samples parity on the L2DP[0:3] signals and ignores the L2DP[4:7] signals.

Timing Comments Assertion/Negation—The same as L2DATA[0:63]

8.5.2 L2 Cache Clock/Control

The following sections describe the L2 clock and control signals.

8.5.2.1 L2 Chip Enable ($\overline{\mathrm{L2CE}}$)—Output

Following are the state meaning and timing comments for the $\overline{\mathrm{L2CE}}$ signal.

State Meaning Asserted—Indicates that the L2 cache memory devices are being selected for a read or write operation.

Negated—Indicates that the MPC7410 is not selecting the L2 cache memory devices for a read or write operation.

Timing Comments Assertion/Negation—May occur on any cycle. $\overline{\mathrm{L2CE}}$ is driven high during $\overline{\mathrm{HRESET}}$ assertion.

8.5.2.2 L2 Write Enable ($\overline{\mathrm{L2WE}}$)—Output

Following are the state meaning and timing comments for the $\overline{\mathrm{L2WE}}$ signal.

State Meaning Asserted—Indicates that the MPC7410 is performing a write operation to the L2 cache memory.

Negated—Indicates that the MPC7410 is not performing an L2 cache memory write operation.

Timing Comments Assertion/Negation—May occur on any cycle. $\overline{\mathrm{L2WE}}$ is driven high during $\overline{\mathrm{HRESET}}$ assertion.

8.5.2.3 L2 Clock Out A (L2CLK_OUTA)—Output

Following are the state meaning and timing comments for the L2CLK_OUTA signal.

State Meaning Asserted/Negated—Clock output for L2 cache memory devices. The L2CLK_OUTA signal is identical and synchronous with the L2CLK_OUTB signal and provides the capability to drive up to four L2 cache memory devices. If differential L2 clocking is configured through the setting of the L2CR, the L2CLK_OUTB signal is driven phase-inverted with relation to the L2CLK_OUTA signal.

Timing Comments Assertion/Negation—Refer to the MPC7410 hardware specifications for timing comments. The output of L2CLK_OUTA is driven with a frequency of core_frequency/8 upon the assertion of $\overline{\text{HRESET}}$, and it is driven low upon deassertion of $\overline{\text{HRESET}}$.

8.5.2.4 L2 Clock Out B (L2CLK_OUTB)—Output

Following are the state meaning and timing comments for the L2CLK_OUTB signal.

State Meaning Asserted/Negated—Clock output for L2 cache memory devices. The L2CLK_OUTB signal is identical and synchronous with the L2CLK_OUTA signal, and provides the capability to drive up to four L2 cache memory devices. If differential L2 clocking is configured through the setting of the L2CR, the L2CLK_OUTA signal is driven phase inverted with relation to the L2CLK_OUTB signal.

Timing Comments Assertion/Negation—See the MPC7410 hardware specifications for timing comments. The output of L2CLK_OUTB is driven with a frequency of core_frequency/8 upon the assertion of $\overline{\text{HRESET}}$, and it is driven low upon deassertion of $\overline{\text{HRESET}}$.

8.5.2.5 L2 Synchronize Out (L2SYNC_OUT)—Output

Following are the state meaning and timing comments for the L2SYNC_OUT signal.

State Meaning Asserted/Negated—Clock output for L2 clock synchronization. The L2SYNC_OUT signal should be routed half of the trace length to the L2 cache memory devices and returned to the L2SYNC_IN signal input.

Timing Comments Assertion/Negation—See the MPC7410 hardware specifications for timing comments. The output of L2SYNC_OUT is driven with a frequency of core_frequency/8 upon the assertion of $\overline{\text{HRESET}}$, and it is driven low upon deassertion of $\overline{\text{HRESET}}$.

8.5.2.6 L2 Synchronize In (L2SYNC_IN)—Input

Following are the state meaning and timing comments for the L2SYNC_IN signal.

State Meaning Asserted/Negated—Clock input for L2 clock synchronization. The L2SYNC_IN signal is driven by the L2SYNC_OUT signal output.

Timing Comments Assertion/Negation—Refer to the MPC7410 hardware specifications for timing comments. The routing of this signal on the printed circuit board should ensure that the rising edge at L2SYNC_IN is coincident with the rising edge of the clock at the clock input of the L2 cache memory devices.

8.5.2.7 L2 Low-Power Mode Enable (L2ZZ)—Output

Following are the state meaning and timing comments for the L2ZZ signal.

State Meaning Asserted/Negated—Enables low-power mode for certain L2 cache memory devices. Operation of the signal is enabled through the L2CR.

Timing Comments Assertion/Negation—Occurs synchronously with the L2 clock when the MPC7410 enters and exits the nap or sleep power modes; after negation of this signal, at least two L2 clock cycles must elapse before L2 cache operations can resume. The L2ZZ signal is driven low during assertion of $\overline{\text{HRESET}}$.

8.5.3 Interrupts/Reset Signals

Most system status signals are input signals that indicate when exceptions are received, when checkstop conditions have occurred, and when the MPC7410 must be reset. The MPC7410 generates the output signal, $\overline{\text{CKSTP\_OUT}}$, when it detects a checkstop condition. For a detailed description of these signals, see Section 9.7, “Interrupt, Checkstop, and Reset Signal Interactions.”

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

The interrupt ($\overline{\text{INT}}$) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{INT}}$ signal.

State Meaning Asserted—Indicates that the MPC7410 should initiate an external interrupt if enabled in the MSR.

Negated—Indicates that the interrupt is not being requested.

Timing Comments Assertion—May occur at any time asynchronously to SYSCLK; The $\overline{\text{INT}}$ input is level-activated.

Negation—Should not occur until after the interrupt is taken.

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

The system management interrupt ($\overline{\text{SMI}}$) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{SMI}}$ signal.

State Meaning Asserted—Indicates that the MPC7410 should initiate a system management interrupt if enabled in the MSR.

Negated—Indicates that the interrupt is not being requested.

Timing Comments Assertion—May occur at any time asynchronously to SYSCLK; The $\overline{\text{SMI}}$ input is level-activated.

Negation—Should not occur until after the interrupt is taken.

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

The machine check ($\overline{\text{MCP}}$) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{MCP}}$ signal.

State Meaning Asserted—Indicates that the MPC7410 should initiate a machine check interrupt or enter the checkstop state as directed by the MSR.

Negated—Indicates that machine check handling is not being requested.

Timing Comments Assertion—May occur at any time asynchronously to SYSCLK; The $\overline{\text{MCP}}$ input is falling-edge activated.

Negation—May occur any time after the minimum $\overline{\text{MCP}}$ pulse width has been met; see the MPC7410 hardware specifications.

8.5.3.4 Reset Signals

There are two reset signals on the MPC7410—hard reset ($\overline{\text{HRESET}}$) and soft reset ($\overline{\text{SRESET}}$).

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

Following are the state meaning and timing comments for the $\overline{\text{SRESET}}$ signal.

State Meaning Asserted—Initiates processing for a reset exception as described in Section 4.6.1, "System Reset Exception (0x00100)."

Negated—Indicates that normal operation should proceed. See Section 9.7.3, "Reset Inputs."

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the MPC7410 input clock. The $\overline{\text{SRESET}}$ input is negative edge-sensitive.

Negation—May be negated two bus cycles after assertion.

This input has additional functionality in certain test modes.

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

The hard reset ($\overline{\text{HRESET}}$) signal must be used at power-on to reset properly the processor. Following are the state meaning and timing comments for the $\overline{\text{HRESET}}$ signal.

State Meaning Asserted—Initiates a complete hard reset operation when this input transitions from negated to asserted. Causes a reset exception as described in Section 4.6.1, "System Reset Exception (0x00100)." Output drivers are released to high impedance within five clocks after the assertion of $\overline{\text{HRESET}}$.

Negated—Indicates that normal operation should proceed. See Section 9.7.3, "Reset Inputs."

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the MPC7410 input clock; must be held asserted for a minimum of 255 clock cycles after the PLL lock time has been met. Refer to the MPC7410 hardware specification for further timing comments.

Negation—May occur any time after the minimum reset pulse width has been met.

This input has additional functionality in certain test modes.

8.5.3.5 Checkstop Input (CKSTP_IN)—Input

Following are the state meaning and timing comments for the CKSTP_IN signal.

State Meaning Asserted—Indicates that the MPC7410 must terminate operation by internally gating off all clocks, and release all outputs (except CKSTP_OUT, L2CE, L2WE, and L2CLK_OUTx) to the high-impedance state. Once CKSTP_IN has been asserted; it must remain asserted until the system has been reset.

Negated—Indicates that normal operation should proceed. See Section 9.7.2, “Checkstops.”

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

Negation—May occur any time after the CKSTP_OUT output signal has been asserted.

8.5.3.6 Checkstop Output (CKSTP_OUT)—Output

Note that the CKSTP_OUT signal is an open-drain type output, and requires an external pull-up resistor (for example, 10 kΩ to OV_{DD}) to assure proper negation of the CKSTP_OUT signal. Following are the state meaning and timing comments for the CKSTP_OUT signal.

State Meaning Asserted—Indicates that the MPC7410 has detected a checkstop condition and has ceased operation.

Negated—Indicates that the MPC7410 is operating normally.
See Section 9.7.2, “Checkstops.”

Timing Comments Assertion—May occur at any time and may be asserted asynchronously to the MPC7410 input clocks.

Negation—Is negated upon assertion of HRESET.

8.5.3.7 Check (CHK)—Input

Following are the state meaning and timing comments for the CHK signal.

State Meaning Asserted/Negated—Reserved for factory test.

Timing Comments Assertion/Negation—Should be pulled-up for normal operation.

8.5.4 Processor Status/Control Signals

Processor status signals indicate the state of the processor. This includes the memory reservation signal, machine quiesce control signals, and time base enable signal.

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

The reservation ($\overline{\text{RSRV}}$) signal is an output signal on the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{RSRV}}$ signal.

State Meaning Asserted/Negated—Indicates the state of the internal reservation coherency bit used by the **lwarx** and **stwcx.** instructions.

Timing Comments Assertion/Negation—May occur on any cycle; occurs immediately following a transition of the reservation coherency bit.

8.5.4.2 Timebase Enable (TBEN)—Input

The timebase enable (TBEN) signal is an input signal on the MPC7410. Following are the state meaning and timing comments for the TBEN signal.

State Meaning Asserted—Indicates that the timebase and decrementer should continue clocking. This signal functions as a count enable control for the timebase and decrementer counter.

Negated—Indicates that the timebase and decrementer should stop clocking.

Timing Comments Assertion/Negation—May occur at any time asynchronously to SYSCLK

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

The quiescent request ($\overline{\text{QREQ}}$) signal is an output signal on the MPC7410. See Chapter 10, "Power Management," for more information about the power management modes of the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{QREQ}}$ signal.

State Meaning Asserted—Indicates that the MPC7410 is requesting all bus activity to terminate or pause so that it may enter a quiescent (low-power) state. Once in this state, the MPC7410 stops snooping further bus activity**.**

Negated—Indicates that the MPC7410 is not requesting to enter a quiescent state.

Timing Comments Assertion/Negation—May occur on any cycle. $\overline{\text{QREQ}}$ remains asserted for the duration of the quiescent state.

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

The quiescent acknowledge ($\overline{\text{QACK}}$) signal is an input signal on the MPC7410. See Chapter 10, "Power Management," for more information about the power management modes of the MPC7410. Following are the state meaning and timing comments for the $\overline{\text{QACK}}$ signal.

State Meaning Asserted—Indicates that all bus activity has terminated or paused, and the MPC7410 may enter nap or sleep mode.

Negated—Indicates that the MPC7410 may not enter nap or sleep mode, or it must return to doze mode from nap mode in order to snoop.

Timing Comments Assertion/Negation—May occur on any cycle following the assertion of $\overline{\text{QREQ}}$. When negated for at least 8 bus cycles; it ensures that the MPC7410 has returned to doze mode from nap mode.

8.5.4.5 Enhanced Mode ($\overline{\text{EMODE}}$)—Input

The enhanced mode ($\overline{\text{EMODE}}$) signal is an input signal on the MPC7410 sampled at $\overline{\text{HRESET}}$ negation used to select the MPX bus mode operation. The state of $\overline{\text{EMODE}}$ sampled at $\overline{\text{HRESET}}$ negation is stored and readable from the EMODE bit in MSSCR0. The state of MSSR0[EMODE] is active high, meaning that if $\overline{\text{EMODE}}$ is detected as asserted at the negation of $\overline{\text{HRESET}}$, MSSR0[EMODE] = 1. Section 9.6, "MPX Bus Protocol," describes the MPX bus mode operation on the MPC7410.

The $\overline{\text{EMODE}}$ signal is also used to select address bus driven mode after $\overline{\text{HRESET}}$ is negated in MPX bus mode. If $\overline{\text{EMODE}}$ is asserted after $\overline{\text{HRESET}}$ is negated, address bus driven mode is selected; if $\overline{\text{EMODE}}$ is negated after $\overline{\text{HRESET}}$ is negated, normal address bus driving mode (address bus not always driven) is selected. The address bus driven mode is stored and readable from the MSSCR0[ABD] bit. See Section 9.6.1.2.1, "Address Bus Driven Mode," for more information. Note that address bus driven mode is only available in MPX bus mode.

Following are the state meaning and timing comments for the $\overline{\text{EMODE}}$ signals.

State Meaning Asserted—Sampled at $\overline{\text{HRESET}}$ negation to select the bus mode. If $\overline{\text{EMODE}}$ is asserted at $\overline{\text{HRESET}}$ negation, MPX bus mode is selected.

Additionally, if MPX mode is selected, $\overline{\text{EMODE}}$ is used after $\overline{\text{HRESET}}$ negation to select address bus driven mode. Address bus driven mode causes the MPC7410 to drive the address bus whenever $\overline{\text{BG}}$ is asserted independent of whether the MPC7410 has a bus transaction to run or not.

Negated—If $\overline{\text{EMODE}}$ is negated at the negation of $\overline{\text{HRESET}}$, 60x bus mode is selected. Additionally, if $\overline{\text{EMODE}}$ remains negated after $\overline{\text{HRESET}}$ negation (in MPX bus mode), then the address bus driven mode is not selected. The state of $\overline{\text{EMODE}}$ after $\overline{\text{HRESET}}$ negation is ignored in 60x bus mode.

Timing Comments Assertion/Negation—May be tied high to select 60x bus interface operation; may be tied to $\overline{\text{HRESET}}$ to select MPX bus interface operation (without address bus driven mode); may be tied low to select MPX bus plus address bus driven mode.

8.5.5 Clock Control Signals

The MPC7410 clock signal inputs determine the system clock frequency and provide a flexible clocking scheme that allows the processor to operate at an integer multiple of the system clock frequency.

Refer to the MPC7410 hardware specification for the exact timing relationships of the clock signals and other signals.

8.5.5.1 System Clock (SYSCLK)—Input

The MPC7410 requires a single system clock (SYSCLK) input. This input sets the frequency of operation for the bus interface. Internally, the MPC7410 uses a phase-locked loop (PLL) circuit to generate a master clock for all the CPU circuitry (including the bus interface circuitry) which is phase-locked to the SYSCLK input. The master clock may be set to an integer or half-integer multiple of the SYSCLK frequency as defined in the MPC7410 hardware specification, allowing the CPU core to operate at an equal or greater frequency than the bus interface.

Following are the state meaning and timing comments for the SYSCLK signals.

State Meaning Asserted/Negated—The SYSCLK input is the primary clock input for the MPC7410 and represents the bus clock frequency for MPC7410 bus operation. Internally, the MPC7410 may be operating at an integer or half-integer multiple of the bus clock frequency.

Timing Comments Duty cycle—Refer to the MPC7410 hardware specification for timing comments and supported ratios.

SYSCLK is used as the frequency reference for the internal PLL clock generator and must not be suspended or varied during normal operation to ensure proper PLL operation.

8.5.5.2 PLL Configuration (PLL_CFG[0:3])—Input

The PLL (phase-locked loop) is configured by the PLL_CFG[0:3] signals. For a given SYSCLK (bus) frequency, the PLL configuration signals set the internal CPU frequency of operation. See the MPC7410 hardware specification for PLL configuration information.

Following are the state meaning and timing comments for the PLL_CFG[0:3] signals.

State Meaning Asserted/Negated—Configure the operation of the PLL and the internal processor clock frequency. Settings are based on the desired bus frequency and internal frequency of operation.

Timing Comments Assertion/Negation—Must remain stable during operation; should only be changed during the assertion of $\overline{\text{HRESET}}$ or during sleep mode. These bits may be read through the PC[0–3] bits in the HID1 register.

8.5.5.3 Clock Out (CLK_OUT)—Output

The clock out (CLK_OUT) signal is an output signal (output-only) on the MPC7410. Following are the state meaning and timing comments for the CLK_OUT signal.

State Meaning Asserted/Negated—Provides a PLL clock output for PLL testing and monitoring. The configuration of the HID0[BCLK] and HID0[ECLK] bits determines whether the CLK_OUT signal clocks at the processor clock frequency, the bus clock frequency, or half of the bus clock frequency. See Table 2-6 for HID0 register configuration of the CLK_OUT signal. Note that the CLK_OUT signal is provided for testing purposes only. The CLK_OUT signal defaults to a clock with the same frequency as the bus clock (SYSCLK) following the assertion of $\overline{\text{HRESET}}$. Upon deassertion of $\overline{\text{HRESET}}$, the CLK_OUT signal is driven low. The signal, while periodic, is not suitable for use as a reference clock.

Timing Comments Assertion/Negation—Refer to the MPC7410 hardware specification for timing comments.

8.5.6 IEEE Std. 1149.1a-1993 (JTAG) Interface Description

The MPC7410 has five dedicated JTAG signals which are described in Table 8-8. The test data input (TDI) and test data output (TDO) scan ports are used to scan instructions as well as data into the various scan registers for JTAG operations. The scan operation is controlled by the test access port (TAP) controller which in turn is controlled by the test mode select (TMS) input sequence. The scan data is latched in at the rising edge of test clock (TCK).

Table 8-8. IEEE Interface Pin Descriptions

Signal Name	Input/Output	Weak Pullup Provided	IEEE Std. 1149.1a Function
TCK	Input	No	Scan clock
TDI	Input	Yes	Serial scan input signal
TDO	Output	No	Serial scan output signal
TMS	Input	Yes	TAP controller mode signal
TRST	Input	Yes	TAP controller reset

Test reset ($\overline{\text{TRST}}$) is an optional JTAG signal which is used in the MPC7410 to reset the TAP controller asynchronously. The $\overline{\text{TRST}}$ signal assures that the JTAG logic does not interfere with the normal operation of the device. It is recommended that $\overline{\text{TRST}}$ be asserted and negated coincident with the assertion of the $\overline{\text{HRESET}}$ signal.

These signals are not used during normal operation. TMS, TDI, and $\overline{\text{TRST}}$ have internal pull-ups provided; TCK does not. For normal operation, TMS and TDI may be left unconnected, and TCK must be set high or low. $\overline{\text{TRST}}$ must be asserted sometime during power-up for JTAG logic initialization. Note that if $\overline{\text{TRST}}$ is tied low, then unnecessary power is consumed.

8.5.6.1 JTAG Test Clock (TCK)—Input

The JTAG test clock (TCK) signal is an input on the MPC7410. Following is the state meaning for the TCK input signal.

State Meaning Asserted/Negated—This input should be driven by a free-running clock signal. Input signals to the test access port are clocked in on the rising edge of TCK. Changes to the test access port output signals occur on the falling edge of TCK. The test logic allows TCK to be stopped.

8.5.6.2 JTAG Test Data Input (TDI)—Input

Following is the state meaning for the TDI input signal.

State Meaning Asserted/Negated—The value presented on this signal on the rising edge of TCK is clocked into the selected JTAG test instruction or data register.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

8.5.6.3 JTAG Test Data Output (TDO)—Output

The JTAG test data output signal is an output on the MPC7410. Following is the state meaning for the TDO output signal.

State Meaning Asserted/Negated—The contents of the selected internal instruction or data register are shifted out onto this signal on the falling edge of TCK. The TDO signal remains in a high-impedance state except when scanning of data is in progress.

8.5.6.4 JTAG Test Mode Select (TMS)—Input

The test mode select (TMS) signal is an input on the MPC7410. Following is the state meaning for the TMS input signal.

State Meaning Asserted/Negated—This signal is decoded by the internal JTAG TAP controller to distinguish the primary operation of the test support circuitry.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

8.5.6.5 JTAG Test Reset ($\overline{\text{TRST}}$)—Input

The test reset ($\overline{\text{TRST}}$) signal is an input on the MPC7410. Following is the state meaning for the $\overline{\text{TRST}}$ input signal.

State Meaning Asserted—This input causes asynchronous initialization of the internal JTAG test access port controller. Note that the signal must be asserted during the assertion of $\overline{\text{HRESET}}$ in order to properly initialize the JTAG test access port.

Negated—Indicates normal operation.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level (negated) to the test logic.

8.5.7 Bus Voltage Select (BVSEL)/L2 Voltage Select (L2VSEL)

The MPC7410 provides several I/O voltages to support both compatibility with existing systems and migration to future systems. See the MPC7410 hardware specification for more information on the BVSEL and L2VSEL signals, which control various I/O voltage options.

8.5.7.1 Bus Voltage Select (BVSEL)—Input

State Meaning Assertion/Negation—Selects the high voltage level for all main bus and utility signals (for example, all signals except L2 interface signals). See the MPC7410 hardware specification for more information.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

Timing Comments Assertion/Negation—Must remain asserted or negated during normal operation.

8.5.7.2 L2 Voltage Select (L2VSEL)—Input

State Meaning Assertion/Negation—Selects the high voltage level for all L2 interface signals. See the MPC7410 hardware specification for more information.

Note that this input contains an internal pull-up resistor to ensure that an unterminated input appears as a high signal level to the test logic.

Timing Comments Assertion/Negation—Must remain asserted or negated during normal operation.

8.5.8 Power and Ground Signals

The MPC7410 provides the following connections for power and ground:

- V_{DD}—The V_{DD} signals provide the supply voltage connection for the processor core.
- OV_{DD}—The OV_{DD} signals provide the supply voltage connection for the system interface drivers.
- $L2OV_{DD}$—The $L2OV_{DD}$ signals provide the supply voltage connection for the L2 cache interface drivers. These power supply signals are isolated from the V_{DD} and OV_{DD} power supply signals.
- AV_{DD}—The AV_{DD} power signal provides power to the clock generation phase-locked loop. See the MPC7410 hardware specifications for information on how to use this signal.
- $L2AV_{DD}$—The $L2AV_{DD}$ power signal provides power to the L2 delay-locked loop. See the MPC7410 hardware specifications for information on how to use this signal.
- GND and OGND—The GND and OGND signals provide the connection for grounding the MPC7410. On the MPC7410, there is no electrical distinction between the GND and OGND signals.
- L2GND—The L2GND signals provide the ground connection for the L2 cache interface. These ground signals are isolated from the GND and OGND ground signals.

See the MPC7410 hardware specification for detailed electrical and mechanical information for each signal.

Chapter 9
System Interface Operation

This chapter describes the MPC7410 microprocessor bus interface and its operation. It shows how the MPC7410 signals, defined in Chapter 8, "Signal Descriptions," interact to perform address and data transfers.

9.1 MPC7410 System Interface Overview

There are two interface protocols used by the MPC7410—the 60x bus interface and the MPX bus interface. The 60x bus interface implements the protocol described in the . Note that although this protocol is implemented by the MPC603e, MPC604, MPC740 and MPC750 processors, it is referenced as the 60x bus interface.

The MPX protocol is derived from the 60x bus interface. This new interface includes several additional features that provide higher memory bandwidth than the 60x bus and more efficient use of the system bus in a multiprocessing environment.

The value of the $\overline{\text{EMODE}}$ signal at the negation of $\overline{\text{HRESET}}$ determines whether the MPC7410 operates with the 60x bus or the MPX bus protocol. This value is stored in and readable from the EMODE bit in MSSCR0. The state of MSSR0[EMODE] is active high, meaning that if $\overline{\text{EMODE}}$ is detected as asserted at the negation of $\overline{\text{HRESET}}$, MSSR0[EMODE] = 1 and MPX bus mode is selected; otherwise, MSSCR0[EMODE] = 0 and 60x bus mode is selected.

When operating in 60x bus mode, the MPC7410 is logically and mechanically compatible with the MPC750, although a few parameters may vary such as the number of outstanding transactions. Also, it may not be electrically compatible due to voltage differences. Refer to the MPC7410 Hardware Specifications for electrical information on I/O levels and power supply levels.

9.1.1 MPC7410 Bus Operation Features

The MPC7410 has a separate address and data bus, each with its own set of arbitration and control signals. This allows for the decoupling of 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
- Enveloped transaction operation

The MPC7410 supports only the normal memory-mapped address segments defined in the architecture. Access to direct-store segments results in a DSI exception.

9.1.1.1 60x Bus Features

The following list summarizes the 60x bus interface features:

- 32-bit address bus (plus 4 bits of odd parity)
- 64-bit data bus (plus 8 bits of odd parity); a 32-bit data bus mode is not provided
- Support for two cache coherency protocols:
  - — Three-state (MEI) similar to the MPC750
  - — Four-state (MESI) similar to the MPC604 processors
- On-chip snooping to maintain L1 data cache and L2 cache coherency for multiprocessing applications
- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Support for limited out-of-order transactions
- Support for up to seven outstanding transactions (six pending plus one data tenure in progress).
- TTL-compatible interface

9.1.1.2 MPX Bus Features

The MPX bus mode provides increased performance over the 60x bus mode.

The following list summarizes the MPX bus mode features:

- Increased address bus bandwidth by eliminating dead cycles under some circumstances
- Full data streaming for reads and writes under some circumstances
- Support for full out-of-order transactions
- Support for data intervention in multiprocessing systems
- Support for third cache five-state coherency protocol, Modified, Exclusive, Reserved, Shared, Invalid (MERSI), where the new R state allows shared intervention
- Improved electrical timings (for example, programmable option for keeping address bus driven)

9.1.2 Overview of System Interface Accesses

The system interface includes address register queues, prioritization logic, and a bus control unit. The system interface latches snoop addresses for snooping in the L1 and L2 data caches, in 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.

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

When the MPC7410 encounters an instruction or data access, it calculates the effective address and uses the low-order address bits to check for a hit in the on-chip, 32-Kbyte L1 instruction and data caches. During L1 cache lookup, the instruction and data memory management units (MMUs) use the higher-order address bits to calculate the virtual address, from which they calculate the physical address (real address in the architecture specification). The physical address bits are then compared with the corresponding cache tag bits to determine if a cache hit occurred in the L1 instruction or data cache. If the access misses in the corresponding cache, the physical address is used to access the L2 cache tags (if the L2 cache is enabled). If no match is found in the L2 cache tags, the physical address is used to access system memory.

In addition to the loads, stores, and instruction fetches, the MPC7410 performs hardware table search operations following TLB misses, L2 cache castout operations when least-recently used cache lines are written to memory after a cache miss, and cache-line snoop push-out operations when a modified cache line detects a snoop hit from another bus master.

Figure 9-1 shows a block diagram of the MPC7410, including the address path from the execution units and instruction fetcher, through the translation logic to the caches and system interface logic.

The MPC7410 uses separate address and data buses and a variety of control and status signals for performing external reads and writes. The address bus is 32 bits wide and the data bus is 64 bits wide. The interface is synchronous—all MPC7410 inputs are sampled at and all outputs are driven from the rising edge of the bus clock. The processor runs at a multiple of the bus-clock speed.

9.1.3 Summary of L1 Instruction and Data Cache Operation

The MPC7410 provides independent L1 instruction and data caches. Each cache is a physically-addressed, 32-Kbyte cache with 8-way set associativity. Both caches consist of 128 sets of 8 cache lines, with 8 words in each cache line.

The MPC7410 data cache tags are dual-ported and non-blocking allowing efficient load/store and snoop operations.

When configured for MPX bus mode and with data intervention enabled, the MPC7410 supports a five-state cache coherency protocol that includes Modified (M), Exclusive (E), Reserved (R), Shared (S), and Invalid (I) cache states. The MERSI protocol together with the MPX bus allows for data intervention between caches. Alternately, when configured for either MPX bus or 60x bus modes, the MPC7410 can be configured to support a four-state MESI protocol (similar to the MPC604-family microprocessors) or a three-state MEI protocol (similar to the MPC603- and the MPC750-family microprocessors). MESI or MEI coherency protocol is selected by the MSSCR0[SHDEN] parameter.

The cache control instructions, **dcbt**, **dcbtst**, **dcbz**, **dcbst**, **dcbf**, **dcba**, **dcbi**, and **icbi**, are intended for the management of the local L1 and L2 caches. The MPC7410 interprets the cache control instructions as if they pertain only to its own L1 or L2 caches. These instructions are not intended for managing other caches in the system (except to the extent necessary to maintain coherency). The MPC7410 snoops all global ($\overline{\text{GBL}}$ asserted) cache control instruction broadcasts. The **dcbst**, **dcbf**, and **dcbi** instructions cause a broadcast on the system bus (when M = 1) to maintain coherency. The **icbi** instruction is always broadcast, regardless of the state of the memory-coherency-required attribute.

Because the data cache on the MPC7410 is an on-chip, write-back primary cache, the predominant type of transaction for most applications is burst-read memory operations, followed by burst-write memory operations and single-beat (caching-inhibited or write-through) memory read and write operations. Additionally, there can be address-only operations, variants of the burst and single-beat operations (for example, global memory operations that are snooped, and atomic memory operations), and address retry activity (for example, when a snooped read access hits a modified line in the cache).

On a cache miss, cache blocks are filled in four beats of 64 bits each. The burst fill is performed as a critical-double-word-first operation. For the instruction cache, the critical double word is simultaneously written to the cache and forwarded to the instruction queue, thus minimizing stalls due to cache fill latency. The instruction cache is not blocked to internal accesses while a load completes, providing for “hits under misses.” For the data cache, an entire cache block is collected in a reload buffer before being loaded into the cache. This allows the data cache to service multiple outstanding misses while at the same time staying available to subsequent load and store hits.

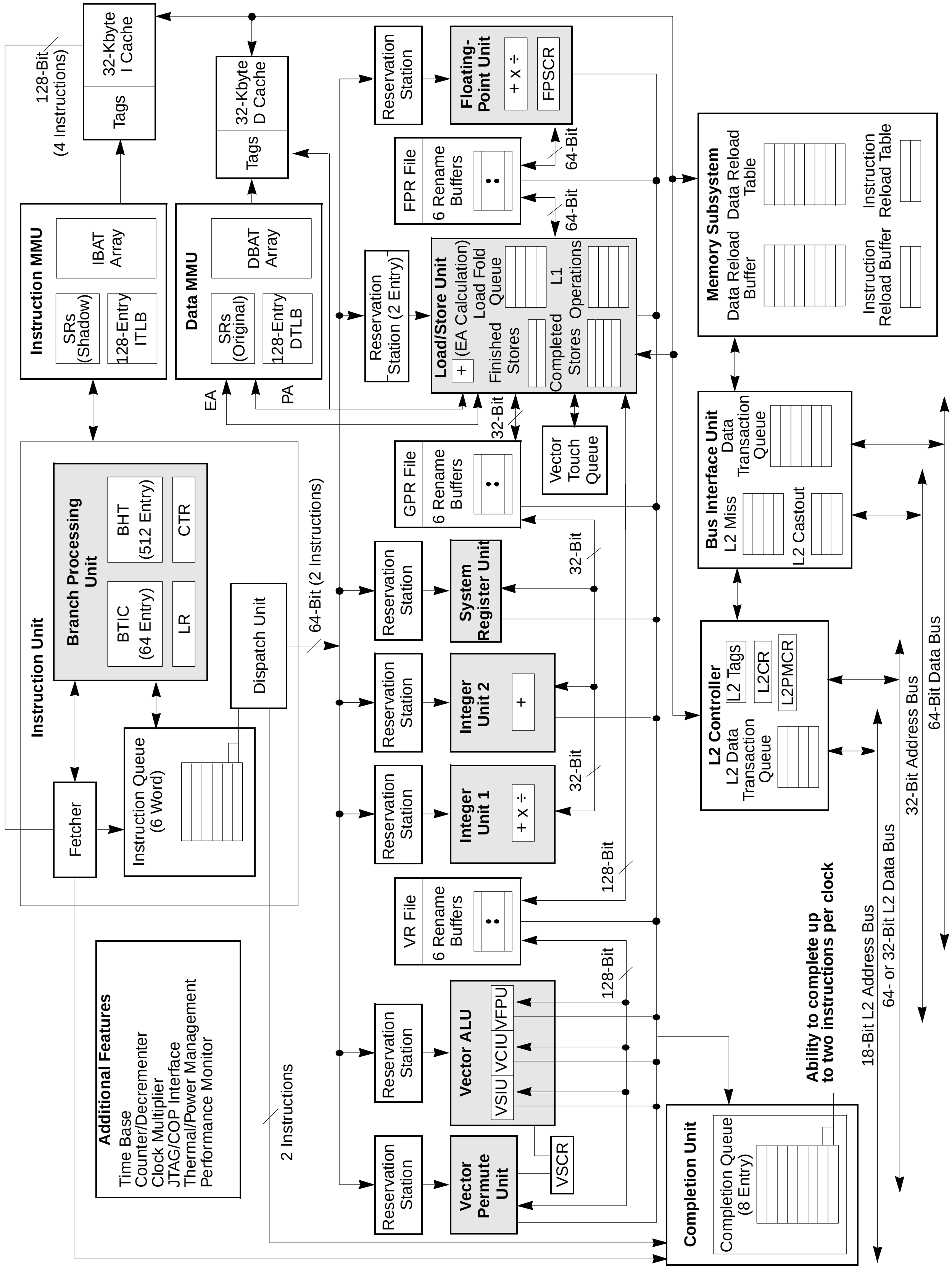


Figure 9-1. MPC7410 Microprocessor Block Diagram

Cache lines are selected for replacement based on a pseudo least-recently-used (PLRU) algorithm. Each time a cache block is accessed, it is tagged as the most recently used way of the set (unless accessed by the AltiVec LRU instructions, see Section 3.6.8.1, “AltiVec LRU Instruction Support”). For every hit in the cache or when a new block is reloaded, the PLRU bits for the set are updated. Data cache replacement selection is performed at reload time, not when a miss occurs. However, instruction cache replacement selection occurs when an instruction cache miss is first recognized—that is, the instruction cache replacement target is selected upon miss and not at reload.

A data cache block fill is caused by a load miss or write-back store miss in the cache. The cache block that corresponds to the missed address is updated by a burst transfer of the data from the L2 cache or system memory after any necessary coherency actions have completed.

For more information about the interactions of the instruction and data caches and the system interface, see Section 3.8, “System Bus Interface Unit.”

9.1.4 L2 Cache and System Interface

The MPC7410 provides an on-chip, two-way set associative tag memory, and a dedicated L2 cache port with support for up to 2 Mbyte of external synchronous SRAMs for data storage. The L2 cache usually operates in write-back mode and supports system cache coherency through snooping.

The L2 cache receives independent memory access requests from both the L1 instruction and data caches. The L1 accesses are compared to the L2 cache tags and the data or instructions are forwarded from the L2 to the L1 cache if there is a cache hit, or are forwarded on to the bus interface unit if there is an L2 cache miss, or if the address being accessed is from a page marked as caching-inhibited. Burst read accesses that miss in the L2 cache initiate a load operation from the bus interface. L1 data cache misses cause allocates into the data cache only; they do not cause allocation into the L2 cache. The L2 cache is solely a victim cache for the L1 data cache. The L2 cache allocates new entries for data accesses only when blocks are cast out of the data cache.

An L1 load, store, or castout operation can cause an L2 cache block allocation resulting in the castout of an L2 cache block marked modified to the bus interface. For additional information about the operation of the L2 cache, refer to Section 3.7, “L2 Cache Interface.”

9.1.5 Operation of the System Interface

Memory accesses can occur in single-beat (1, 2, 3, 4, and 8 bytes), double-beat (16 bytes), and four-beat (32 bytes) burst data transfers. For memory accesses, the address and data buses are independent to support pipelining and split transactions. The MPX bus protocol can pipeline as many as seven transactions and supports full out-of-order split-bus transactions.

Access to the system interface is granted through an external arbitration mechanism that allows devices to compete for bus mastership. This arbitration mechanism is flexible, allowing the MPC7410 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 MPC7410 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 following sections describe how the MPC7410 interfaces operate, providing detailed timing diagrams that illustrate how the signals interact. Several general timing diagrams are included as examples of typical bus operations. See Figure 9-2 for the conventions used in the timing diagrams.

This is a synchronous interface—all MPC7410 input signals are sampled and output signals are driven on the rising edge of the bus clock cycle (see MPC7410 Hardware Specification for exact timing information).

9.1.6 Memory Subsystem Control Register (MSSCR0) Effects

The MSSCR0 control register is used to configure many aspects of the memory subsystem and bus protocols for the MPC7410. It is a supervisor-only read/write, implementation-specific register accessed as SPR 1014. MSSCR0 alters how the MPC7410 responds to snoop requests, see Section 2.1.5.3, “Memory Subsystem Control Register (MSSCR0),” for more details. MSSCR0 enables the shared cache coherency state (MEI vs. MESI), disables the sampling of the $\overline{\text{SHD}}$[0:1] signals in MEI coherency mode, enables L1 and L2 data cache intervention in MPX bus mode, enables L1 data cache flushing in hardware, reflects the address bus driven mode state in MPX bus mode, and reflects the state of the $\overline{\text{EMODE}}$ signal during power-on reset. See Section 2.1.5.3, “Memory Subsystem Control Register (MSSCR0),” for more detailed information about the bits of MSSCR0.

9.1.7 Direct-Store Accesses Not Supported

The MPC7410 does not support the extended transfer protocol for accesses to the direct-store storage space. The transfer protocol used for any given access is selected by the T bit in the MMU segment registers; if the T bit is set, the memory access is a direct-store access. An attempt to access instructions or data in a direct-store segment causes the MPC7410 to take an ISI or DSI exception.

Figure 9-2 provides the timing diagram legend.

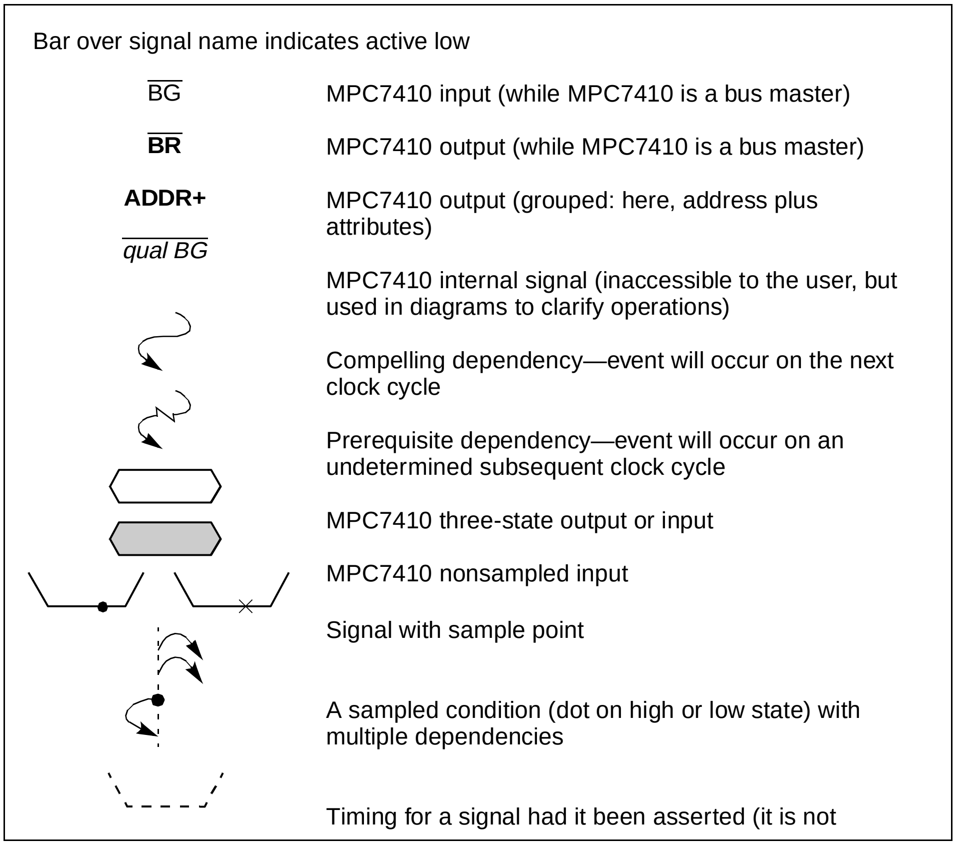

Bar over signal name indicates active low
BG
MPC7410 input (while MPC7410 is a bus master)
BR
MPC7410 output (while MPC7410 is a bus master)
ADDR+
MPC7410 output (grouped: here, address plus attributes)
qual BG
MPC7410 internal signal (inaccessible to the user, but used in diagrams to clarify operations)
Compelling dependency—event will occur on the next clock cycle
Prerequisite dependency—event will occur on an undetermined subsequent clock cycle
MPC7410 three-state output or input
MPC7410 nonsampled input
Signal with sample point
A sampled condition (dot on high or low state) with multiple dependencies
Timing for a signal had it been asserted (it is not

Figure 9-2. Timing Diagram Legend

9.2 60x Bus Protocol

Memory accesses that use the 60x bus protocol are divided into address and data tenures. Each tenure has three phases—bus arbitration, transfer, and termination. The MPC7410 60x bus protocol also supports address-only transactions. Note that address and data tenures can overlap, as shown in Figure 9-3.

Figure 9-3 shows that the address and data tenures in the 60x bus protocol are distinct from one another consist of three phases—arbitration, transfer, and termination. Address and data tenures are independent; that is, the data tenure begins before the address tenure ends, which allows split-bus transactions to be implemented at the system level in multiprocessor systems (see Figure 9-3). Figure 9-3 shows a data transfer that consists of a single-beat transfer of as many as 64 bits. Four-beat burst transfers of 32-byte cache lines require data transfer termination signals for each beat of data.

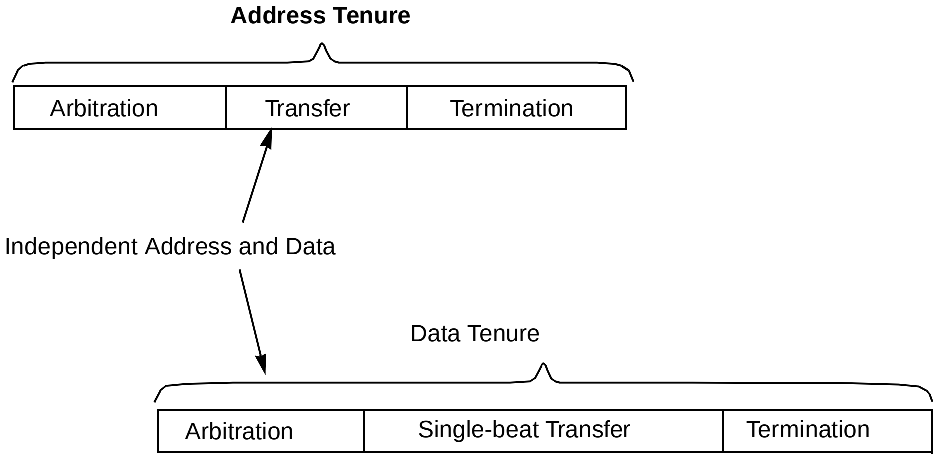


Figure 9-3. Overlapping Tenures on the MPC7410 Bus for a Single-Beat Transfer

The basic functions of the address and data tenures are as follows:

- Address tenure
  - Arbitration: During arbitration, address bus arbitration signals are used to gain mastership of the address bus.
  - Transfer: After the MPC7410 is the address bus master, it transfers the address on the address bus. The address signals and the transfer attribute signals control the address transfer. The address parity signals ensure the integrity of the address transfer.
  - Termination: After the address transfer, the system signals that the address tenure is complete or that it must be repeated.
- Data tenure
  - Arbitration: To begin the data tenure, the MPC7410 arbitrates for mastership of the data bus.
  - Transfer: After the MPC7410 is the data bus master, it samples the data bus for read operations or drives the data bus for write operations. The data parity signals ensure the integrity of the data transfer.
  - Termination: Data termination signals are required after each data beat in a data transfer. Note that in a single-beat transaction, the data termination signals also indicate the end of the tenure, while in burst accesses, the data termination signals apply to individual beats and indicate the end of the tenure only after the final data beat.

The MPC7410 generates an address-only bus transfer during the execution of the **dcbz**, **sync**, and **eieio** instructions and in some instances of the **dcbf** and **dcbst** instructions (when they do not result in a hit of modified data in the cache). Additionally, the MPC7410’s retry capability provides an efficient snooping protocol for systems with multiple memory systems (including caches) that must remain coherent.

9.2.1 Arbitration Signals—Overview

Arbitration for both address and data bus mastership is performed by a central, external arbiter and, minimally, by the arbitration signals shown in Section 8.2.2, “Address Bus Arbitration Signals.” Most arbiter implementations require additional signals to coordinate bus master/slave/snooping activities. Note that while address bus busy ($\overline{\text{ABB}}$) and data bus busy ($\overline{\text{DBB}}$) are bidirectional signals on other processors that implement the 60x bus, they are output-only on the MPC7410. However, they must be connected high through pull-up resistors, so that they remain negated when no devices have control of the buses.

The following list describes the address arbitration signals:

- $\overline{\text{BR}}$ (bus request)—Assertion indicates that the MPC7410 is requesting mastership of the address bus.
- $\overline{\text{BG}}$ (bus grant)—Assertion indicates that the MPC7410 may, with the proper qualification, assume mastership of the address bus. A qualified bus grant occurs when $\overline{\text{BG}}$ is asserted and $\overline{\text{ARTRY}}$ is negated in the current and previous cycle, $\overline{\text{TS}}$ is negated, and the internally-generated address bus busy (*abb*) signal is negated.

  If the MPC7410 is parked, $\overline{\text{BR}}$ need not be asserted for the qualified bus grant.
- $\overline{\text{ABB}}$ (address bus busy)— Assertion by the MPC7410 indicates that the MPC7410 is the address bus master.

The following list describes the data arbitration signals:

- $\overline{\text{DBG}}$ (data bus grant)—Indicates that the MPC7410 may, with the proper qualification, assume mastership of the data bus. A qualified data bus grant occurs when $\overline{\text{DBG}}$ is asserted, $\overline{\text{ARTRY}}$ is not asserted (for the current transaction), and the MPC7410 has a data transaction to perform.

  Note that $\overline{\text{DRTRY}}$ is not implemented on the MPC7410 and thus the 60x bus mode implemented in the MPC7410 is a 60x no-$\overline{\text{DRTRY}}$ mode protocol.
- $\overline{\text{DBWO}}$ (data bus write only)—Assertion indicates that the MPC7410 may perform the data bus tenure for an outstanding write address even if a read address is pipelined before the write address. If $\overline{\text{DBWO}}$ is asserted, the MPC7410 assumes data bus mastership for a pending data bus write operation; the MPC7410 assumes data bus mastership for a pending read operation if this input is asserted along with $\overline{\text{DBG}}$ and no write is pending. Care must be taken with $\overline{\text{DBWO}}$ to ensure the desired write is queued (for example, a cache-line snoop push-out operation). See Section 9.4.4, “Using Data Bus Write Only (DBWO),” for more information.
- $\overline{\text{DBB}}$ (data bus busy)—Assertion by the MPC7410 indicates that the MPC7410 is the data bus master. The MPC7410 always assumes data bus mastership if it needs the data bus and is given a qualified data bus grant (see $\overline{\text{DBG}}$ above).

  For more detailed information on the arbitration signals, refer to Section 8.2.2, “Address Bus Arbitration Signals,” and Section 8.2.6, “Data Bus Arbitration Signals.”

9.2.2 Address Pipelining and Split-Bus Transactions

The 60x bus protocol provides independent address and data bus capability to support pipelined and split-bus transaction system organizations. Address pipelining allows the address tenure of a new bus transaction to begin before the data tenure of the current transaction has finished. Split-bus transaction capability allows other bus activity to occur (either from the same master or from different masters) between the address and data tenures of a transaction.

While this capability does not inherently reduce memory latency, support for address pipelining and split-bus transactions can greatly improve effective bus/memory throughput. For this reason, these techniques are most effective in shared-memory multimaster implementations where bus bandwidth is an important measurement of system performance.

External arbitration is required in systems in which multiple devices must compete for the system bus. The external arbiter must control the pipeline depth and synchronization between masters and slaves.

The design of the external arbiter affects pipelining by regulating address bus grant (BG), data bus grant (DBG), and address acknowledge (AACK) signals. For example, a one-level pipeline is enabled by asserting AACK to the current address bus master and granting mastership of the address bus to the next requesting master before the current data bus tenure has completed. The MPC7410 can pipeline up to six address tenures before starting a data tenure. A seventh address tenure can be initiated once the DBG for the first data tenure has occurred.

In a pipelined implementation, data bus tenures are kept in strict order with respect to address tenures. The MPC7410 60x bus protocol supports a limited intraprocessor out-of-order, split-transaction capability via the data bus write only (DBWO) signal. For more information about using DBWO, see Section 9.4.4, “Using Data Bus Write Only (DBWO).”

9.3 60x Address Bus Tenure

This section describes the three phases of the address tenure used in the 60x bus protocol—address bus arbitration, address transfer, and address transfer termination.

9.3.1 Address Bus Arbitration

When the MPC7410 needs access to the external bus and it is not parked (BG is negated), it asserts bus request (BR) until it is granted mastership of the bus and the bus is available; see Figure 9-4. The external arbiter must grant master-elect status to the potential master by asserting the bus grant (BG) signal after an externally synthesized version of ABB is negated.

When designing external bus arbitration logic, note that the MPC7410 may assert BR without using the bus after it receives the qualified bus grant. The internally generated *abb* signal is asserted in the cycle of any TS assertion on the external bus. The *abb* signal remains asserted until the corresponding assertion of AACK. For example, in a system using bus snooping, if the MPC7410 asserts BR to perform a replacement copyback operation, another device can invalidate that line before the MPC7410 is granted mastership of the bus. When the MPC7410 is granted mastership of the bus, it no longer needs to perform the copyback operation; therefore, the MPC7410 does not assert ABB and does not use the bus for the copyback operation. Another example is the case of a transaction generated by a **stwcx.** instruction whose reservation is cancelled by the snoop of another master currently using the bus. Note that the MPC7410 asserts BR for at least one clock cycle before negating it in these instances.

9.3.1.1 Qualified Bus Grant

A qualified bus grant occurs when BG is asserted, TS is negated, ARTRY is negated in both the current cycle and the preceding cycle, and the internally generated *abb* signal is negated. The MPC7410 asserts ABB when it receives a qualified bus grant.

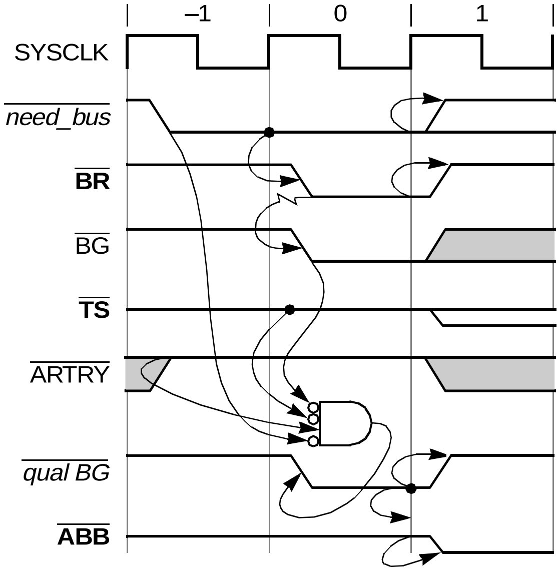


Figure 9-4. Address Bus Arbitration

9.3.1.2 Bus Parking

External arbiters must allow only one device at a time to be the address bus master. In systems where no other device can be a master, BG can be grounded (always asserted) to grant continually mastership of the address bus to the MPC7410 (called bus parking).

If the MPC7410 asserts BR before the external arbiter asserts BG, the MPC7410 is considered to be unparked, as shown in Figure 9-4. Figure 9-5 shows the parked case, where a qualified bus grant exists on the clock edge following a *need_bus* condition. Notice that the bus clock cycle required for arbitration is eliminated if the MPC7410 is parked, reducing overall memory latency for a transaction. The MPC7410 always negates ABB for at least one bus clock cycle after AACK is asserted, even if it is parked and has another transaction pending.

The MPC7410 qualifies BG in 60x bus mode with the negation of an internally generated ABB as well as the previously defined terms before accepting address bus ownership. Thus, the qualified bus grant (QBG) is as follows:

QBG = BG & ¬ARTRY & ¬TS & ¬(latched state variables) & ¬*abb* (internally generated)

where “latched state variables” include latched ARTRY. Thus, a qualified bus grant occurs when BG is asserted, ARTRY is not asserted in the current or in the preceding cycle, and TS is not asserted by this or any other processor.

The negation of ABB indicates that the MPC7410 is not currently using the address bus. The negation of ARTRY indicates that the address retry window for any just-completed address tenure has passed. The

ARTRY input is only monitored for a qualified bus grant on the cycle after the assertion of AACK. (Note that, in general, the sampling of ARTRY may require qualification since ARTRY may be set to the high-impedance state the 2nd cycle following the assertion of AACK and cannot be sampled reliably on that clock.) Upon recognizing a qualified bus grant, the processor takes address bus mastership by asserting ABB and negating BR (non-parked case). At this time or later, the processor drives the address and transfer attributes for the requested access and asserts TS to indicate the start of a new transaction. Note that the MPC7410 may not be ready to immediately assert TS after a qualified BG. The timing of TS may be dependent on resource constraints and may require forward progress on the data bus.

Typically, bus parking is provided to the device that was the most recent bus master; however, system designers may choose other schemes such as providing unrequested bus grants in situations where it is easy to predict correctly the next device requesting bus mastership.

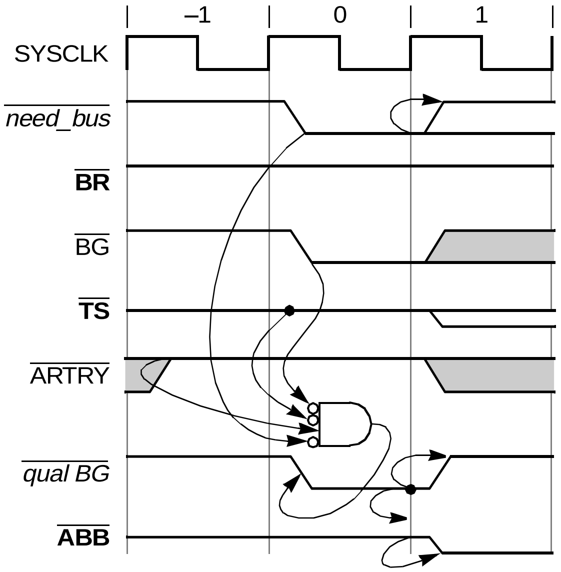


Figure 9-5. Address Bus Arbitration Showing Bus Parking

9.3.1.3 Ignoring ABB

The 60x bus protocol allows for masters that do not implement the ABB signal as an input, such as the MPC7410. The elimination of ABB from the MPC7410 interfaces removes logic from critical timing paths in the processor interface, allowing higher frequency bus operation.

The ABB signal may be ignored if ABB and TS are asserted simultaneously by all masters, or where arbitration (through assertion of BG) is properly managed in cases where the regenerated ABB may not properly track the ABB signal on the bus. If the MPC7410's ABB signal is ignored by the system, it must be connected to a pull-up resistor to ensure proper operation.

9.3.2 Address Transfer

During the address transfer, the physical address and all attributes of the transaction are transferred from the bus master to the slave device(s). Snooping logic may monitor the transfer to enforce cache coherency; see the description of bus snooping in Section 9.3.3, "Address Transfer Termination."

The MPC7410 supports a little-endian mode in which the low-order address bits are operated on (or *munged*) based on the program-requested data transfer size. This munging is performed internally before the address reaches the internal caches and bus units. When little-endian mode is selected, the 60x bus interface still operates in big-endian mode. That is, byte address 0 of a double word—as selected by A[29:31] on the bus, still selects the most significant (left-most) byte of the double word on data bus byte D[0:7]. Byte lane swapping or other operations may have to be performed externally by the system if the MPC7410 is interfaced to a true little-endian environment.

Note that the MPC7410 does not work with the MPC106 bridge device in little-endian mode if misaligned data is accessed.

The signals used in the address transfer include the following signal groups:

- Address transfer start signal—transfer start ($\overline{\text{TS}}$)
- Address transfer signals—address bus (A[0:31]), and address parity (AP[0:3])
- Address transfer attribute signals—transfer type (TT[0:4]), transfer size (TSIZ[0:2]), transfer burst ($\overline{\text{TBST}}$), cache inhibit ($\overline{\text{CI}}$), write-through ($\overline{\text{WT}}$), and global ($\overline{\text{GBL}}$)

Figure 9-6 shows that the timing for all of these signals, except $\overline{\text{TS}}$, is identical. All of the address transfer and address transfer attribute signals are combined into the ADDR+ grouping in Figure 9-6. The $\overline{\text{TS}}$ signal indicates that the MPC7410 has begun an address transfer and the address and transfer attributes are valid (within the context of a synchronous bus). The MPC7410 always asserts $\overline{\text{TS}}$ coincident with $\overline{\text{ABB}}$.

In Figure 9-6, the address transfer occurs during bus clock cycles 1 and 2 (arbitration occurs in bus clock cycle 0, and the address transfer is terminated in bus clock 3). In this diagram, the address bus termination input, $\overline{\text{AACK}}$, is asserted to the MPC7410 on the bus clock following assertion of $\overline{\text{TS}}$ (as shown by the dependency line). This is the minimum duration of the address transfer for the MPC7410; the duration can be extended by delaying the assertion of $\overline{\text{AACK}}$ for one or more bus clock cycles.

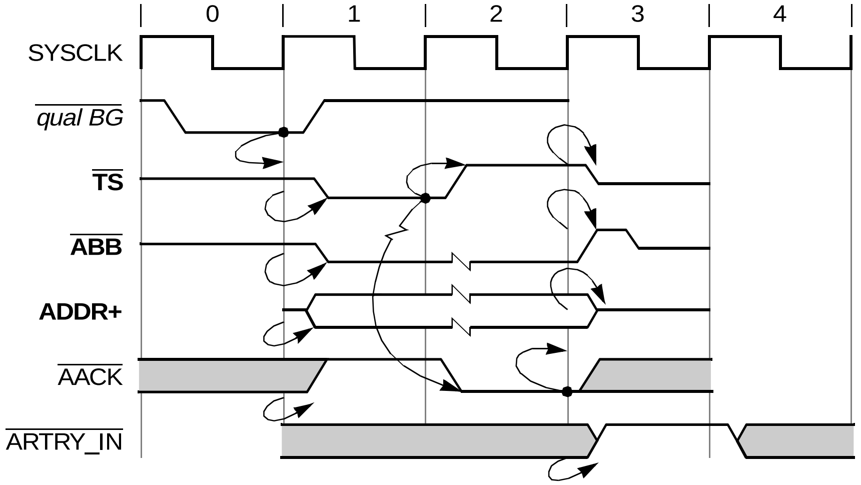


Figure 9-6. Address Bus Transfer

9.3.2.1 Address Bus Parity

The MPC7410 always generates one bit of correct odd-byte parity for each of the four bytes of address when a valid address is on the bus. The calculated values are placed on the AP[0:3] outputs when the MPC7410 is the address bus master. See Section 8.2.3.2, "Address Bus Parity (AP[0:3])," for the parity bit assignments. If the MPC7410 is not the master and $\overline{\text{TS}}$ and $\overline{\text{GBL}}$ are asserted together (qualified condition for snooping memory operations), the MPC7410 calculates parity values for the address bus and the calculated values are compared with the AP[0:3] inputs. If there is an error, and address parity checking is enabled (HID0[EBA] = 1), a machine check exception is generated. An address bus parity error causes a checkstop condition if MSR[ME] is cleared to 0. For more information about checkstop conditions, see Chapter 4, "Exceptions."

9.3.2.2 Address Transfer Attribute Signals

The transfer attribute signals include several encoded signals such as the transfer type (TT[0:4]) signals, transfer burst ($\overline{\text{TBST}}$) signal, transfer size (TSIZ[0:2]) signals, write-through ($\overline{\text{WT}}$), cache inhibit ($\overline{\text{CI}}$), and global ($\overline{\text{GBL}}$).

9.3.2.2.1 Transfer Type (TT[0:4]) Signals in 60x Bus Mode

Snooping logic should fully decode the transfer type signals if the $\overline{\text{GBL}}$ signal is asserted. Slave devices can sometimes use the individual transfer type signals without fully decoding the group. Table 9-1 describes the 60x bus specification transfer encodings and the behavior of the MPC7410 as a bus master.

Table 9-1. Transfer Type Encodings for 60x Bus Mode

Generated by MPC7410 as Bus Master		TT0	TT1[1]	TT2	TT3	TT4	60x Bus Specification	
Type	**Source**						**Command**	**Type**
Address only	**dcbst**	0	0	0	0	0	Clean block	Address only
Address only	**dcbf**	0	0	1	0	0	Flush block	Address only
Address only	**sync**	0	1	0	0	0	**sync**	Address only
Address only	**dcba**, **dcbz**, **dcbi**, store hit on shared block, or store miss merge to 32 bytes	0	1	1	0	0	Kill block	Address only
Address only	**eieio**	1	0	0	0	0	**eieio**	Address only
Single-beat write (non$\overline{\text{GBL}}$)	**ecowx**	1	0	1	0	0	External control word write	Single-beat write
Address only	**tlbie**	1	1	0	0	0	TLB invalidate	Address only
Single-beat read (non$\overline{\text{GBL}}$)	**eciwx**	1	1	1	0	0	External control word read	Single-beat read
N/A	N/A	0	0	0	0	1	**lwarx** reservation set	Address only
N/A	N/A	0	0	1	0	1	Reserved	—

Table 9-1. Transfer Type Encodings for 60x Bus Mode (continued)

Generated by MPC7410 as Bus Master		TT0	TT1[1]	TT2	TT3	TT4	60x Bus Specification	
Type	**Source**						**Command**	**Type**
Address only	**tlbsync**	0	1	0	0	1	**tlbsync**	Address only
Address only	**icbi**	0	1	1	0	1	**icbi**	Address only
N/A	N/A	1	X	X	0	1	Reserved	—
Single-beat write or burst	Caching-inhibited or write-through store	0	0	0	1	0	Write-with-flush	Single-beat write or burst
Burst (non$\overline{\text{GBL}}$)	Cast-out, **dcbf**, **dcbst** push, or snoop copyback	0	0	1	1	0	Write-with-kill	Burst
Single-beat read or burst	Data load or instruction fetch	0[2]	1	0	1	0	Read	Single-beat read or burst
Burst	Store miss, **dcbtst**, **dstst**, **dststt**[3]	0	1	1	1	0	Read-with-intent-to-modify	Burst
Single-beat write	**stwcx.** (caching-inhibited store)	1	0	0	1	0	Write-with-flush-atomic	Single-beat write
N/A	N/A	1	0	1	1	0	Reserved	N/A
Single-beat read or burst	**lwarx** (caching-inhibited load)	1[1]	1	0	1	0	Read-atomic	Single-beat read or burst
Burst	**stwcx.**	1	1	1	1	0	Read-with-intent-to-modify-atomic	Burst
N/A	N/A	0	0	0	1	1	Reserved	—
N/A	N/A	0	0	1	1	1	Reserved	—
N/A	N/A	0	1	0	1	1	Read-with-no-intent-to-cache	Single-beat read or burst
N/A	N/A	0	1	1	1	1	Reserved	—
N/A	N/A	1	X	X	1	1	Reserved	—

[1] TT1 can generally be interpreted as a read/write indicator for the bus.

[2] If HID0[IFTT] = 0b0, TT0 differentiates between a Read-atomic (**lwarx**) operation—TT0 high, and a Read (cache-inhibiting load or instruction fetch) operation—TT0 low. If HID0[IFTT] = 1, TT0 differentiates between data loads (including both atomic and non-atomic loads)—TT0 high, and instruction fetches—TT0 low. Note that touch-for-load instructions (**dcbt**, **dst**, and **dstt**) are identified as read operations (TT = 01010) regardless of the value in HID0 (IFTT).

[3] In 60x bus mode, TT[0:4] = 0b01110 for reads caused by **dcbtst**, **dstst**, and **dststt**. In MPX bus mode, TT[0:4] = 0b01111 for reads caused by **dcbtst**, **dstst**, and **dststt**. See Section 9.6.1.3.1, “Transfer Type 0–4 (TT[0:4]) in MPX Bus Mode,” for more information.

9.3.2.2.2 Transfer Size (TSIZ[0:2]) Signals

The TSIZ[0:2] signals indicate the size of the requested data transfer. The TSIZ[0:2] signals may be used along with $\overline{\text{TBST}}$ and A[29:31] to determine which portion of the data bus contains valid data for a write transaction or which portion of the bus should contain valid data for a read transaction.

In 60x bus mode, the MPC7410 only supports bursting for cache block transfers (4 double words). In 60x bus mode, quad-word AltiVec loads and stores are split into two separate 8-byte, single-beat transactions on the system bus. Table 9-2 defines the $\overline{\text{TBST}}$ and TSIZ[0:2] encodings used by the MPC7410 in 60x bus mode.

Table 9-2. TBST and TSIZ[0:2] Encodings in 60x Bus Mode

$\overline{\text{TBST}}$	TSIZ0	TSIZ1	TSIZ2	Transfer Size[1]
Asserted	0	0	0	undefined
Asserted	0	0	1	undefined
Asserted	0	1	0	4 double-word burst
Asserted	0	1	1	undefined
Asserted	1	0	0	undefined
Asserted	1	0	1	undefined
Asserted	1	1	0	undefined
Asserted	1	1	1	undefined
Negated	0	0	0	8 bytes
Negated	0	0	1	1 byte
Negated	0	1	0	2 bytes
Negated	0	1	1	3 bytes
Negated	1	0	0	4 bytes
Negated	1	0	1	5 bytes (N/A)
Negated	1	1	0	6 bytes (N/A)
Negated	1	1	1	7 bytes (N/A)

[1] 3-byte transfers may be requested by the MPC7410 starting at any byte address within the double word from byte address 0 to byte address 5.
4-byte transfers may be requested by MPC7410 starting at any byte address within the double word from byte address 0 to byte address 4.

The basic coherency size of the bus is defined to be 32 bytes (corresponding to one cache line). Data transfers that cross an aligned, 32-byte boundary either must present a new address onto the bus at that boundary (for coherency consideration) or must operate as noncoherent data with respect to the MPC7410. The MPC7410 never generates a bus transaction with a transfer size of 5 bytes, 6 bytes, or 7 bytes.

For operations generated by the **eciwx**/**ecowx** instructions, a transfer size of 4 bytes is implied, and the $\overline{\text{TBST}}$ and TSIZ[0:2] signals are redefined to specify the resource ID (RID). The RID is copied from bits 28–31 of the external access register (EAR). For these operations, the $\overline{\text{TBST}}$ signal carries the EAR[28] data without inversion (active high).

9.3.2.2.3 Write-Through ($\overline{\text{WT}}$), Cache Inhibit ($\overline{\text{CI}}$), and Global ($\overline{\text{GBL}}$) Signals

In general, the MPC7410 provides the $\overline{\text{WT}}$, $\overline{\text{CI}}$, and $\overline{\text{GBL}}$ signals to indicate the status of a transaction target as determined by the WIM bit settings during address translation by the MMU. There are exceptions, as described in Section 3.9.2, "Transfer Attributes."

9.3.2.3 Burst Ordering During Data Transfers

During burst data transfer operations, 32 bytes of data (one cache line) are transferred to or from the cache in order. However, since burst reads are performed critical double word first, a burst read transfer may not start with the first double word of the cache line, and the cache line fill may wrap around the end of the cache line. Non-intervention burst write transfers (the only burst writes performed in 60x bus mode) are always performed zero double word first. Intervention burst writes (see Section 9.6.2, "Data Tenure in MPX Bus Mode,") are performed critical double word first.

The MPC7410 allows the transfer of any block of contiguous bytes within a double word. No single bus transactions may cross a double-word boundary. Transfers of strings of data that are aligned in such a way that they cross a double-word boundary must be broken down into multiple bus transactions.

Table 9-3 describes the order of the double words (DW) transferred during burst operations.

Table 9-3. Burst Ordering[1]

Data Transfer	For Starting Address:			
	A[27:28] = 00	A[27:28] = 01	A[27:28] = 10	A[27:28] = 11
First data beat	DW0	DW1	DW2	DW3
Second data beat	DW1	DW2	DW3	DW0
Third data beat	DW2	DW3	DW0	DW1
Fourth data beat	DW3	DW0	DW1	DW2

[1] A[29:31] are always 0b000 for burst transfers performed by the MPC7410.

9.3.2.4 Effect of Alignment in Data Transfers

Table 9-4 lists the aligned transfers that can occur on the MPC7410 bus. These are transfers in which the data is aligned to an address that is an integral multiple of the size of the data. For example, Table 9-4 shows that 1-byte data is always aligned; however, for a 4-byte word to be aligned, it must be oriented on an address that is a multiple of 4.

Table 9-4. Aligned Data Transfers

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

[1] A These entries indicate the byte portions of the requested operand that are read or written during that bus transaction.
—These entries are not required and are ignored during read transactions; they are driven with undefined data during all write transactions.

All transactions generated by AltiVec instructions must be aligned to a natural boundary. All burst transactions must be double-word aligned. Note that the MPC7410 does not work with the MPC106 bridge device in little-endian mode if misaligned data is accessed.

The MPC7410 supports misaligned memory operations, although their use may substantially degrade performance. Misaligned memory transfers address memory that is not aligned to the size of the data being transferred (such as, a word read of an odd byte address).

9.3.2.4.1 Misalignment Example

Although most operations hit in the primary cache (or generate burst memory operations if they miss), the MPC7410 interface supports misaligned transfers within a double-word (64-bit aligned) boundary, as shown in Table 9-5. Note that the 4-byte transfer in Table 9-5 is only one example of misalignment. As long as the attempted transfer does not cross a double-word boundary, the MPC7410 can transfer the data on the misaligned address (for example, a half-word read from an odd byte-aligned address). An attempt to address data that crosses a double-word boundary requires two bus transfers to access the data.

Due to the performance degradations, misaligned memory operations should be avoided. In addition to the double-word straddle boundary condition, the address translation logic can generate substantial exception overhead when the load/store multiple and load/store string instructions access misaligned data. It is strongly recommended that software attempt to align data where possible.

Table 9-5. Misaligned Data Transfers (Four-Byte Examples)

Transfer Size (Four Bytes)	TSIZ[0:2]	A[29:31]	Data Bus Byte Lanes[1]							
			0	1	2	3	4	5	6	7
Aligned	1 0 0	0 0 0	A	A	A	A	—	—	—	—
Misaligned	1 0 0	0 0 1	—	A	A	A	A	—	—	—
Misaligned	1 0 0	0 1 0	—	—	A	A	A	A	—	—
Misaligned	1 0 0	0 1 1	—	—	—	A	A	A	A	—
Aligned	1 0 0	1 0 0	—	—	—	—	A	A	A	A
Misaligned—First access	0 1 1	1 0 1	—	—	—	—	—	A	A	A
Second access	0 0 1	0 0 0	A	—	—	—	—	—	—	—
Misaligned—First access	0 1 0	1 1 0	—	—	—	—	—	—	A	A
Second access	0 1 0	0 0 0	A	A	—	—	—	—	—	—
Misaligned—First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
Second access	0 11	0 0 0	A	A	A	—	—	—	—	—

[1] A Byte lane used
—Byte lane not used

9.3.2.4.2 Alignment of External Control Instructions

The size of the data transfer associated with **eciwx** and **ecowx** instructions is always four bytes. If an operand for an **eciwx** or **ecowx** instruction is misaligned and crosses any word boundary, the MPC7410 generates an alignment exception.

9.3.3 Address Transfer Termination

The address tenure of a bus operation is terminated when completed with the assertion of $\overline{\text{AACK}}$ (address acknowledge), or retried with the assertion of $\overline{\text{ARTRY}}$ (address retry). The MPC7410 does not terminate the address transfer until the $\overline{\text{AACK}}$ input is asserted; therefore, the system can extend the address transfer phase by delaying the assertion of $\overline{\text{AACK}}$ to the MPC7410. The assertion of $\overline{\text{AACK}}$ can be as early as the bus clock cycle following $\overline{\text{TS}}$ (see Figure 9-7), which allows a minimum address tenure of two bus cycles.

Both ARTRY and SHD go through a precharge cycle unless this is disabled by setting the precharge disable bit in HID0. As shown in Figure 9-7, these signals are asserted for one bus clock cycle, released to high-impedance for half the next clock cycle, driven high for one clock cycle, and finally, released to high-impedance for the remaining half clock cycle. Note that AACK must be asserted for only one bus clock cycle.

The address transfer can be terminated with the requirement to retry if ARTRY is asserted anytime during the address tenure and through the cycle following AACK. The assertion of ARTRY causes the entire transaction (address and data tenure) to be rerun. As a snooping device, the MPC7410 asserts ARTRY for a snooped transaction that hits modified data in the data cache that must be written back to memory, or if the snooped transaction could not be serviced. As a bus master, the MPC7410 responds to the assertion of ARTRY by aborting the bus transaction and re-requesting the bus. Note that after recognizing the assertion of ARTRY and aborting the transaction in progress, the MPC7410 is not guaranteed to run the same transaction the next time it is granted the bus due to internal reordering of load and store operations.

Note that the MPC7410 implements a shared state (MESI instead of MEI) cache coherency protocol based on the setting of MSSCR0[SHDEN]. See Section 2.1.5.3, “Memory Subsystem Control Register (MSSCR0),” for more detailed information about the bits of MSSCR0.

9.3.3.1 Address Retry Window and Qualified ARTRY

If an address retry is required, ARTRY is asserted by a bus snooping device as early as the second cycle after the assertion of TS. Note that the MPC7410 is self-snooping and may assert ARTRY for its own transaction. See Section 3.9.3, “Snooping,” for more information. Once asserted, ARTRY must remain asserted through the cycle after the assertion of AACK; the bus clock cycle starting two clock cycles after TS and ending with the cycle after the assertion of AACK is referred to as the address retry window.

The assertion of ARTRY during the cycle after the assertion of AACK is referred to as a qualified ARTRY. An earlier assertion of ARTRY during the address tenure is referred to as an early ARTRY.

As a bus master, the MPC7410 recognizes either an early or qualified ARTRY and preempts the data tenure associated with the retried address tenure. If the data tenure has already begun, the MPC7410 aborts and terminates the data tenure immediately even if the burst data has been received. If the assertion of ARTRY is received up to or on the bus cycle of the first (or only) assertion of TA for the data tenure, the MPC7410 ignores the first data beat, and if it is a load operation, does not forward data internally to the cache and execution units. If ARTRY is asserted after the first (or only) assertion of TA, improper operation of the bus interface may result.

9.3.3.2 Snoop Copyback and Window of Opportunity

During the clock cycle of a qualified ARTRY, the MPC7410 also determines if it should negate BR and ignore BG on the following cycle. On the following cycle, all other bus devices negate address bus requests, and they do not qualify address bus grants. This cycle is the window of opportunity for the snooping master that asserted ARTRY and needs to perform a snoop copyback operation. Thus, the snooping master that asserted ARTRY is the only device allowed to assert BR. Note that a nonclocked bus arbiter may detect the assertion of address bus request by the bus master that asserted ARTRY, and return a qualified bus grant one cycle earlier than shown in Figure 9-7.

When the MPC7410 asserts $\overline{\text{ARTRY}}$ due to a snoop operation and is ready to perform the snoop push, it always asserts $\overline{\text{BR}}$ in the window of opportunity to obtain bus mastership for the copyback cycle. (A copyback operation due to a snoop hit to a modified block is sometimes referred to as a snoop push.) Note that the copyback is a non-global ($\overline{\text{GBL}}$ negated) transaction. External devices on the 60x bus must not assert $\overline{\text{ARTRY}}$ for non-global transactions.

Note that even if the MPC7410 asserts $\overline{\text{BR}}$ in the window of opportunity for a snoop push, it may be several bus cycles later before the MPC7410 is able to perform the necessary transaction. The timing of $\overline{\text{TS}}$ may be dependent on resource constraints and may require forward progress on the data bus. The bus arbiter should keep $\overline{\text{BG}}$ asserted until it detects that $\overline{\text{BR}}$ is negated or $\overline{\text{TS}}$ is asserted from the MPC7410 indicating that the snoop copyback has begun. The system should ensure that no other address tenures occur until the current snoop push from the MPC7410 is completed.

It may occur in some systems that the MPC7410 was unable to perform a pending snoop copyback when a new snoop operation is performed. In this case, the MPC7410 requests the bus in the window of opportunity if it hits on the new snooped address, and it performs the snoop copyback operation for the earlier snooped address rather than for the current snooped address in order to clear its internal snoop queue.

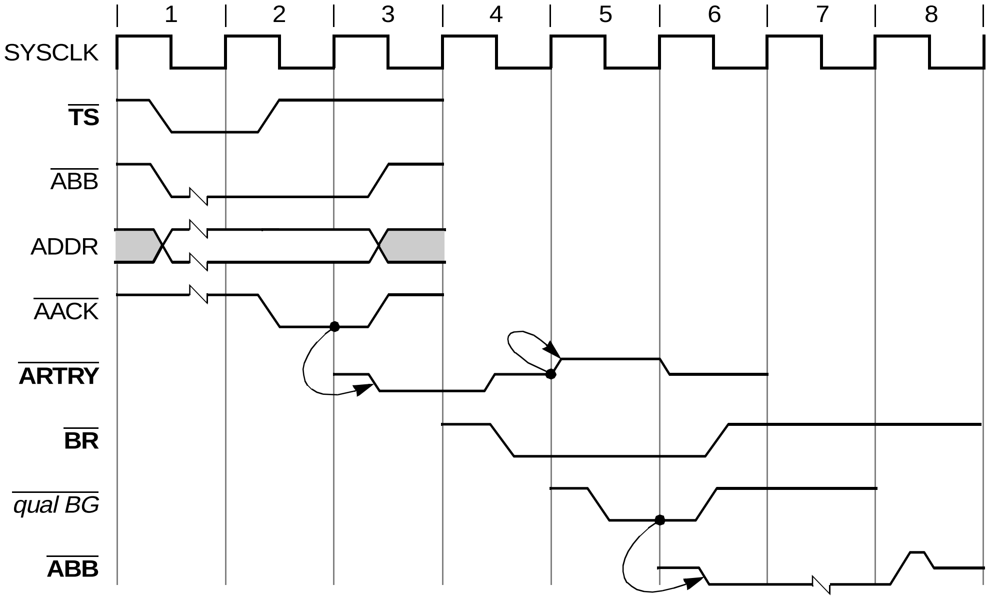


Figure 9-7. Snooped Address Cycle with $\overline{\text{ARTRY}}$

9.3.3.3 Snoop Response and $\overline{\text{SHD}}$ Signal

The MPC7410 asserts the $\overline{\text{SHD}}$ signal as an output coincident with the $\overline{\text{ARTRY}}$ output signal if the cache block that caused a snoop hit is pushed as the processor’s next address transaction.

9.4 60x Data Bus Tenure

This section describes the data bus arbitration, transfer, and termination phases defined by the 60x bus protocol used by the MPC7410. The phases of the data tenure are identical to those of the address tenure, underscoring the symmetry in the control of the two buses.

9.4.1 Data Bus Arbitration

Data bus arbitration uses the data arbitration signal group—$\overline{\text{DBG}}$, $\overline{\text{DBWO}}$, and $\overline{\text{DBB}}$. Additionally, the combination of $\overline{\text{TS}}$ and TT[0:4] provides information about the data bus request to external logic.

9.4.1.1 Qualified Data Bus Grant in 60x Bus Mode

The $\overline{\text{TS}}$ signal is an implied data bus request from the MPC7410; the arbiter must qualify $\overline{\text{TS}}$ with the transfer type (TTx) encodings to determine if the current address transfer is an address-only operation, which does not require a data bus transfer. If the data bus is needed, the arbiter grants data bus mastership by asserting the $\overline{\text{DBG}}$ input to the MPC7410. As with the address bus arbitration phase, the MPC7410 must qualify the $\overline{\text{DBG}}$ input with a number of input signals before assuming bus mastership

When the MPC7410 is operating in 60x bus mode, a qualified data bus grant occurs when the following conditions are satisfied:

- $\overline{\text{DBG}}$ is asserted
- $\overline{\text{ARTRY}}$ is not asserted in the address retry window for the address phase of this transaction
- The processor is ready to begin a data transaction
- The processor is not already using the data bus

A data bus grant may be qualified as early as the clock cycle when $\overline{\text{TS}}$ is asserted if $\overline{\text{DBG}}$ is asserted during that cycle (for example, when the data bus is parked). However, the requirement remains that the first (or only) assertion of $\overline{\text{TA}}$ for the transaction must occur no sooner than two clock cycles after $\overline{\text{TS}}$ is asserted. This requirement is necessary to guarantee that the first (or only) assertion of $\overline{\text{TA}}$ will not occur before a possible $\overline{\text{ARTRY}}$ response. See Section 9.3.3.1, "Address Retry Window and Qualified ARTRY," for more information about $\overline{\text{ARTRY}}$ signal requirements.

When a data tenure overlaps with its associated address tenure, a qualified $\overline{\text{ARTRY}}$ assertion coincident with a data bus grant signal does not result in data bus mastership ($\overline{\text{DBB}}$ is not asserted). Otherwise, the MPC7410 always asserts $\overline{\text{DBB}}$ on the bus clock cycle after recognition of a qualified data bus grant. Because the MPC7410 can pipeline transactions, there may be an outstanding data bus transaction when a new address transaction is retried. In this case, the MPC7410 becomes the data bus master to complete the previous transaction and the $\overline{\text{ARTRY}}$ does not affect the data tenure.

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

The 60x bus protocol allows for masters that do not implement the $\overline{\text{DBB}}$ signal as an input, such as the MPC7410. The elimination of $\overline{\text{DBB}}$ from the MPC7410 interfaces removes logic from critical timing paths in the processor interface, allowing higher frequency bus operation. However, this puts more responsibility on the data bus arbiter. The memory system can control data tenure scheduling directly with $\overline{\text{DBG}}$. It is possible to ignore the $\overline{\text{DBB}}$ signal in the system if the $\overline{\text{DBB}}$ input is not used as the final data bus allocation control between data bus masters, and if the memory system can track the start and end of the data tenure. If $\overline{\text{DBB}}$ is not used to signal the end of a data tenure, $\overline{\text{DBG}}$ is only asserted to the next bus master the cycle before the cycle that the next bus master may actually begin its data tenure. Even if $\overline{\text{DBB}}$ is ignored in the system, the MPC7410 always recognizes its own assertion of $\overline{\text{DBB}}$, and requires one cycle after data tenure completion to negate its own $\overline{\text{DBB}}$ before recognizing a qualified data bus grant for another data tenure. If $\overline{\text{DBB}}$ is ignored in the system, it must still be connected to a pull-up resistor on the MPC7410 to ensure proper operation.

9.4.1.3 Data Bus Write Only ($\overline{\text{DBWO}}$) and Data Bus Arbitration

As a result of address pipelining, the MPC7410 may have up to six data tenures queued to perform when it receives a qualified $\overline{\text{DBG}}$. Generally, the data tenures should be performed in strict order (the same order) as their address tenures were performed. The MPC7410, however, also supports a limited out-of-order capability with the data bus write only ($\overline{\text{DBWO}}$) input signal. When recognized on the clock of a qualified $\overline{\text{DBG}}$, $\overline{\text{DBWO}}$ may direct the MPC7410 to perform the next pending data write tenure even if a pending read tenure would have normally been performed first. For more information on the operation of $\overline{\text{DBWO}}$, refer to Section 9.4.4, “Using Data Bus Write Only (DBWO).”

If the MPC7410 has any data tenures to perform, it always accepts data bus mastership to perform a data tenure when it recognizes a qualified $\overline{\text{DBG}}$. If $\overline{\text{DBWO}}$ is asserted with a qualified $\overline{\text{DBG}}$ and no write tenure is queued to run, the MPC7410 still assumes mastership of the data bus to perform the next pending read data tenure.

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

9.4.2 Data Transfer Signals and Protocol

The data transfer signals include DH[0:31], DL[0:31], and DP[0:7]. For memory accesses, the DH and DL signals form a 64-bit data path for read and write operations.

The MPC7410 transfers data in either single- or 4-beat burst transfers. Single-beat transactions represent caching-inhibited or write-through operations and they can transfer from 1 to 8 bytes at a time. They can also be misaligned; see Section 9.3.2.4, “Effect of Alignment in Data Transfers.” Burst operations always transfer eight words and are aligned on eight-word address boundaries. In 60x bus mode, 128-bit, caching-inhibited or write-through, AltiVec loads or stores are broken into two separate double-word transactions. The four double-word burst transaction is used for transferring a cache block. Burst transfers can achieve significantly higher bus throughput than single-beat operations.

The type of transaction initiated by the MPC7410 depends on whether the code or data is caching-inhibited or caching-allowed and, for store operations, whether the cache is in write-back or write-through mode which is controlled by software on either a page or block basis. Burst transfers support caching-allowed operations only; that is, memory structures must be marked as caching-allowed (and write-back for data store operations) in the respective page or block descriptor to take advantage of burst transfers.

The MPC7410 $\overline{\text{TBST}}$ output indicates to the system whether the current transaction is a single- or four-beat transfer (except during **eciwx**/**ecowx** transactions, when it signals the state of EAR[28]). A burst transfer has an assumed address order. For load or store operations that miss in the cache (and are marked as caching-allowed and, for stores, write-back in the MMU), the MPC7410 uses the double-word-aligned address associated with the critical code or data that initiated the transaction. This minimizes latency by allowing the critical code or data to be forwarded to the processor before the rest of the cache line is filled. For all other burst operations, however, the cache line is transferred beginning with the eight-word-aligned data.

Because there is not a $\overline{\text{DRTRY}}$ mode in the MPC7410, data must never be transferred before the first cycle of the system's address retry window. That is, valid data must never precede a possible $\overline{\text{ARTRY}}$ for that transaction.

9.4.3 Data Transfer Termination

The 60x bus interface defines four signals to terminate data bus transactions—$\overline{\text{TA}}$, $\overline{\text{DRTRY}}$ (data retry), $\overline{\text{TEA}}$ (transfer error acknowledge), and $\overline{\text{ARTRY}}$. Note that the MPC7410 does not implement the $\overline{\text{DRTRY}}$ signal; therefore, the 60x interface on the MPC7410 is always operating in the higher-performance, no-$\overline{\text{DRTRY}}$ mode.

The $\overline{\text{TA}}$ signal indicates normal termination of data transactions. It must always be asserted on the bus cycle coincident with the data it is qualifying. It may be withheld by the slave for any number of clocks until valid data is ready to be supplied or accepted.

If the $\overline{\text{ARTRY}}$ and $\overline{\text{TEA}}$ signals are asserted in the same clock, the $\overline{\text{ARTRY}}$ signal takes precedence and the $\overline{\text{TEA}}$ signal is ignored. This means that the transaction is repeated until the $\overline{\text{ARTRY}}$ condition is resolved.

An assertion of $\overline{\text{ARTRY}}$ causes the data tenure to be terminated immediately if the $\overline{\text{ARTRY}}$ for the address tenure is associated with the data tenure in operation. If $\overline{\text{ARTRY}}$ is connected for the MPC7410, the earliest allowable assertion of $\overline{\text{TA}}$ to the MPC7410 is directly dependent on the earliest possible assertion of $\overline{\text{ARTRY}}$ to the MPC7410; see Section 9.3.3, "Address Transfer Termination."

The $\overline{\text{TEA}}$ input is used to signal a nonrecoverable error during the data transaction. It may be asserted on any cycle during $\overline{\text{DBB}}$. The assertion of $\overline{\text{TEA}}$ terminates the data tenure immediately, even if in the middle of a burst; however, it does not prevent incorrect data that has just been acknowledged with $\overline{\text{TA}}$ from being written into the MPC7410's cache or GPRs. The assertion of $\overline{\text{TEA}}$ initiates either a machine check exception or a checkstop condition based on the setting of the MSR[ME] bit.

Upon receiving a final (or only) termination condition, the MPC7410 always negates $\overline{\text{DBB}}$ for one cycle.

9.4.3.1 Normal Single-Beat Termination

Normal termination of a single-beat data read operation occurs when TA is asserted by a responding slave. The TEA signal must remain negated during the transfer (see Figure 9-8).

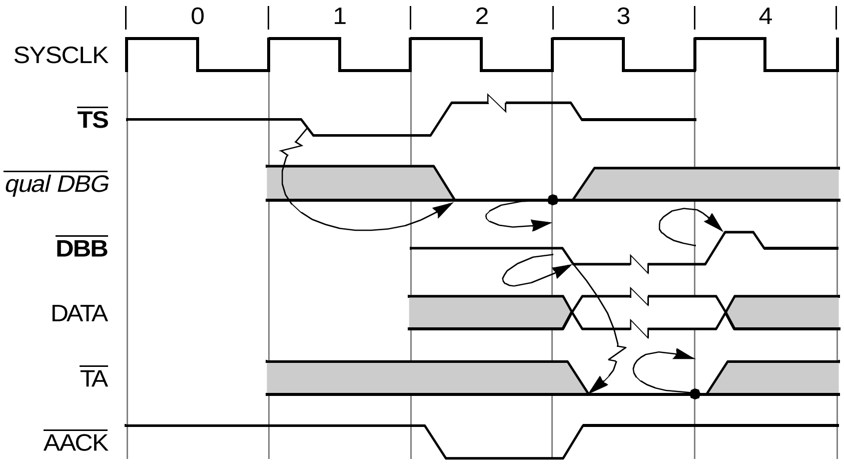


Figure 9-8. Normal Single-Beat Read Termination

Figure 9-9 shows a write operation with a normal termination.

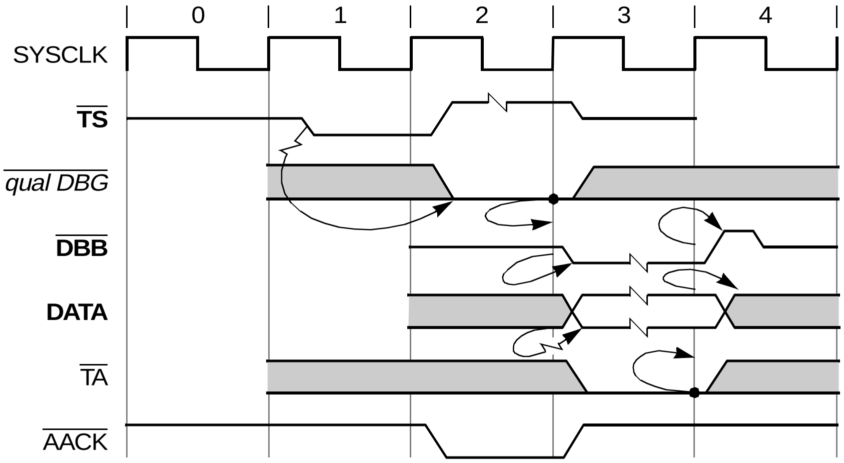


Figure 9-9. Normal Single-Beat Write Termination

Normal termination of a burst transfer occurs when TA is asserted for four bus clock cycles, as shown in Figure 9-10. The bus clock cycles in which TA is asserted need not be consecutive, thus allowing pacing of the data transfer beats. For read bursts to terminate successfully, TEA must remain negated during the transfer. For write bursts, TEA must remain negated for a successful transfer.

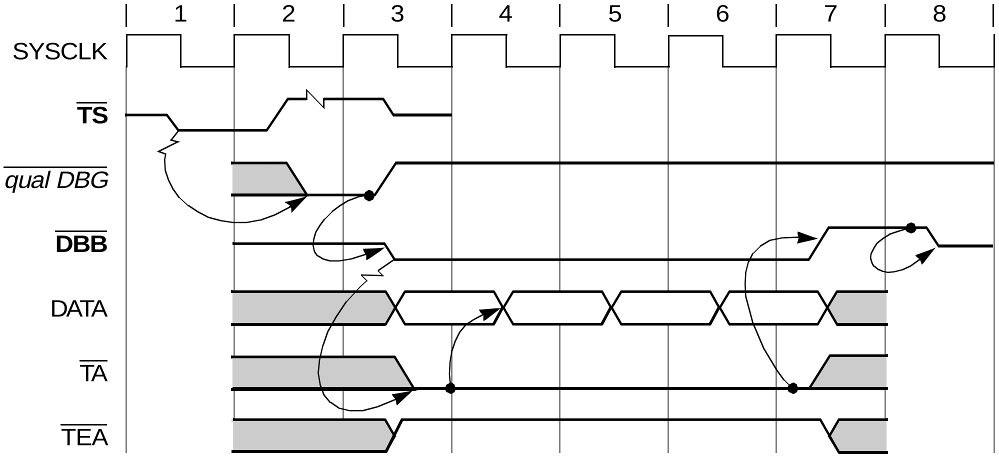


Figure 9-10. Normal Burst Transaction

Figure 9-11 shows the effect of using TA to pace the data transfer rate. Notice that in bus clock cycle 4 of Figure 9-11, TA is negated for the second data beat. The MPC7410 data pipeline does not proceed until bus clock cycle 5 when TA is reasserted.

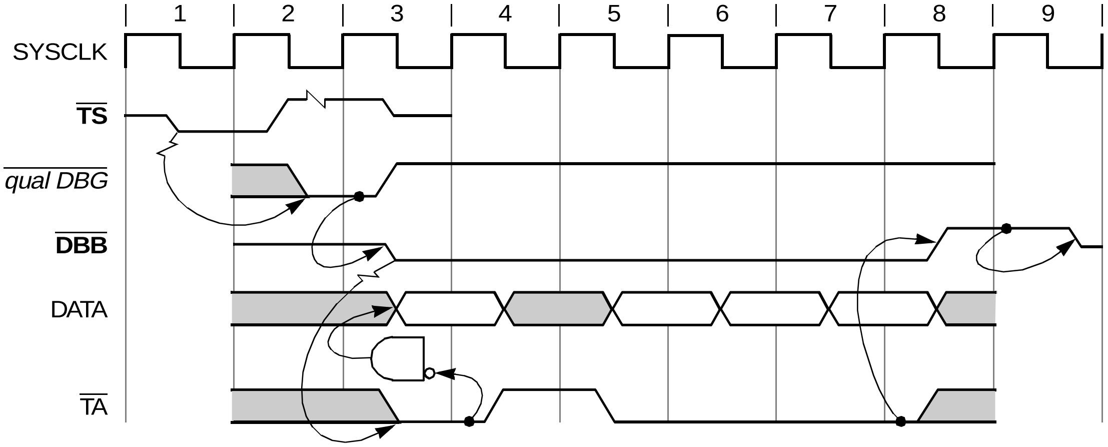


Figure 9-11. Read Burst with TA Wait States

9.4.3.2 Data Transfer Termination Due to a Bus Error

The TEA signal indicates that a bus error occurred. It may be asserted while DBB is asserted. Asserting TEA to the MPC7410 terminates the transaction; that is, subsequent assertions of TA are ignored and DBB is negated.

Assertion of the TEA signal causes a machine check exception (and possibly a checkstop condition within the MPC7410). For more information, see Section 4.6.2, “Machine Check Exception (0x00200).” Note that the MPC7410 does not implement a synchronous error capability for memory accesses. This means that the exception instruction pointer saved into the SRR0 register does not point to the memory operation that caused the assertion of $\overline{\text{TEA}}$, but to the instruction about to be executed (perhaps several instructions later). Also note that assertion of $\overline{\text{TEA}}$ does not invalidate data entering the GPR or the cache. Additionally, the address corresponding to the access that caused $\overline{\text{TEA}}$ to be asserted is not latched by the MPC7410. To recover, the exception handler must determine and remedy the cause of the $\overline{\text{TEA}}$, or the MPC7410 must be reset; therefore, this function should only be used to indicate fatal system conditions to the processor (such as parity or uncorrectable ECC errors).

After the MPC7410 has committed to run a transaction, that transaction must eventually complete. Address retry causes the transaction to be restarted; $\overline{\text{TA}}$ wait states delay termination of individual data beats. Eventually, however, the system must either terminate the transaction or assert the $\overline{\text{TEA}}$ signal. For this reason, care must be taken to check for the end of physical memory and the location of certain system facilities to avoid memory accesses that result in the assertion of $\overline{\text{TEA}}$.

Note that $\overline{\text{TEA}}$ generates a machine check exception depending on MSR[ME]. Clearing the machine check exception enable control bits leads to a true checkstop condition (instruction execution halted and processor clock stopped).

9.4.3.3 No-$\overline{\text{DRTRY}}$ Mode

The MPC7410 disallows the use of the data retry function provided by some 60x processors, and no-$\overline{\text{DRTRY}}$ mode is always selected. The no-$\overline{\text{DRTRY}}$ mode provides higher performance because it allows the forwarding of data during load operations to the internal CPU one bus cycle sooner than in the standard 60x bus protocol that implements $\overline{\text{DRTRY}}$.

Because the MPC7410 always operates in no-$\overline{\text{DRTRY}}$ mode, the assertion of $\overline{\text{ARTRY}}$ by a snooping device must occur prior to or coincident with the first assertion of $\overline{\text{TA}}$ to the MPC7410; assertion of $\overline{\text{ARTRY}}$ must never occur on the cycle after the first assertion of $\overline{\text{TA}}$.

9.4.4 Using Data Bus Write Only (DBWO)

The MPC7410 supports split-bus pipelined transactions and a limited out-of-order capability for its own pipelined transactions through the data bus write only ($\overline{\text{DBWO}}$) signal. When recognized on the clock of a qualified $\overline{\text{DBG}}$, the assertion of $\overline{\text{DBWO}}$ directs the MPC7410 to perform the next pending data write tenure (if any), even if a pending read tenure would have normally been performed because of address pipelining. The $\overline{\text{DBWO}}$ signal does not change the order of write tenures with respect to other write tenures from the same MPC7410. It only allows that a write tenure be performed ahead of a pending read tenure from the same MPC7410.

Note that $\overline{\text{DBWO}}$ can be asserted if no data bus read is pending, but it has no effect on write ordering.

The arbiter must monitor bus operations and coordinate the various masters and slaves with respect to the use of the data bus when $\overline{\text{DBWO}}$ is used. Individual $\overline{\text{DBG}}$ signals associated with each bus device should allow the arbiter to synchronize both pipelined and split-transaction bus organizations. Individual $\overline{\text{DBG}}$ and $\overline{\text{DBWO}}$ signals provide a primitive form of source-level tagging for the granting of the data bus.

Note that use of the $\overline{\text{DBWO}}$ signal allows some operation-level tagging with respect to the MPC7410 and the use of the data bus.

9.5 60x Bus Timing Examples

This section shows timing diagrams for various scenarios using the 60x bus interface. Figure 9-12 illustrates the fastest single-beat reads possible for the MPC7410 and shows both minimal latency and maximum single-beat throughput. By delaying the data bus tenure, the latency increases, but, because of split-transaction pipelining, the overall throughput is not affected unless the data bus latency causes the third address tenure to be delayed.

Note that all bidirectional signals are released to high-impedance between bus tenures.

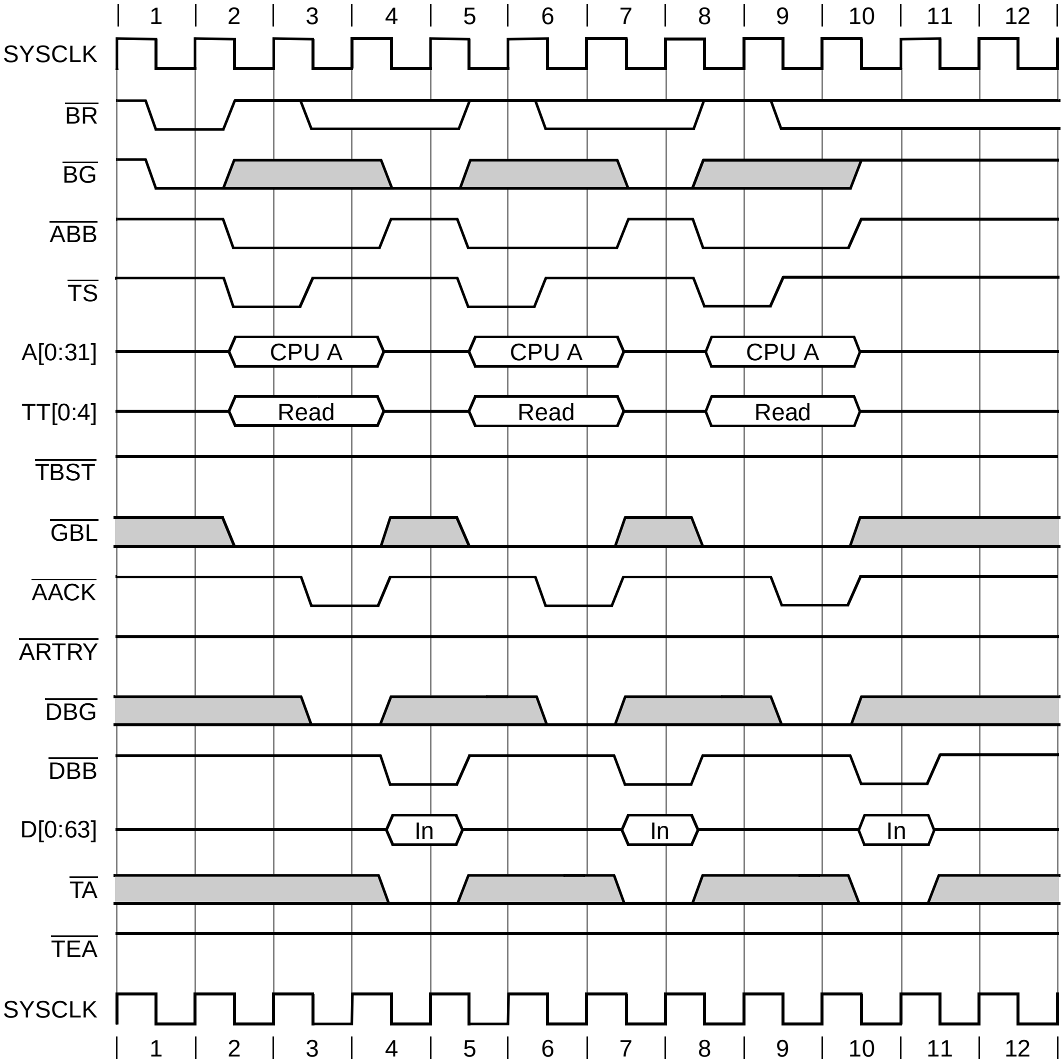


Figure 9-12. Fastest Single-Beat Reads

Figure 9-13 illustrates the fastest single-beat writes supported by the MPC7410. All bidirectional signals are released to high-impedance between bus tenures.

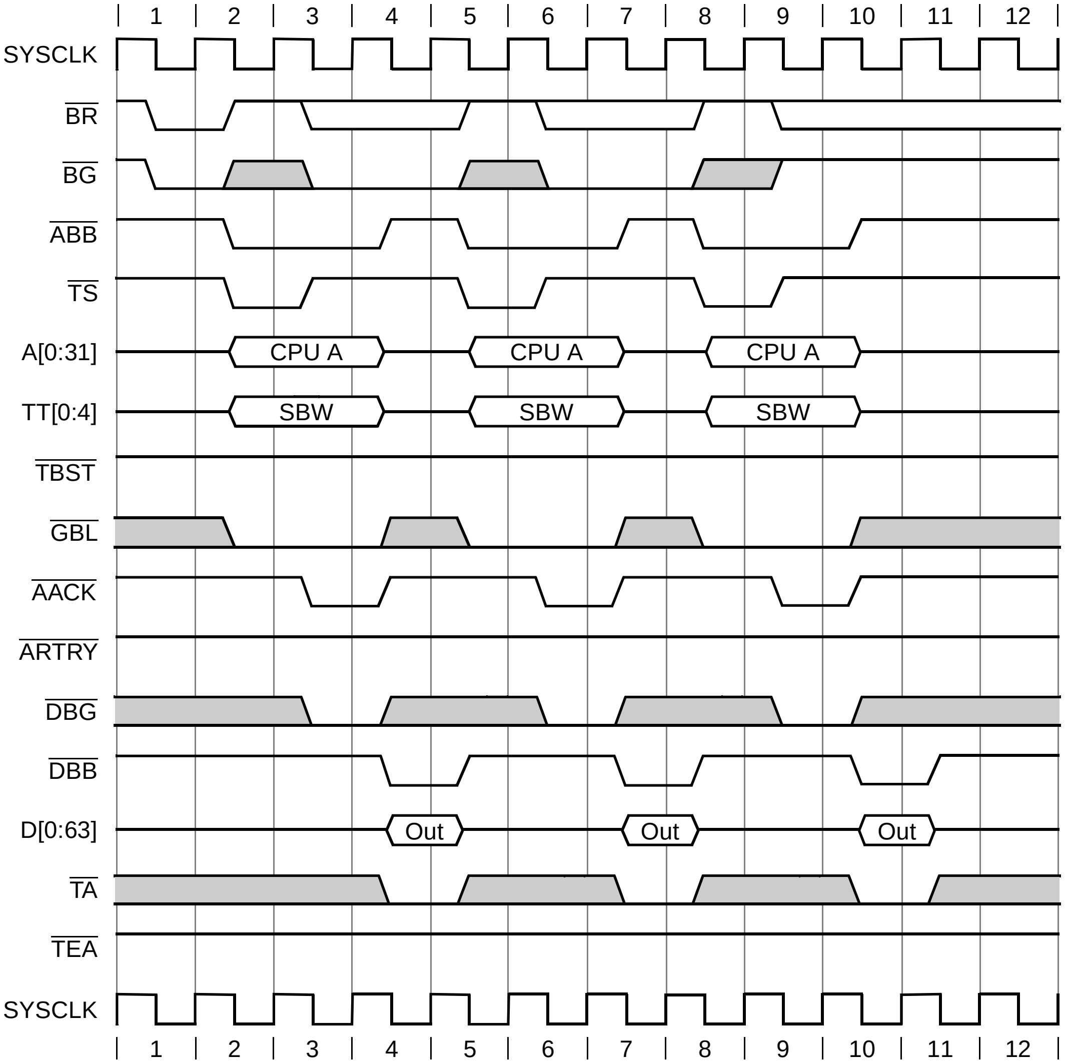


Figure 9-13. Fastest Single-Beat Writes

Figure 9-14 shows two ways that single-beat reads are delayed:

- The $\overline{\text{TA}}$ signal is negated to insert wait states in clock cycles 4 and 5.
- For the second access, $\overline{\text{DBG}}$ is delayed until clock cycle 8 (could have been asserted in clock cycle 7).

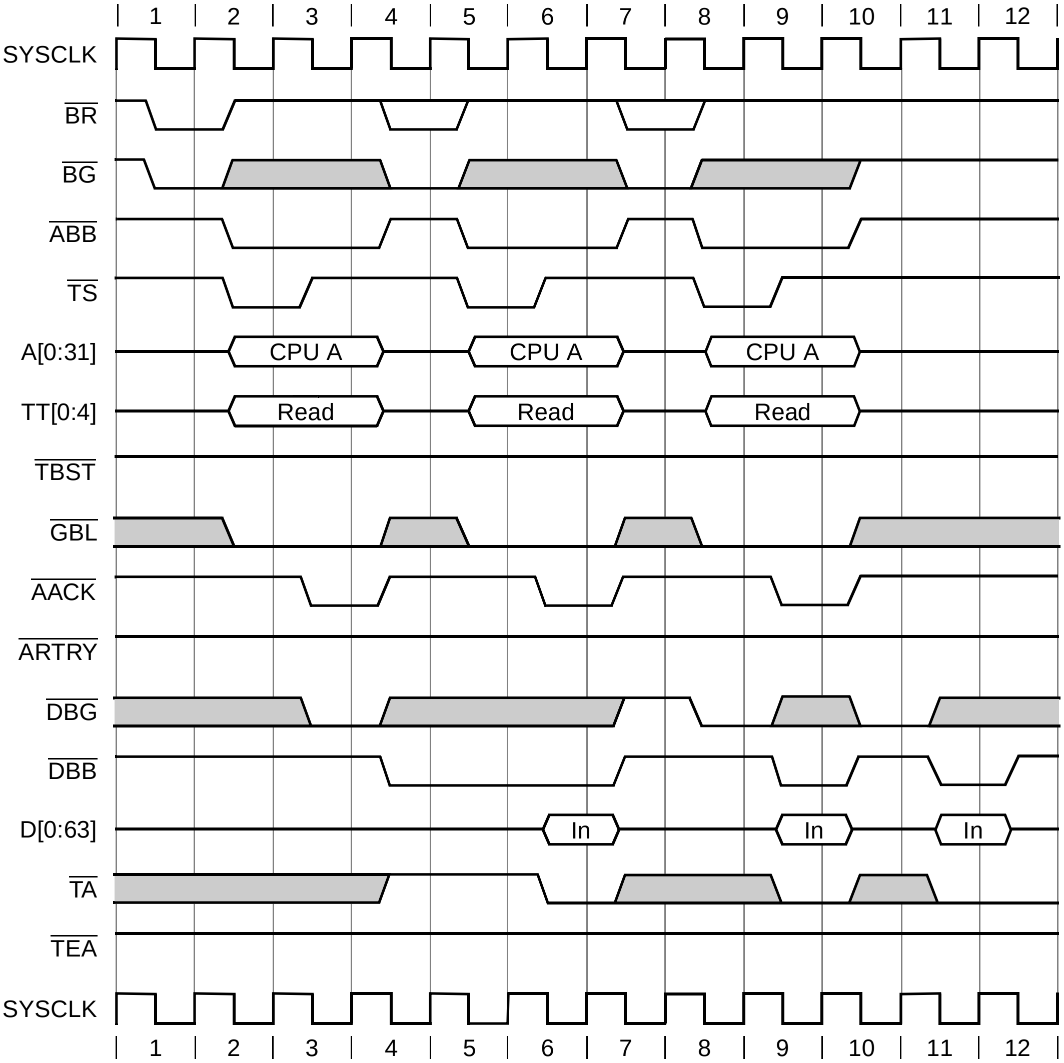


Figure 9-14. Single-Beat Reads Showing Data-Delay Controls

Figure 9-15 shows data-delay controls in a single-beat write operation. Note that all bidirectional signals are released to high-impedance between bus tenures. Data transfers are delayed in the following ways:

- The $\overline{\text{TA}}$ signal is held negated to insert wait states in clocks 4 and 5.
- In clock 7, $\overline{\text{DBG}}$ is held negated, delaying the start of the data tenure.

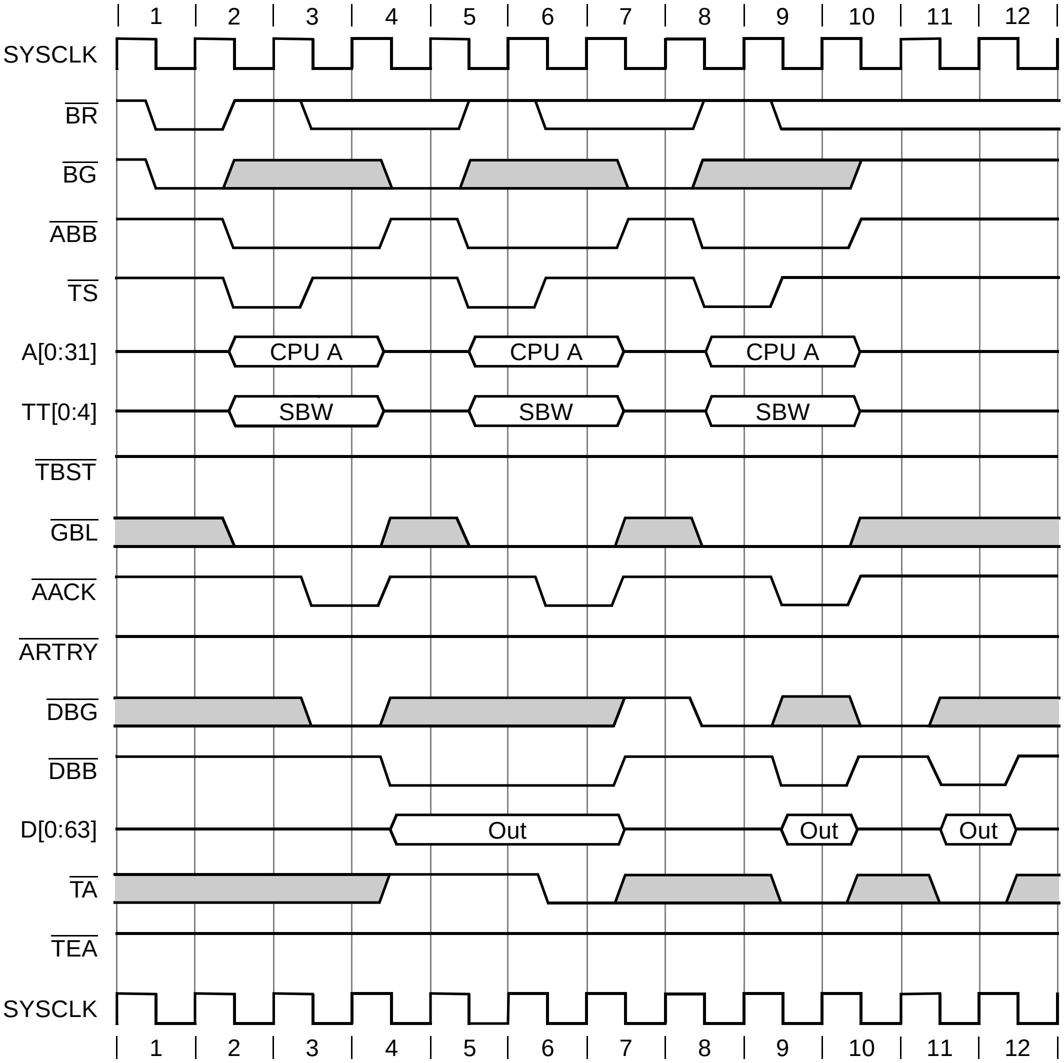


Figure 9-15. Single-Beat Writes Showing Data Delay Controls

Figure 9-16 shows the use of data-delay controls with burst transfers. Note that all bidirectional signals are released to high-impedance between bus tenures. Burst transfers are delayed in the following ways:

- The first data beat of bursted read data (clock 4) is the critical quad word.
- The write burst shows the use of TA signal negation to delay the third data beat.

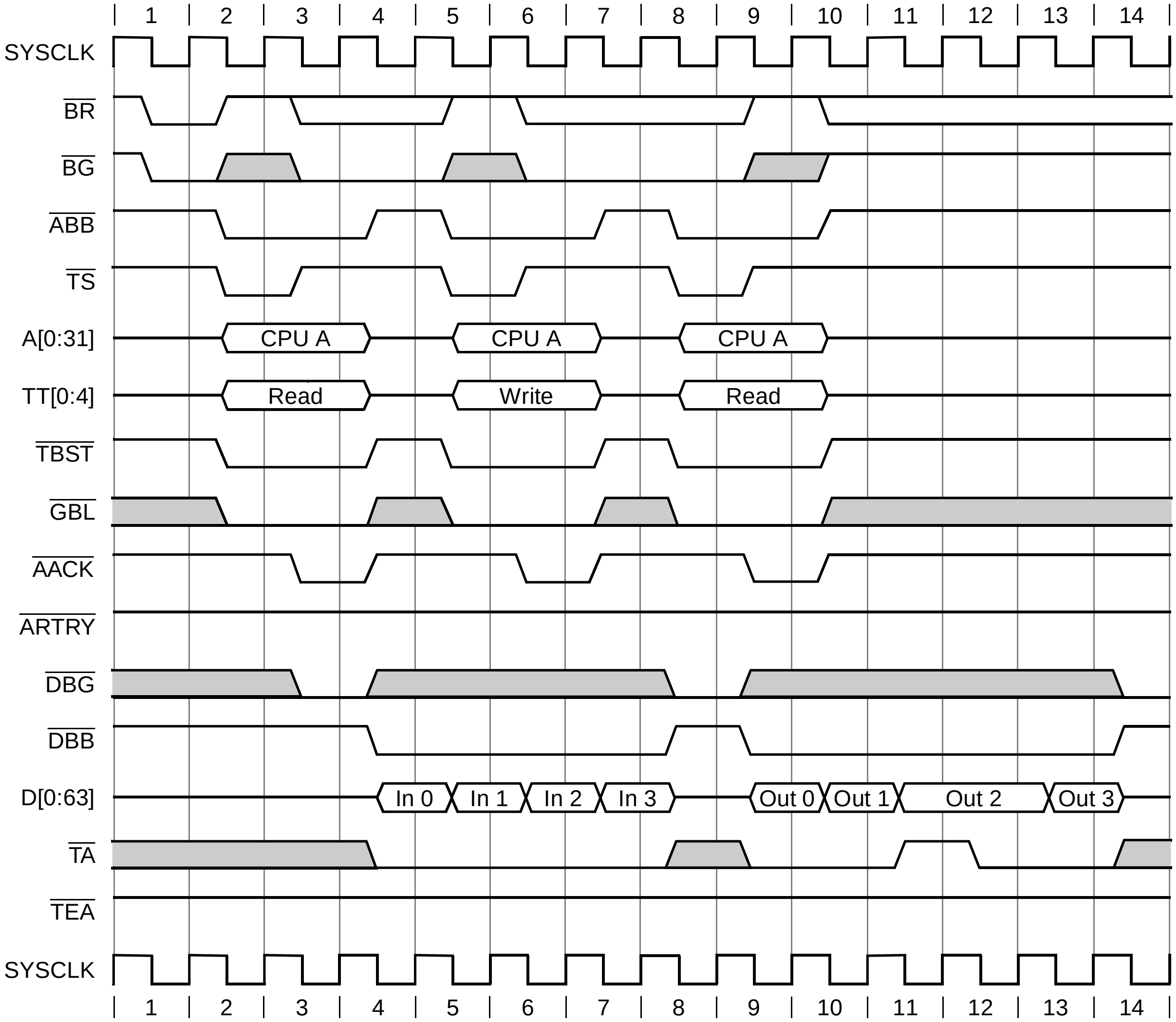


Figure 9-16. Burst Transfers with Data Delay Controls

Figure 9-17 shows the use of the TEA signal. Note that all bidirectional signals are released to high-impedance between bus tenures. Note the following:

- The first data beat of the read burst (in clock 4) is the critical quad word.
- The TEA signal truncates the burst write transfer on the third data beat.
- The TEA eventually causes the MPC7410 to take an exception.

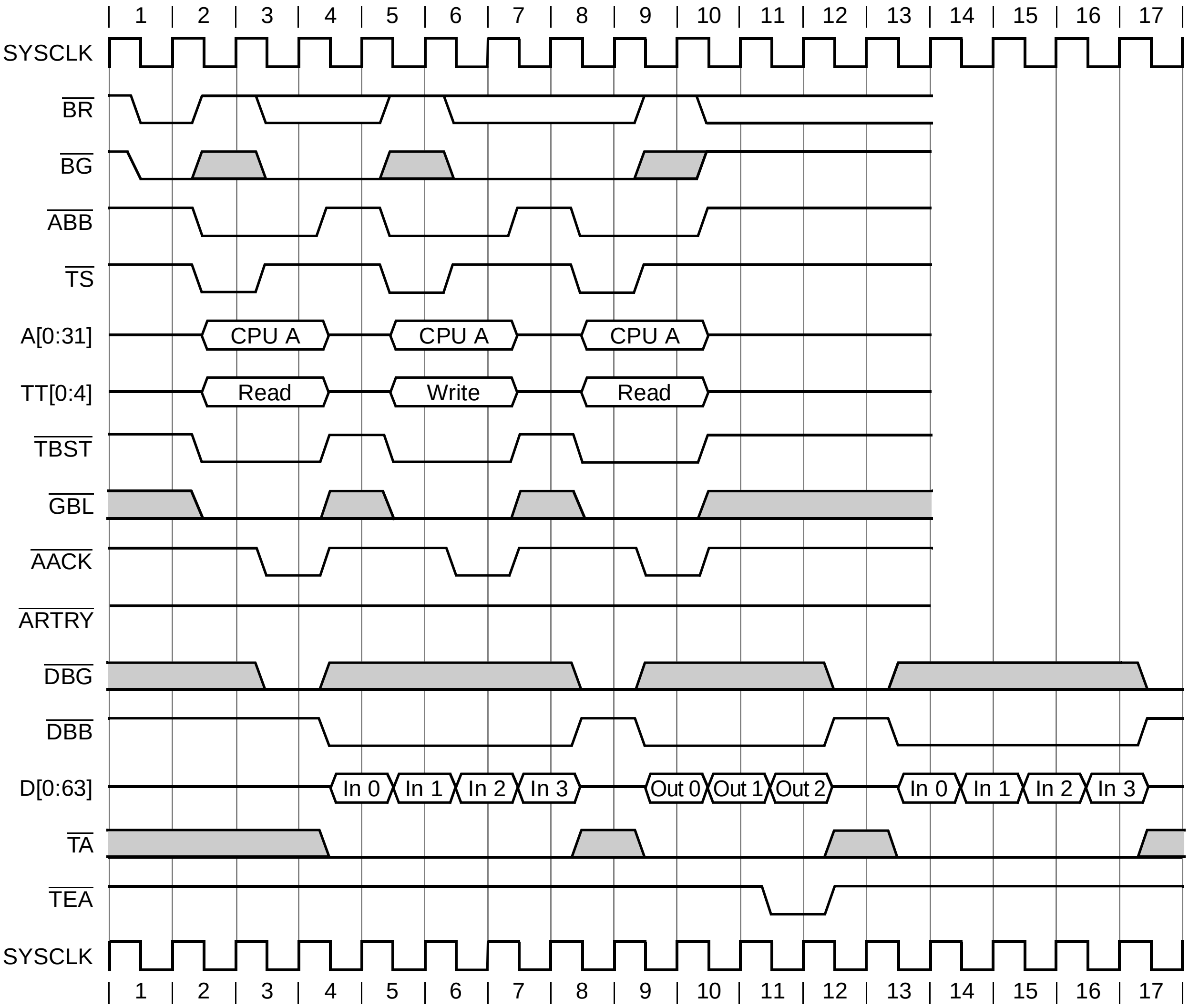


Figure 9-17. Use of Transfer Error Acknowledge (TEA)

9.6 MPX Bus Protocol

The MPX bus protocol is based on the 60x bus protocol, except that it includes several additional features that allow it to provide higher memory bandwidth than the 60x bus, and more efficient utilization of the system bus in an multiprocessing environment.

As described before, the value of the EMODE signal at the negation of HRESET determines whether the MPC7410 operates in 60x bus mode or MPX bus mode. This value is stored and readable from the

EMODE bit in MSSCR0. The state of MSSR0[EMODE] is active high, meaning that if $\overline{\text{EMODE}}$ is detected as asserted at the negation of $\overline{\text{HRESET}}$, MSSR0[EMODE] = 1 and MPX bus mode is selected; otherwise, MSSR0[EMODE] = 0 and 60x bus mode is selected.

All signals for the MPX interface are specified with respect to the rising-edge of the external system clock input (SYSCLK), and they are guaranteed to be sampled as inputs or changed as outputs with respect to that edge.

Since the same clock edge is referenced for driving or sampling the bus signals, the possibility of clock skew could exist between various modules in a system due to routing or the use of multiple clock lines. It is the responsibility of the system to handle any such clock skew problems that might occur.

The following sections provide a description of how the MPX interface operates and notes or points to consider when designing a system with it.

In addition to the basic transfer protocol of the 60x bus interface, the MPX bus also supports a data-only transfer in order to support cache-to-cache transfers (or data intervention) and a local bus slave. The $\overline{\text{HIT}}$ and $\overline{\text{DRDY}}$ signals are defined in the MPX bus protocol to support this type of transfer.

9.6.1 Address Tenure in MPX Bus Mode

The address tenure in MPX bus mode is very similar to that in 60x bus mode, except for those differences outlined in the following subsections.

An address retry capability, similar to 60x bus mode, is provided to support the snoop coherency protocol. In MPX bus mode, the MPC7410 additionally supports data intervention for more efficient coherency management. Address retry is used only when $\overline{\text{HIT}}$-style data intervention is not enabled or in those cases where $\overline{\text{HIT}}$-style data intervention is not allowed. An address retry response is issued by a snooping master in order to interrupt another master’s transaction on the bus, usually to write back that memory which it has modified in its own cache. The address retry causes the original master to abort the current transaction and rerun the transaction at a later time.

9.6.1.1 Address Arbitration Phase

Address bus arbitration in MPX mode differs from arbitration in the 60x interface in that the MPC7410 can drive consecutive address tenures without a dead cycle on the address bus in MPX bus mode if those address tenures are from the same processor. The MPC7410 does not use the $\overline{\text{ABB}}$ signal as an input (in 60x or MPX bus mode). When configured for MPX bus mode, the MPC7410 does provide an indication when the address bus is busy on the $\overline{\text{AMON}}$ output signal. Note that this signal is not a requirement of the MPX bus protocol and may not be available on future products.

9.6.1.1.1 Qualified Bus Grant in MPX Bus Mode

In MPX bus mode, the MPC7410 has a different equation from 60x bus mode for the qualification of $\overline{\text{BG}}$. The internally-generated address bus busy ($\overline{abb}$) is not used to qualify $\overline{\text{BG}}$. This removes the timing dependence of a qualified bus grant on the AACK signal, which is used to terminate the internally-generated $\overline{abb}$ in 60x bus mode. However, this means that in MPX bus mode the system arbiter must not assert $\overline{\text{BG}}$ to a second master until the cycle after $\overline{\text{AACK}}$ is asserted to end an address tenure from the first master. Another consequence of removing the dependence on the internally-generated $\overline{abb}$ is that $\overline{\text{BG}}$ must be negated in every cycle that $\overline{\text{AACK}}$ is delayed from the clock cycle after the assertion of $\overline{\text{TS}}$. The $\overline{\text{TS}}$ signal is still used to qualify bus grant in order to optimize speculative bus parking.

To gain access to the address bus, a bus master asserts $\overline{\text{BR}}$ and holds it asserted until it detects a qualified bus grant (QBG). The equation for a qualified bus grant for the MPC7410 in MPX bus mode is:

QBG = $\overline{\text{BG}}$ & ¬$\overline{\text{ARTRY}}$ & ¬$\overline{\text{TS}}$ &¬ (latched state variables)

where “latched state variables” include latched $\overline{\text{ARTRY}}$. Thus, a qualified bus grant occurs when $\overline{\text{BG}}$ is asserted, $\overline{\text{ARTRY}}$ is not asserted in the current or the preceding cycle, and $\overline{\text{TS}}$ is not asserted by this or any other processor.

A typical arbitration sequence is illustrated in Figure 9-18. When the MPC7410 needs to perform a bus access, it asserts $\overline{\text{BR}}$ to the arbiter (non-parked case). Xs in the figure mark the values of the signals that allow a qualified bus grant.

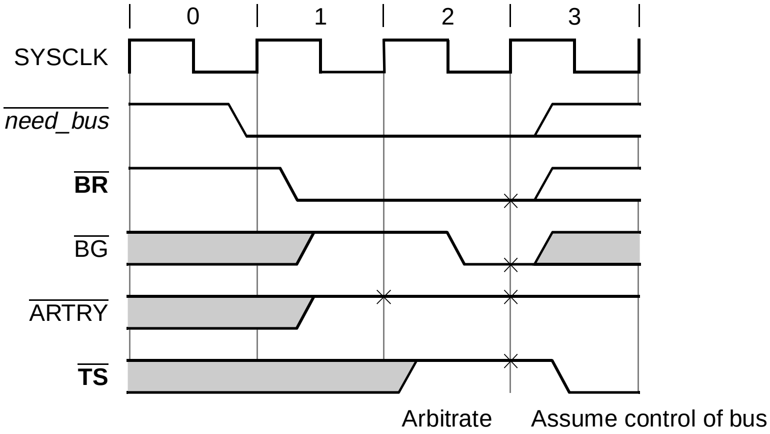


Figure 9-18. MPX Bus Address Bus Arbitration—Non-Parked Case

Arbiter designs must ensure that no more than one address bus master can be granted the bus at one time (that is, bus grants must be mutually exclusive). In single-master applications, $\overline{\text{BG}}$ can effectively be tied asserted, always granting the bus (called bus parking). However, as explained above, if $\overline{\text{AACK}}$ is used to delay an address tenure beyond the first cycle after $\overline{\text{TS}}$, the bus arbiter must negate $\overline{\text{BG}}$ in every cycle that the arbiter delays $\overline{\text{AACK}}$.

As is the case with 60x bus mode, the MPC7410 may assert $\overline{\text{BR}}$ but not use the bus after it receives a qualified bus grant, or it may negate (cancel) $\overline{\text{BR}}$ before accepting a qualified bus grant.

9.6.1.1.2 MPX Bus Mode Address Bus Parking

In the non-parked case, the processor must first assert $\overline{\text{BR}}$ to the arbiter to request the bus, and then wait to receive a bus grant from the arbiter on a later (or possibly the same) cycle.

Address parking becomes a very important feature of the MPX bus interface, because address bus tenures may be driven every other cycle by the same master. By speculatively parking the current master’s bus grant, the system gains back not only the arbitration latency, but also the dead cycle that exists between each tenure in 60x bus protocol.

As shown in Figure 9-19 bus parking for the MPX interface is the same as that for the 60x interface from the bus master’s perspective. If a master needs the bus and receives a bus grant and all qualifying conditions are met, the master may immediately assume control of the address bus. Note that the qualifications for the bus grant condition are different from that of the 60x interface, as described above.

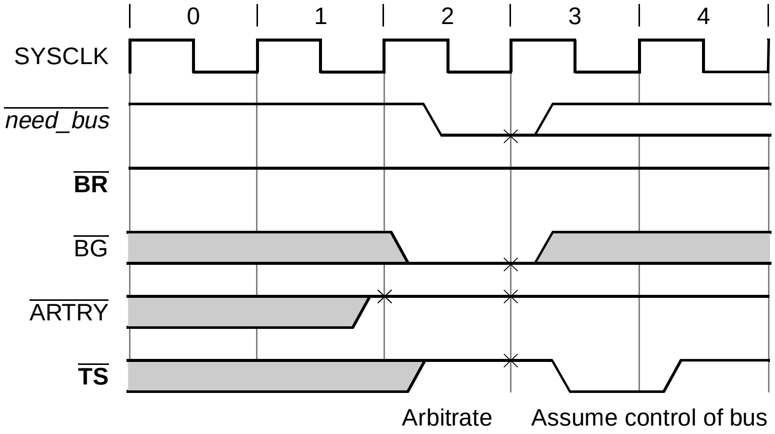
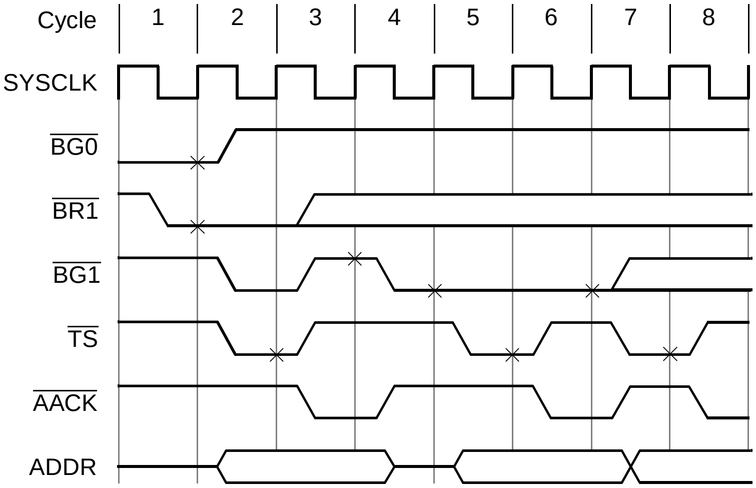


Figure 9-19. Bus Arbitration—Parked Case

From the system arbiter’s perspective, address bus parking must be implemented more carefully in MPX bus systems than in 60x bus systems because the qualified bus grant equation no longer includes a synthesized internal $\overline{abb}$.

As shown in Figure 9-20, optimal address parking can be implemented in a multimaster system because $\overline{\text{TS}}$ is still in the qualified bus grant equation for MPX bus masters.



Cycle 1: Master 0 has a parked address bus grant. Master 0 has an address tenure ready and a qualified bus grant, so it queues an address tenure for the next cycle. Also in cycle 1, the arbiter samples a bus request from master 1, so the arbiter queues a switch of the bus grants from master 0 to master 1. The arbiter can safely do this, because the qualified bus grant equation for MPX bus masters includes ¬$\overline{\text{TS}}$.

Cycle 2: Master 0 begins an address tenure, and master 1 does NOT get a qualified bus grant.

Cycle 3: The arbiter MUST negate the bus grant to master 1, because without $\overline{\text{ABB}}$ or $\overline{\text{AACK}}$ in the QBG equation, nothing would prevent master 1 from beginning an address tenure in cycle 4, colliding with the end of master 0's address tenure. Since it would introduce a difficult timing path to require the arbiter to sample $\overline{\text{TS}}$ in cycle 2 and negate $\overline{\text{BG1}}$ in cycle 3, it is suggested that arbiters always pulse $\overline{\text{BGx}}$ high a cycle after swapping $\overline{\text{BGx}}$ and $\overline{\text{BGy}}$, only reasserting $\overline{\text{BGx}}$ after $\overline{\text{AACK}}$ has been driven.

Cycle 4: The arbiter reasserts $\overline{\text{BG1}}$ because it asserted $\overline{\text{AACK}}$ in the previous cycle. (If the arbiter does not know in advance when $\overline{\text{AACK}}$ is to be asserted, this timing might be difficult, and the reassertion of bus grant may have to be delayed a cycle, but most systems should be able to do this.)

Cycle 5: Master 1 gets to start its address tenure. Note that this is the optimal timing for a new master to drive the address bus.

Cycles 5 and 6: The arbiter speculatively parks $\overline{\text{BG1}}$ enabling master 1 to begin another address tenure immediately in cycle 7.

Figure 9-20. Address Parking in MPX Bus Multiprocessor Systems

9.6.1.2 Address Transfer in MPX Bus Mode

During the address transfer phase in MPX bus mode, the physical address and transfer attributes are transferred from the master to the slave(s) similar to 60x bus mode. However, two differences in the address transfer phase in MPX bus mode are the addition of address bus driven mode and support for address bus streaming, described in the following sections.

9.6.1.2.1 Address Bus Driven Mode

In addition to selecting MPX bus mode at the negation of $\overline{\text{HRESET}}$, the $\overline{\text{EMODE}}$ signal is also used to select address bus driven mode after $\overline{\text{HRESET}}$ is negated. This mode provides for improved electrical characteristics on the address and attributes signals by reducing the time that these signals are not actively driven.

If $\overline{\text{EMODE}}$ is asserted after $\overline{\text{HRESET}}$ is negated, address bus driven mode is selected; if $\overline{\text{EMODE}}$ is negated after $\overline{\text{HRESET}}$ is negated, normal address bus driving mode (address bus not always driven) is selected. The read-only ABD bit in MSSCR0 indicates whether the MPC7410 is in address bus driven mode. Note that address bus driven mode is only available in MPX bus mode.

9.6.1.2.2 Address Bus Streaming

The 60x bus protocol forces a turn-around cycle on the bus between each address tenure implying that 60x address tenures last at least three bus clock cycles (because the system must provide $\overline{\text{AACK}}$ no earlier than the cycle following the assertions of $\overline{\text{TS}}$).

In MPX bus mode, the MPC7410 can drive consecutive address tenures without a dead cycle in between. So, two-cycle address tenures are possible if $\overline{\text{AACK}}$ is not delayed and the same master receives a qualified bus grant to drive another address tenure. This is referred to as address bus streaming.

9.6.1.2.3 Address Bus Parity

The MPC7410 address parity generation and reporting in MPX bus mode operates identically to that in 60x bus mode.

9.6.1.2.4 Address Pipelining

Address pipelining in the MPX bus mode uses split transactions in the same way as the 60x interface with the exception that data need not necessarily be returned in order. Generalized data tenure reordering, beyond that allowed by the $\overline{\text{DBWO}}$ function, is possible in the MPX interface. This generalized reordering protocol is described in Section 9.6.2.2.8, "Data Tenure Reordering in MPX Bus Only."

9.6.1.3 Transfer Attributes in MPX Bus Mode

The transfer attribute signals for the MPC7410 include the TT[0:4] and TSIZ[0:2] signals, as well as $\overline{\text{TBST}}$, $\overline{\text{CI}}$, $\overline{\text{WT}}$, and $\overline{\text{GBL}}$. There is no change to the implementation of the $\overline{\text{GBL}}$ signal in MPX bus mode. In MPX bus mode, the $\overline{\text{CI}}$ and $\overline{\text{WT}}$ signals function as inputs (as well as outputs). They are used as inputs in determining illegal conditions for data-only intervention where $\overline{\text{ARTRY}}$ must be asserted instead of $\overline{\text{HIT}}$ in response to a snooped transaction.

Attribute differences from the 60x bus are as follows:

- The definition of the TSIZ signals is expanded
- The read claim (RCLAIM) transfer type is added to the MPX bus mode for touch-for-store instructions.

9.6.1.3.1 Transfer Type 0–4 (TT[0:4]) in MPX Bus Mode

The TT*x* encodings for the MPX bus mode are the same as in Section 9.3.2.2.1, “Transfer Type (TT[0:4]) Signals in 60x Bus Mode,” with one new addition, RCLAIM. RCLAIM is used to identify touch-for-store instructions on the MPX bus. The effect of the RCLAIM transaction is to establish exclusive ownership of a cache line without marking that cache line as modified in the requesting processor’s cache.

Table 9-6. Transfer Type Encodings for MPX Bus Mode

Generated by MPC7410 as Bus Master		TT0	TT1	TT2	TT3	TT4	Command
Type	**Source**						
Burst	**dstst**, **dststt**, or **dcbtst**	0	1	1	1	1	Read claim (RCLAIM)

Note that in 60x bus mode, TT[0:4] = 0b01110 for reads caused by **dcbtst**, **dstst**, and **dststt**. See Section 9.3.2.2.1, “Transfer Type (TT[0:4]) Signals in 60x Bus Mode,” for more information.

9.6.1.3.2 Transfer Size

The transfer size (TSIZ[0:2]) signals indicate the size of the requested data transfer. The MPC7410 allows for two burst sizes in order to support both cache block transfers (32 bytes) and quad-word AltiVec loads and stores (16 bytes). Thus the definition of the TSIZ[0:2] bits when $\overline{\text{TBST}}$ is asserted is expanded from that in 60x bus mode. Table 9-7 defines the $\overline{\text{TBST}}$ and TSIZ[0:2] encodings used by the MPC7410 in MPX bus mode.

Table 9-7. $\overline{\text{TBST}}$ and TSIZ[0:2] Encodings in MPX Bus Mode

$\overline{\text{TBST}}$	TSIZ0	TSIZ1	TSIZ2	Transfer Size[1]
Asserted	0	0	0	reserved
Asserted	0	0	1	2 double-word burst
Asserted	0	1	0	4 double-word burst
Asserted	0	1	1	undefined
Asserted	1	0	0	reserved
Asserted	1	0	1	undefined
Asserted	1	1	0	undefined
Asserted	1	1	1	undefined
Negated	0	0	0	8 bytes
Negated	0	0	1	1 byte
Negated	0	1	0	2 bytes
Negated	0	1	1	3 bytes
Negated	1	0	0	4 bytes

Table 9-7. TBST and TSIZ[0:2] Encodings in MPX Bus Mode (continued)

$\overline{\text{TBST}}$	TSIZ0	TSIZ1	TSIZ2	Transfer Size[1]
Negated	1	0	1	5 bytes (N/A)
Negated	1	1	0	6 bytes (N/A)
Negated	1	1	1	7 bytes (N/A)

[1] 3-byte transfers may be requested by the MPC7410 starting at any byte address within the double word from byte address 0 to byte address 5.
4-byte transfers may be requested by MPC7410 starting at any byte address within the double word from byte address 0 to byte address 4.

9.6.1.3.3 Aligned and Misaligned Transfers

Performance on misaligned transfers may be substantially less than on aligned transfers, and it is recommended that software attempt to align code and data if possible. See Section 9.3.2.4, “Effect of Alignment in Data Transfers,” for a detailed description of alignment considerations for transactions in 60x and MPX bus modes.

9.6.1.4 Address Termination Phase in MPX Bus Mode

The address tenure is terminated by the assertion of the $\overline{\text{AACK}}$ input. In 60x bus mode, the 60x interface forces a turn-around cycle on the bus between each address tenure (because the system must provide $\overline{\text{AACK}}$ no earlier than the cycle following the assertions of $\overline{\text{TS}}$) implying that 60x address tenures last at least three bus clock cycles.

Because MPX bus mode supports address bus streaming, the MPC7410 can drive consecutive address tenures without a dead cycle in between. Using address bus streaming, two-cycle address tenures are possible if $\overline{\text{AACK}}$ is not delayed and the same master receives a qualified bus grant to drive another address tenure. Note that to accommodate address bus streaming, $\overline{\text{AACK}}$ must be asserted for only one bus clock cycle in MPX bus mode.

The address tenure can be terminated with the assertion of $\overline{\text{ARTRY}}$ anytime during the address tenure and through the cycle following $\overline{\text{AACK}}$ in MPX bus mode the same as in 60x bus mode. As a snooper (if data intervention is not enabled), the MPC7410 asserts $\overline{\text{ARTRY}}$ similarly (and for the same conditions) to the 60x bus mode. As in 60x bus mode, the MPC7410 is self-snooping and may assert $\overline{\text{ARTRY}}$ for its own transaction. See Section 3.9.3, “Snooping,” for more information. As a bus master, the MPC7410 also responds to the assertion of $\overline{\text{ARTRY}}$ in the same way as in 60x bus mode. See Section 3.4.3, “Coherency Protocols,” for more information.

Note that the snoop response in MPX bus mode is composed of the $\overline{\text{ARTRY}}$ signal, the shared ($\overline{\text{SHD}}$[0:1]) signals, and the $\overline{\text{HIT}}$ signal. These signals must be driven in the address retry window (as defined for 60x bus interface) and remain asserted until the end of the window. See Section 9.3.3.1, “Address Retry Window and Qualified ARTRY,” for more information.

9.6.1.4.1 Address Retry (ARTRY) in MPX Bus Mode

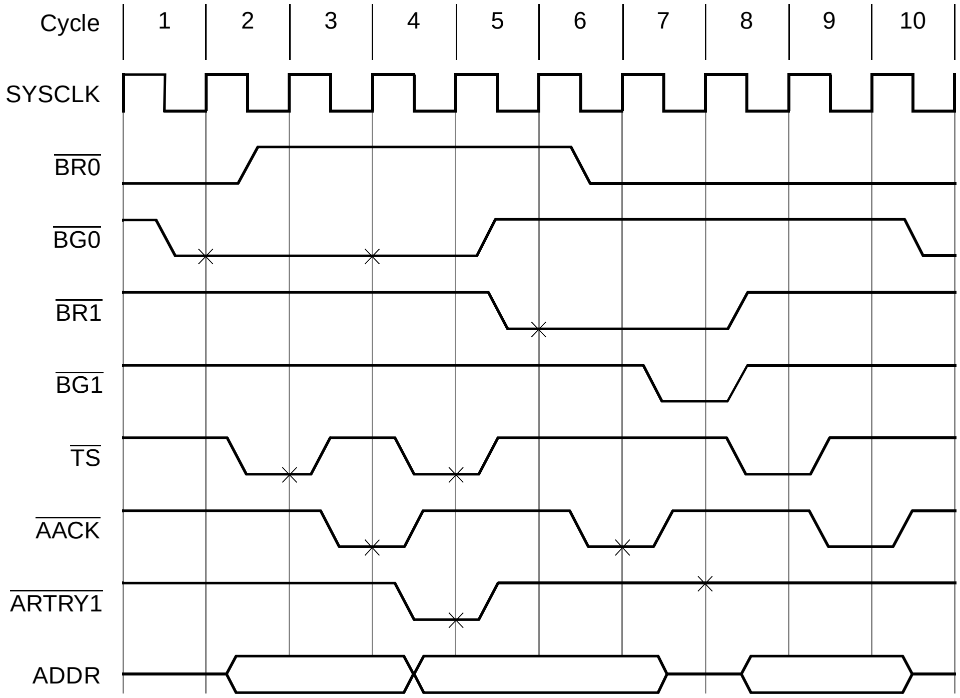
The following list identifies the cases in which ARTRY is asserted in the address retry window (when using MPX bus mode):

- No device on the bus has the required buffer space to handle the transaction.
- A device has a transient pipeline collision.
- A snooping device must push modified data to maintain coherency and data intervention is not enabled (L1_INTVEN and L2_INTVEN bits cleared in MSSCR0).
- A snooping device must push modified data to maintain coherency and data intervention is enabled but intervention is not possible for this transaction because the transaction is not for a full cache-line (detected by WT or CI asserted).

Note that in some instances, a single master may assert both ARTRY and HIT in response to a snooped transaction. In this case, the ARTRY has precedence and the HIT signal is ignored.

ARTRY assertions in MPX bus mode can have additional implications because the MPC7410 allows new address tenures to begin without a dead cycle in between. The MPC7410 allows a new address tenure from the same master to begin the cycle after AACK, which overlaps the address retry window of the previous address tenure. If this happens, the system and all bus devices must recognize that the second TS is implicitly retried as well. As with the 60x interface, however, ARTRY does not affect the termination of an address tenure—address tenures are only terminated by AACK. Therefore, even if an address tenure is to be retried by ARTRY for the previous address tenure, AACK is asserted to terminate the second address tenure (shown in Figure 9-21).

Note that in Figure 9-21, the AACK for the second address tenure is delayed by a clock cycle. This is to demonstrate how a system must handle a delayed AACK overlapping the window of opportunity after a retry. In this case the address bus grant for the snoop push requested in the window of opportunity must be delayed until at least the cycle after AACK. This may be required to avoid critical timing conflicts between ARTRY and AACK. Note that the grant could be delayed further to avoid any critical timing conflicts between AACK and the bus grant, if necessary. In any case, if the address tenure of the second transaction extends past the window of opportunity after the assertion of ARTRY, the arbiter must not rearbitrate and grant the address bus to any device that may have requested the bus after the window of opportunity but before the address tenure for the snoop push.



Cycle 1: The master has requested the bus and receives a qualified bus grant.

Cycle 2: The master begins the address tenure by driving $\overline{\text{TS}}$ and the address.

Cycle 3: The system responds with $\overline{\text{AACK}}$, ending the address tenure. The master receives another (parked) address bus grant.

Cycle 4: The master begins a new address tenure by driving $\overline{\text{TS}}$ and a new address. Some snooping device, however, asserts $\overline{\text{ARTRY}}$ for the first transaction. Bus grant remains parked to processor 0.

Cycle 5: The system delays $\overline{\text{AACK}}$ for the second transaction for some reason. $\overline{\text{BG0}}$ is negated to allow the retrying processor to request the bus. Processor 1 takes advantage of this window of opportunity and requests the bus to perform a push of the data that caused the retry.

Cycle 6: The system asserts $\overline{\text{AACK}}$ to terminate the second address tenure. Since the window of opportunity has passed, processor 0 requests the address bus again to retry its transaction. But the arbiter may NOT rearbitrate and grant the address bus to processor 0 before the push requested in the window of opportunity.

Cycle 7: Even though this cycle would be the address retry window for the second address tenure, no processor may assert $\overline{\text{ARTRY}}$, because that transaction was implicitly canceled by the $\overline{\text{ARTRY}}$. (If $\overline{\text{AACK}}$ had not been delayed, an assertion of $\overline{\text{ARTRY}}$ here could cause contention with the snooper that would be driving $\overline{\text{ARTRY}}$ negated from the address retry window of the first address tenure.) A bus grant is given to processor 1 to perform its push.

Cycle 8: Processor 1 begins its snoop push.

Cycle 9: The snoop push address tenure is acknowledged and terminated.

Cycle 10: The arbiter now grants processor 0 the address bus to retry its transaction.

Figure 9-21. Overlapped $\overline{\text{ARTRY}}$ and $\overline{\text{TS}}$ (with a Delayed $\overline{\text{AACK}}$) in MPX Bus Mode

9.6.1.4.2 Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$) Signals for MPX Bus Mode

In 60x bus mode the shared response ($\overline{\text{SHD}}$) is multiplexed with the $\overline{\text{SHD0}}$ output signal as described in Section 8.4.5.3, "MPX Bus Shared ($\overline{\text{SHD0}}$, $\overline{\text{SHD1}}$) Signals." The shared response for the MPC7410 in MPX bus mode is divided into two separate output signals, $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$. The MPX bus mode requires two signals because address tenures may be produced every other cycle in this mode. Because the shared response must be driven negated between assertions, and since multiple devices may drive a shared response in a given address retry window, it is necessary to release the shared signal to high-impedance after asserting it, drive it negated, and then release it to high-impedance again. Timing requirements make this very difficult for a single signal that may need to be asserted on the second cycle after a previous assertion.

If the MPC7410 needs to assert a shared snoop response and $\overline{\text{SHD0}}$ was not asserted in any of the three cycles prior to the address retry window for the current transaction, then it asserts $\overline{\text{SHD0}}$ during the address retry window. If $\overline{\text{SHD0}}$ was asserted in the three cycles before the shared response needs to be asserted, then the MPC7410 asserts $\overline{\text{SHD1}}$ instead. A master observing the snoop response must consider the shared response asserted if either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ is asserted.

The timing for the release to high-impedance, negating, and re-release to high-impedance, of $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ may vary. To ensure compatibility with the standard 60x interface in which $\overline{\text{SHDx}}$ might need to be asserted up to every three bus cycles, the MPC7410 implements the 60x-style timing for both $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$; that is $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ have the same timing as $\overline{\text{ARTRY}}$, in which the signal is released to high-impedance for a fraction of a cycle, then negated for up to an entire cycle (crossing a bus cycle boundary) before being released to high-impedance again. Note that future implementations with the MPX protocol may define this timing differently. The MPC7410 does not assert either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ as outputs to be asserted more often than every fourth bus clock cycle. Additionally, it does not allow either $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ as inputs to be asserted more often than every fourth bus clock cycle. See Figure 9-22.

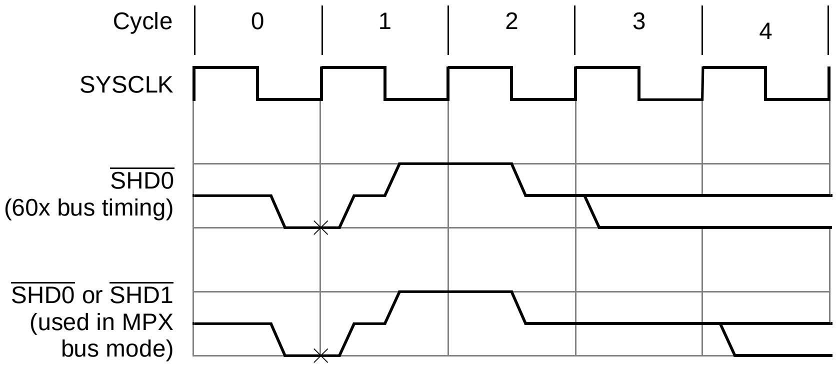


Figure 9-22. $\overline{\text{SHD0}}$ and $\overline{\text{SHD1}}$ Negation Timing

To ease timing requirements, devices detecting $\overline{\text{SHD0}}$ or $\overline{\text{SHD1}}$ should not sample these signals beyond the second cycle after $\overline{\text{AACK}}$, since the signals can either be released to high-impedance or fractionally precharged in those cycles.

9.6.1.4.3 Hit ($\overline{\text{HIT}}$) Signal and Data Intervention

The $\overline{\text{HIT}}$ signal in MPX bus mode was added to support direct cache-to-cache transfers (also called data intervention) and local bus slaves. A device that asserts $\overline{\text{HIT}}$ in the address retry window indicates to the system that it owns the data for the requested load, and that it will supply the data by performing a data-only transaction. Note that the $\overline{\text{HIT}}$ signal is a point-to-point signal between the MPC7410 and the central arbiter/memory controller. The master that requested the load does not see the $\overline{\text{HIT}}$ signal and does not know that the data is coming from a master or local bus slave rather than memory.

When data intervention is enabled with the L1_INTVEN or L2_INTVEN bits set in MSSCR0, the MPC7410 attempts to intervene to service a load when it detects a snoop hit for data it owns exclusively (modified or not) or data that is in the recent (R) state seeSection 3.4.3, “Coherency Protocols.” Unless $\overline{\text{ARTRY}}$ is asserted simultaneously by another bus device, the MPC7410 requests a data-only transaction by using the $\overline{\text{DRDY}}$ protocol described in Section 9.6.2.2.2, “Data Intervention—MPX Bus Mode.”

When the MPC7410 performs data intervention, it always provides 32-bytes of intervention data in critical double-word first ordering, regardless of the TSIZ[0:2] encoding or the state of $\overline{\text{TBST}}$ for the snooped transaction. Thus, if a system implementation provides global snoop traffic to the MPC7410 that could result in data intervention from the MPC7410 and does not consist of 32-byte transfers, the system must either sink the entire 32-bytes of intervention data from the MPC7410, or else disable all types data intervention in the MPC7410 with the L1_INTVEN and L2_INTVEN bits of MSSCR0.

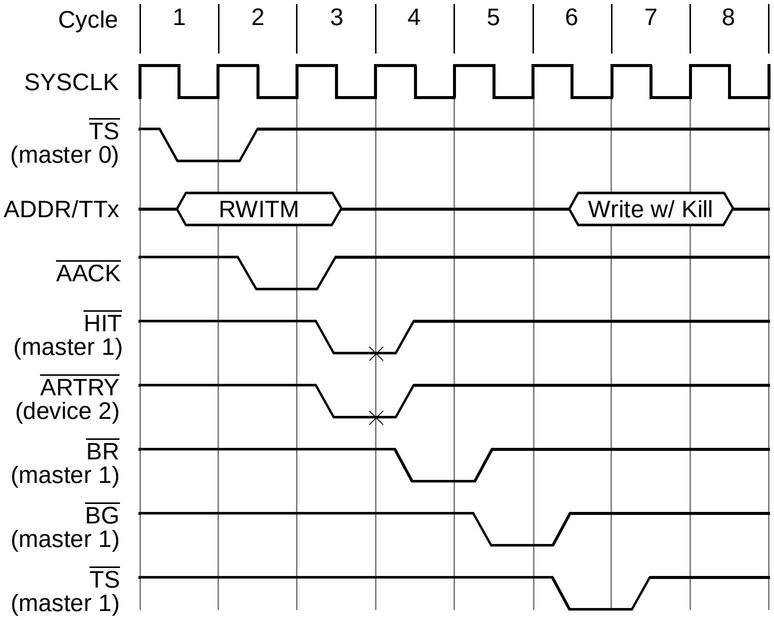
If a snoop is sampled by the MPC7410 with either the $\overline{\text{WT}}$ or $\overline{\text{CI}}$ signal asserted, the MPC7410 does not assert $\overline{\text{HIT}}$ in response. If for any reason, this access with $\overline{\text{WT}}$ or $\overline{\text{CI}}$ asserted generates a snoop hit on a cache line in the modified state, the MPC7410 performs a window of opportunity-style write-with-kill push operation. This occurs if one of the following programming errors exists:

- A global W = 1 (write-through) page was aliased with a W = 0 (write-back) page, or
- A global I = 1 (caching-inhibited) page was aliased with a I = 0 (caching-allowed) page.

Neither of these types of aliasing are supported by the MPC7410.

Because the coherency protocol insures that only one device has the ability to supply intervention data, an $\overline{\text{ARTRY}}$ asserted by a second device when one master is asserting $\overline{\text{HIT}}$ can only occur when that second device cannot handle the snooping of the address due to buffer space limitations or pipeline collisions. In this case, the device asserting $\overline{\text{ARTRY}}$ is not trying to intervene or perform a snoop push. Therefore, if the MPC7410 asserts $\overline{\text{HIT}}$ because it detected a snoop hit for modified data and $\overline{\text{ARTRY}}$ is asserted simultaneously by another device, the MPC7410 takes advantage of the window of opportunity to perform a push operation. Note that if the $\overline{\text{HIT}}$ occurred in order to supply intervention data that was shared or exclusive unmodified, the MPC7410 does not perform a push after another device asserts $\overline{\text{ARTRY}}$ because the data is still clean and valid in both the cache and main memory.

Figure 9-23 shows an example timing diagram of $\overline{\text{HIT}}$ and $\overline{\text{ARTRY}}$ being asserted together.


Cycle
1
2
3
4
5
6
7
8
SYSCLK
$\overline{\text{TS}}$ (master 0)
ADDR/TTx
RWITM
Write w/ Kill
$\overline{\text{AACK}}$
$\overline{\text{HIT}}$ (master 1)
$\overline{\text{ARTRY}}$ (device 2)
$\overline{\text{BR}}$ (master 1)
$\overline{\text{BG}}$ (master 1)
$\overline{\text{TS}}$ (master 1)

Cycle 3: Master 1 asserts $\overline{\text{HIT}}$, but device 2 retries the transaction.

Cycle 4: Master 1 asserts $\overline{\text{BR}}$ to take advantage of the window of opportunity to flush the data, just as if it had asserted $\overline{\text{ARTRY}}$ itself instead of $\overline{\text{HIT}}$.

Cycles 6 and 7: Master 1 gains control of the address bus and begins a Write w/ Kill transaction to flush the data.

Figure 9-23. $\overline{\text{HIT}}$ and $\overline{\text{ARTRY}}$ Asserted Together

9.6.1.4.4 $\overline{\text{HIT}}$ Signal Timing and Data Snarfing

The MPC7410 distinguishes between shared and exclusive intervention in the system and the intervention of modified data with the timing of $\overline{\text{HIT}}$ negation. For shared or exclusive intervention, the MPC7410 holds the $\overline{\text{HIT}}$ signal asserted for a second cycle after the address retry window indicating to the system that the data being supplied through intervention does not need to be forwarded to memory (snarfed) because the data is not modified.

Snarfing is necessary when a processor forwards modified data through intervention because the processor performing the read operation will mark the data exclusive even though the data has been modified. Snarfing is not necessary for RWITM transactions serviced via intervention because the data will be marked modified in the requesting processor’s cache.

9.6.2 Data Tenure in MPX Bus Mode

The MPX bus mode implements several new features to improve both bandwidth and utilization of the data bus compared with the 60x bus mode. These include:

- Simplification of signals and modes to optimize the most timing-critical logic paths.

- A new data-only transaction type to support cache-to-cache data transfers (data intervention) and data transfers from local bus slaves
- Support for generalized data tenure reordering

The MPX bus mode supports a streaming mode of data transfer. This, in general, allows burst data tenures from a single source to be driven back-to-back without a dead cycle in between.

9.6.2.1 Data Bus Arbitration Phase in MPX Bus Mode

Data bus arbitration in MPX bus mode uses some of the same data arbitration signal group ($\overline{\text{DBG}}$) as 60x bus mode except that DTI[0:2] are used instead of $\overline{\text{DBWO}}$ for data tenure reordering.

9.6.2.1.1 Qualified Data Bus Grant in MPX Bus Mode

The use of data streaming requires significant changes to the qualification of $\overline{\text{DBG}}$ by a master in MPX bus mode. When the MPC7410 is operating in MPX bus mode, a qualified data bus grant occurs when the following conditions are satisfied:

- $\overline{\text{DBG}}$ is asserted
- $\overline{\text{ARTRY}}$ is not asserted in the address retry window for the address phase of this transaction
- The processor is ready to begin a data transaction, which means the following:.
  - — The processor is not already using the data bus, or
  - — The processor has a burst access in progress, the processor has already received the next-to-last $\overline{\text{TA}}$ for the current burst, and the source of the next access is the same as the source for the current access.

These conditions mean that the system arbiter must never assert $\overline{\text{DBG}}$ to a processor when the data bus is busy with a transaction for another processor. The system arbiter must synthesize its own data bus busy state. However, note that if a transaction for device A is currently receiving its last $\overline{\text{TA}}$ and the same device is ready to source data to processor B, $\overline{\text{DBG}}$ to processor B may be asserted, and the data source can continue to stream another data tenure.

The negation of $\overline{\text{ARTRY}}$ for that address tenure indicates that the address tenure associated with the data tenure about to be granted was not retried by a snooping device. Because of address pipelining, an assertion of $\overline{\text{ARTRY}}$ may not be for the data tenure about to be granted; therefore, it may not affect data bus grant qualification.

One bus clock cycle after accepting a qualified data bus grant, the MPC7410 begins driving or sampling the data bus and sampling the transfer acknowledge signals.

9.6.2.1.2 Data Streaming Constraints for Data Bus Arbitration in MPX Bus Mode

Data streaming from one data tenure to the next in MPX mode is supported only for data tenures in which the source of the data is the same. A dead cycle must be placed in between two adjacent data tenures when the data is driven by two different devices. For example, if the first data transfer is a processor read from memory and the second data transfer is a processor write to memory, data streaming is not supported; a minimum of one dead cycle must be inserted between the final $\overline{\text{TA}}$ of the read and the first $\overline{\text{TA}}$ of the write.

In this example, the earliest cycle that the processor accepts $\overline{\text{DBG}}$ as a qualified $\overline{\text{DBG}}$ is the cycle after the final $\overline{\text{TA}}$ of the read.

In addition, data streaming from one data transfer to a second is only allowed if the first transfer is a multiple-beat transfer (requiring at least two $\overline{\text{TA}}$s). If the first data transfer is a single-beat transfer (only one $\overline{\text{TA}}$), then the earliest that the processor will accept a $\overline{\text{DBG}}$ as a qualified $\overline{\text{DBG}}$ is the cycle after the single $\overline{\text{TA}}$ assertion.

Note that with these two constraints, it is possible to stream from a multiple-beat transaction into a single-beat transaction. While this may not affect overall system performance, it may simplify the design of the data bus arbiter and memory controller.

9.6.2.2 Data Bus Transfers

The MPC7410 in MPX bus mode expects data transactions to be carried out in 64-bit beats just like in the 60x bus interface. Four-beat burst transfers are used by the MPC7410 to transfer cache lines into or out of its internal cache. Two-beat burst transfers are used for caching-inhibited or write-through 128-bit AltiVec loads or stores. Non-burst transfers are performed for non-AltiVec caching-inhibited or write-through operations. The MPC7410 does not directly support dynamic interfacing to subsystems with less than a 64-bit wide data path, and it does not support a static 32-bit data bus mode. (See Section 9.6.1.3, “Transfer Attributes in MPX Bus Mode”).

Note that the operation of the data and data parity signals is the same as that in the 60x bus mode (see Section 9.4.2, “Data Transfer Signals and Protocol,”) except as noted in the following subsections.

9.6.2.2.1 Earliest Transfer of Data

In 60x bus mode, data must never be transferred before the first cycle of the address retry window (that is, valid data must never precede a possible $\overline{\text{ARTRY}}$ for that transaction). This same restriction applies for MPX bus mode.

An additional requirement in systems that support data intervention or local bus slaves that use the $\overline{\text{HIT}}$/$\overline{\text{DRDY}}$ protocol is that data must only be driven or sampled after the completion of the address retry window, so that the $\overline{\text{HIT}}$ signal can be sampled.

9.6.2.2.2 Data Intervention—MPX Bus Mode

If the MPC7410 performs a read or RWITM of data that exists modified in another processor’s cache, the 60x bus requires that the transaction be retried and data pushed to memory before the transaction is begun a second time. A more efficient approach, used by the MPC7410 in MPX bus mode, is to allow the data to be forwarded directly to the requesting master from the processor that has it cached. This is called data intervention. The MPC7410 performs this function through the $\overline{\text{HIT}}$/$\overline{\text{DRDY}}$ protocol and data-only transactions. The MPC7410 also supports intervention for data that is not modified. Exclusive data can also be forwarded through intervention, and even shared data can be forwarded by using the MERSI coherency protocol described in Section 3.4.3. Each of these intervention types is selected by the setting of the L1_INTVEN and L2_INTVEN bits in MSSCR0 in MPX bus mode.

Note that data intervention is only allowed for full cache-line transfers. A snooping MPC7410 does not assert $\overline{\text{HIT}}$ if it detects $\overline{\text{WT}}$ or $\overline{\text{CI}}$ asserted because data intervention is not supported for caching-inhibited or write-through accesses.

An important implication of data intervention is that data must always be pushed with the critical data first, and the full double-word address must always be placed on the address bus. Otherwise, data could be received in the wrong order by the requesting master.

Intervention allows the latency for data that exists in another processor's cache to be reduced from over 20 bus cycles to as low as 5 or 6 cycles, as shown in Figure 9-24. Figure 9-24 shows transfer involving different DTI indices occurring in 6 cycles. Note that this could be reduced in some systems to 5 cycles if the DTI index provided to both processors is the same for this transfer and the DTI is able to be presented in clock cycle 3.

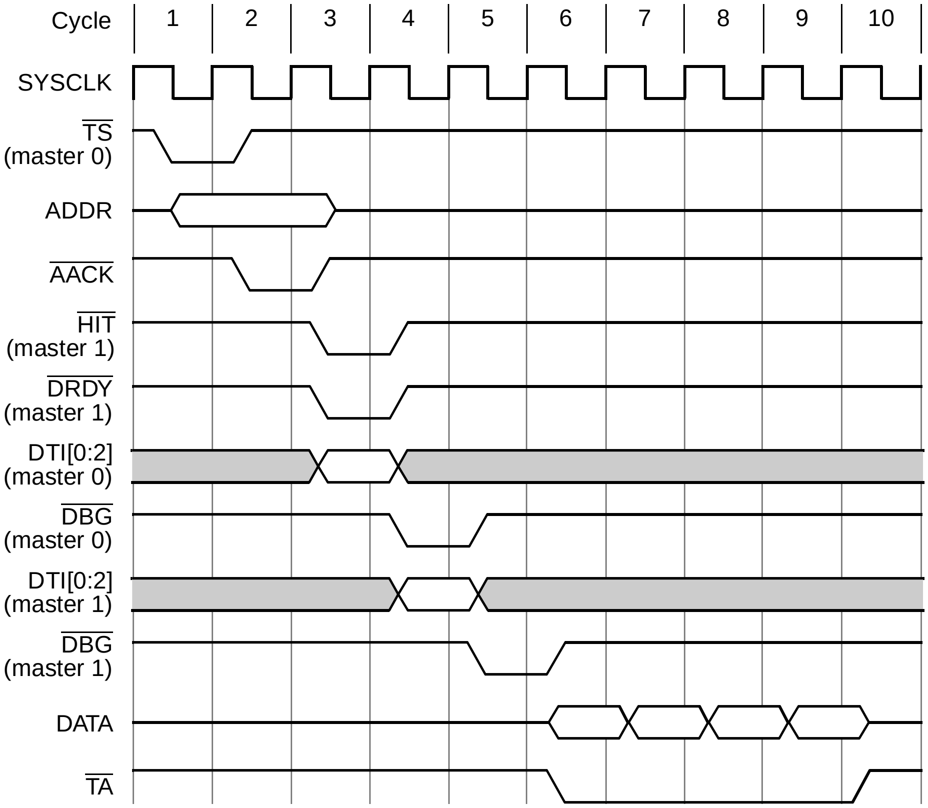


Figure 9-24. Data Intervention for Read (Atomic) and RWITM (Atomic) Using the Data-Only Transfer Protocol

9.6.2.2.3 Data-Only Transaction Protocol

A data-only transaction in MPX bus mode is differentiated from a typical address and data transaction in the 60x protocol in that the data bus is explicitly requested by the snooper or slave involved in the transaction, and no new address is specified by the snooper or slave. Data-only transactions are responses by snoopers or slaves to a transaction already initiated by a master with an address tenure, so the address is already known to the system.

Data-only transactions can be issued in response to address-only transactions (for example, Flush or Clean), or address and data transactions (for example, read or RWITM) to which the MPC7410 asserted the $\overline{\text{HIT}}$ response in the address retry window. The MPC7410 can use data-only transactions to supply intervention data in response to a read or RWITM, or to push data in response to an address-only Flush or Clean.

Figure 9-25 shows an example of a data-only transaction for a Flush.

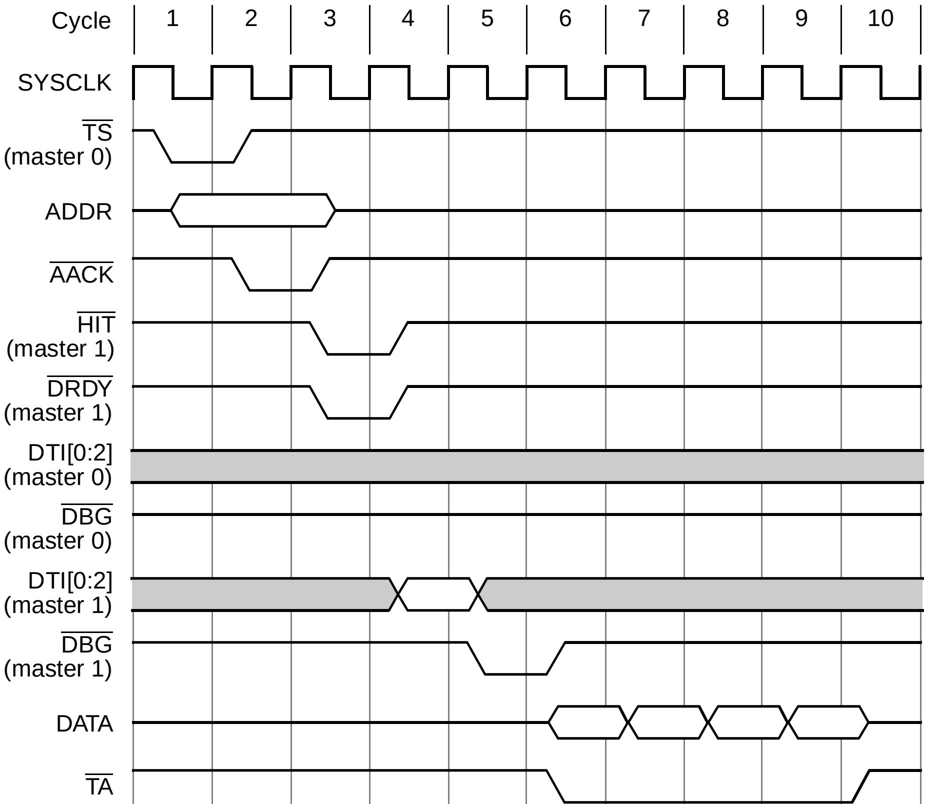

Cycle
1
2
3
4
5
6
7
8
9
10
SYSCLK
$\overline{\text{TS}}$ (master 0)
ADDR
$\overline{\text{AACK}}$
$\overline{\text{HIT}}$ (master 1)
$\overline{\text{DRDY}}$ (master 1)
DTI[0:2] (master 0)
$\overline{\text{DBG}}$ (master 0)
DTI[0:2] (master 1)
$\overline{\text{DBG}}$ (master 1)
DATA
$\overline{\text{TA}}$

Figure 9-25. Data-Only Transaction for a Flush Operation

As with the 60x bus, address and data transactions do not need to request explicitly the data bus, since the request is implicit in an address tenure for an address and data transaction. However, processors performing data-only transactions request the data bus by asserting $\overline{\text{DRDY}}$ when they have the data available. The $\overline{\text{DRDY}}$ causes the arbiter to assert $\overline{\text{DBG}}$ to all devices participating in the transaction. Whichever device in the system is responsible for asserting $\overline{\text{TA}}$ then does so to complete the transaction.

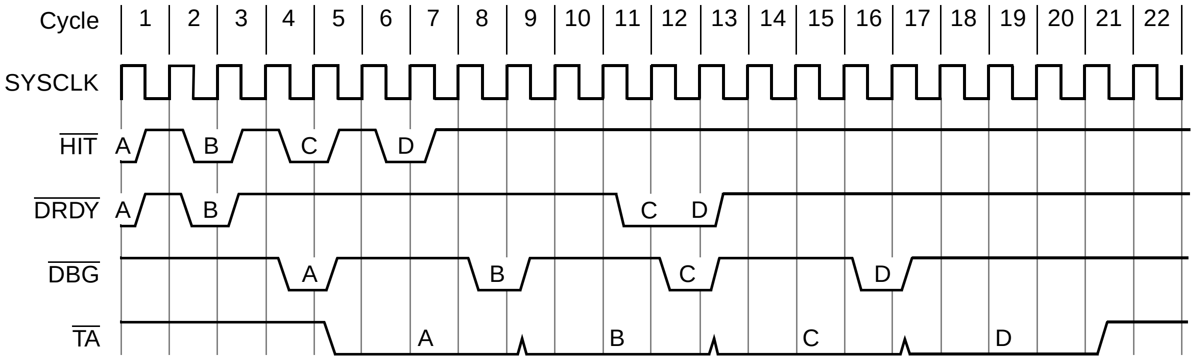
9.6.2.2.4 $\overline{\text{DRDY}}$ Timing (Data-Only Transactions)

$\overline{\text{DRDY}}$ may be asserted simultaneously with $\overline{\text{HIT}}$, but it must be asserted no sooner than two cycles before the device is ready to supply data because as shown above, $\overline{\text{TA}}$ may be driven two cycles after $\overline{\text{DRDY}}$. $\overline{\text{DRDY}}$ is a pulsed signal and must be negated on the cycle after it is asserted unless pipelining of data-only intervention requests would naturally require consecutive assertions of $\overline{\text{DRDY}}$. Because $\overline{\text{DRDY}}$ is a pulsed signal, it is not held when $\overline{\text{AACK}}$ is delayed (unlike $\overline{\text{HIT}}$).

9.6.2.2.5 Pipelining of Data-Only Transactions

The MPC7410 may pipeline multiple data-only transactions. That is, it may assert HIT for additional transactions, even if it has not been able to complete data-only transactions for prior snoop hits. It may do this until its internal buffers fill up. Similarly, a device may assert DRDY for additional data-only transactions before previous data-only transactions have completed their data tenures. There is no restriction on the maximum number of outstanding DRDY assertions. The MPC7410 can assert DRDY for the same number of times corresponding to the number of outstanding HIT assertions. The MPC7410 supports six outstanding transactions if no DBGs have been issued for any of the queued data transactions.

Because DRDY is a pulsed signal, if DRDY is held low for multiple cycles, the system interprets this as multiple assertions of DRDY. If a device asserts DRDY when the system is not expecting a DRDY (no pending data-only transaction has been indicated), the system ignores the DRDY signal. An important example of this would be the cycle after an ARTRY. See Section 9.6.2.2.6, "Retrying Data-Only Transactions." An example of pipelined data-only transactions is shown in Figure 9-26. (Note that Figure 9-26 assumes that data is all driven from the same source, so that data streaming is possible.)



Cycles 1, 3, 5, and 7: The device asserts HIT for transactions A, B, C, and D.

Cycles 1 and 3: The device asserts DRDY for the first A and B respectively.

Cycle 5: DBG for transaction A issued. DRDY for transaction C is delayed for some reason.

Cycle 6: Data for transaction A is driven. DRDY for transaction C still delayed.

Cycle 7: Transaction D can be driven even though no DRDY for transaction C and DBG for transactions B and C have been received.

Cycle 9: DBG for transaction B issued. DRDY for transaction C still delayed.

Cycle 10: Data for transaction B is driven. DRDY for transaction C still delayed.

Cycle 12: DRDY for transaction C is driven.

Cycle 13: DRDY for transaction D is pipelined. DBG for transaction C issued.

Cycle 13: Data for transaction C is driven

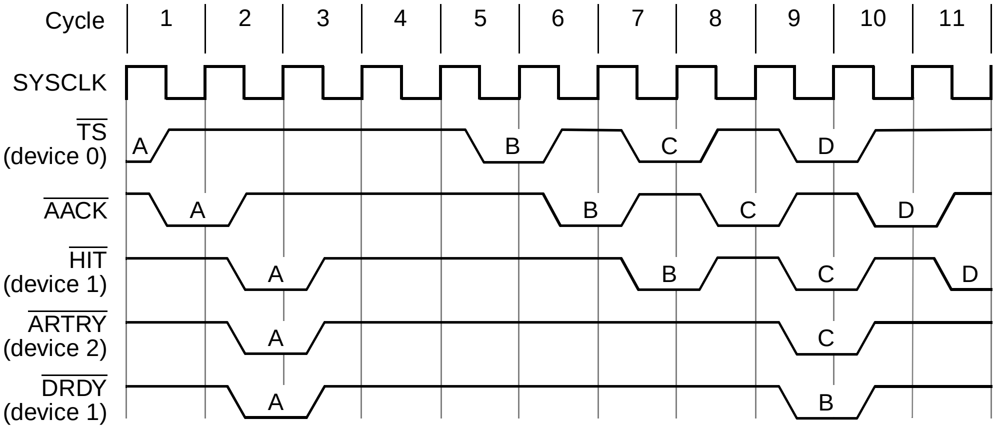
Figure 9-26. Pipelined Data-Only Transactions

9.6.2.2.6 Retrying Data-Only Transactions

As described in Section 9.6.1.4.4, "HIT Signal Timing and Data Snarfing," it is possible for the MPC7410 to signal a data-only transaction with the HIT signal while another device asserts ARTRY. In this case the data-only transaction must be considered retried, and the system does not expect a corresponding DRDY. If DRDY was asserted simultaneously with HIT, it is ignored. In general, if DRDY is asserted by the MPC7410 when the system is not expecting a DRDY, it will be ignored. So if the MPC7410 asserts DRDY after the transaction has been retried, the system should ignore it. However, the MPC7410 does not assert DRDY after the transaction has been retried (or any other spurious DRDY) if the system is expecting a DRDY for another transaction from that device. That is, there is no ambiguity between a spurious DRDY from a retried transaction and a valid DRDY for a later transaction.

Note that if data-only tenures are being pipelined, and the DRDY for a previous HIT is asserted at the same time as a new HIT and ARTRY, the DRDY must not be ignored.

Examples of retrying data-only transactions, including pipelined HITs and DRDYs are shown below in Figure 9-27.



Cycle 3: Transaction A receives an ARTRY so the HIT and DRDY signals are ignored by the system.

Cycle 6: Transaction B starts. This is the earliest possible TS after an ARTRY.

Cycle 8: Transaction B receives a HIT response. DRDY is delayed.

Cycle 10: Transaction C receives a HIT and an ARTRY. The system understands that the DRDY is for transaction B and considers it valid.

Cycle 12: HIT for transaction D is asserted. The bus master must not assert DRDY for transaction C at this point since the system would interpret it as the DRDY for transaction D.

Figure 9-27. Retry Examples of Data-Only Transactions

9.6.2.2.7 Ordering of Data-Only Transactions

All data-only transactions for a given MPC7410 processor must be handled in order. This does not mean that other data tenures must be handled in order with respect to data-only transactions, or that data-only transactions from different devices must maintain order relative to one another. See Section 9.6.2.2.8, “Data Tenure Reordering in MPX Bus Only.” However, if the MPC7410 has asserted HIT for more than one transaction, the corresponding data-only transactions must be serviced in the same order, since there is no defined way for the arbiter to distinguish between them except to expect them in order.

9.6.2.2.8 Data Tenure Reordering in MPX Bus Only

The MPC7410 allows data tenures to be executed out of order with respect to their corresponding address tenures in MPX bus mode. The system must supply an index with each data bus grant to indicate which of the master’s outstanding data tenures is being serviced. The data transfer index inputs, DTI[0:2], provide this function.

The DTI[0:2] signals act as a pointer into the queue of outstanding transactions within the MPC7410. The MPC7410 supports up to six outstanding transactions. DTI[0:2] for a given data bus tenure is driven by the system on the cycle prior to the associated $\overline{\text{DBG}}$. The MPC7410 continually samples DTI[0:2] and qualifies the pointer on the subsequent cycle if the data bus grant is qualified.

A DTI value of 0b000 indicates that the data tenure for the oldest remaining transaction is to be serviced; a value of 0b001 indicates that the second oldest remaining transaction is to be serviced, etc. A value of 0b101 selects the sixth and oldest MPC7410 transaction. The system tracks the status of the MPC7410 queues by monitoring the $\overline{\text{TS}}$ and $\overline{\text{DBG}}$. The MPC7410 adds a new transaction to the tail of its queue with each assertion of $\overline{\text{TS}}$ for an address and data transaction and each assertion of $\overline{\text{HIT}}$ for data-only transactions. It removes an entry if the transaction is retried with $\overline{\text{ARTRY}}$, or when it receives a qualified $\overline{\text{DBG}}$. When a transaction is removed from the queue, all transactions newer than the transaction removed shift forward.

The system must not provide an illegal DTI value the cycle before any qualified data bus grant when the processor has one or more valid data transactions queued. An illegal DTI value is defined as one of the following:

- A DTI value greater than 0b101
- A DTI value that does not have any valid corresponding data transaction unless the DTI value is 0b000. For instance, if only two data transactions are queued, a DTI value of 0b010 or greater is illegal.

The processor’s behavior is boundedly undefined if an illegal DTI value is detected at this time.

9.6.2.3 Data Termination Phase in MPX Bus Mode

Three signals are used to terminate the individual data beats of the data tenure and the data tenure itself—$\overline{\text{TA}}$, $\overline{\text{TEA}}$ and $\overline{\text{ARTRY}}$. In MPX bus mode, they function as in the 60x bus mode.

9.7 Interrupt, Checkstop, and Reset Signal Interactions

This section describes external interrupts, checkstop operations, and hard and soft reset inputs. See Chapter 4, “Exceptions,” and Chapter 8, “Signal Descriptions,” for more information on the exceptions caused by these signals and for signal descriptions, respectively.

9.7.1 External Interrupts

The external interrupt input signals ($\overline{\text{INT}}$, $\overline{\text{SMI}}$ and $\overline{\text{MCP}}$) of the MPC7410 force the processor to take the external interrupt vector or the system management interrupt vector if the MSR[EE] is set, or the machine check interrupt if the MSR[ME] and the HID0[EMCP] bits are set.

9.7.2 Checkstops

The MPC7410 has two checkstop input signals—$\overline{\text{CKSTP\_IN}}$ (nonmaskable) and $\overline{\text{MCP}}$ (enabled when MSR[ME] is set, and HID0[EMCP] is set), and a checkstop output ($\overline{\text{CKSTP\_OUT}}$) signal. If $\overline{\text{CKSTP\_IN}}$ or $\overline{\text{MCP}}$ is asserted, the MPC7410 halts operations by gating off all internal clocks. The MPC7410 asserts $\overline{\text{CKSTP\_OUT}}$ if $\overline{\text{CKSTP\_IN}}$ is asserted.

If $\overline{\text{CKSTP\_OUT}}$ is asserted by the MPC7410, it has entered the checkstop state, and processing has halted internally. The $\overline{\text{CKSTP\_OUT}}$ signal can be asserted for various reasons including receiving a $\overline{\text{TEA}}$ signal and detection of external parity errors. For more information about checkstop state, see Section 4.6.2.2, “Checkstop State (MSR[ME] = 0).”

All non-test output signals except $\overline{\text{CKSTP\_OUT}}$, $\overline{\text{L2CE}}$, $\overline{\text{L2WE}}$, and L2CLK_OUTn are disabled during a checkstop. The $\overline{\text{L2CE}}$, $\overline{\text{L2WE}}$, and L2CLK_OUTn signals are driven to maintain the SRAM state.

9.7.3 Reset Inputs

The MPC7410 has two reset inputs, described as follows:

- $\overline{\text{HRESET}}$ (hard reset)—The $\overline{\text{HRESET}}$ signal is used for power-on reset sequences, or for situations in which the MPC7410 must go through the entire cold start sequence of internal hardware initializations.
- $\overline{\text{SRESET}}$ (soft reset)—The soft reset input provides warm reset capability. This input can be used to avoid forcing the MPC7410 to complete the cold start sequence.

When $\overline{\text{HRESET}}$ is negated or $\overline{\text{SRESET}}$ is recognized, the processor attempts to fetch code from the system reset exception vector. The vector is located at offset 0x00100 from the exception prefix (all zeros or ones, depending on the setting of the exception prefix bit in the machine state register (MSR[IP])). The MSR[IP] bit is set when $\overline{\text{HRESET}}$ negates.

9.8 Processor State Signal Interactions

This section describes the MPC7410's support for power management and atomic update and memory access through the use of the **lwarx**/**stwcx.** opcode pair.

9.8.1 System Quiesce Control Signals

The system quiesce control signals ($\overline{\text{QREQ}}$ and $\overline{\text{QACK}}$) allow the processor to enter the nap or sleep low-power states and bring bus activity to a quiescent state in an orderly fashion.

Prior to entering the nap or sleep-power state, the MPC7410 asserts the $\overline{\text{QREQ}}$ signal. This signal allows the system to terminate or pause any bus activities that are normally snooped. When the system is ready to enter the system quiesce state, it asserts the $\overline{\text{QACK}}$ signal. At this time the MPC7410 may enter the nap or sleep power-saving state. When the MPC7410 is in the quiescent state, it stops snooping bus activity.

While the MPC7410 is in one of these power-saving states, the system power controller can enable snooping by the MPC7410 by negating the $\overline{\text{QACK}}$ signal for at least eight bus clock cycles, after which the MPC7410 is capable of snooping bus transactions. The reassertion of $\overline{\text{QACK}}$ following the snoop transactions causes the MPC7410 to reenter the nap power state. See Chapter 10, "Power Management," for more information on the power-saving modes of the MPC7410.

Once the MPC7410 has made a request to enter the nap power-saving state, the $\overline{\text{QREQ}}$ signal may be negated on any clock cycle to service an internal interrupt (such as a decrementer or time base exception). The $\overline{\text{TS}}$ for the exception vector fetch can occur as early as the clock cycle that $\overline{\text{QREQ}}$ is negated.

9.8.2 Support for the lwarx/stwcx. Instruction Pair

The Load Word and Reserve Indexed (**lwarx**) and the Store Word Conditional Indexed (**stwcx.**) instructions provide a means for atomic memory updating. Memory can be updated atomically by setting a reservation on the load and checking that the reservation is still valid before the store is performed. In the MPC7410, the reservations are made on behalf of aligned, 32-byte sections of the memory address space.

The reservation ($\overline{\text{RSRV}}$) output signal is driven synchronously with the bus clock and reflects the status of the reservation coherency bit in the reservation address register; see Chapter 3, "L1 and L2 Cache Operation," for more information. For information about $\overline{\text{RSRV}}$ signal timing, see Section 8.5.4.1, "Reservation (RSRV)—Output."

9.9 IEEE Std. 1149.1a-1993 Compliant Interface

The MPC7410 boundary-scan interface is a fully-compliant implementation of the IEEE Std. 1149.1a-1993 standard. This section briefly describes the MPC7410's IEEE Std. 1149.1a-1993 (JTAG) interface. See Section 8.5.6, "IEEE Std. 1149.1a-1993 (JTAG) Interface Description," for more information on the function of the JTAG signals.

9.9.1 JTAG/COP Interface

The MPC7410 has extensive on-chip test capability including the following:

- Debug control/observation (COP)
- Boundary scan (standard IEEE Std. 1149.1a-1993 (JTAG) compliant interface)
- Support for manufacturing test
- Support for the following standard JTAG instructions:
  - BYPASS
  - EXTEST
  - SAMPLE/PRELOAD
  - CLAMP
  - HIGHZ

The COP and boundary-scan logic are not used under typical operating conditions. Detailed discussion of the MPC7410 test functions is beyond the scope of this document; however, sufficient information has been provided to allow the system designer to disable the test functions that would impede normal operation.

The JTAG/COP interface is shown in Figure 9-28. For more information, refer to *IEEE Standard Test Access Port and Boundary Scan Architecture IEEE STD 1149-1a-1993*.

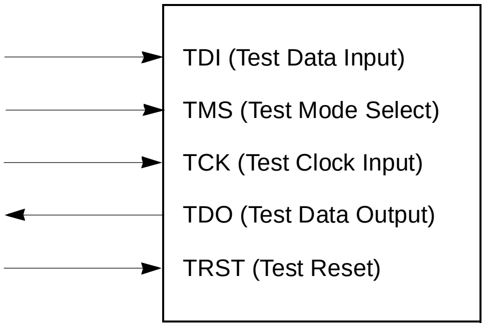


Figure 9-28. IEEE Std. 1149.1a-1993 Compliant Boundary-Scan Interface

Chapter 10
Power Management

The MPC7410 is designed for low-power operation. It provides both automatic and program-controlled power reduction modes. The MPC7400 additionally supports a thermal assist unit (TAU) that allows on-chip thermal measurement and management for portable systems. Note that the MPC7410 does not support the thermal assist unit. This chapter describes the operation of the MPC7410 power and the thermal management features of the MPC7400.

10.1 Dynamic Power Management

Dynamic power management automatically supplies or withholds power to execution units individually, based upon the contents of the instruction stream.

Because CMOS circuits consume negligible power when they are not switching, stopping the clock to an execution unit effectively eliminates its power consumption. Each MPC7410 execution unit has an independent clock input that is controlled automatically on a clock-by-clock basis; for example, clocking is withheld from the floating-point unit if no floating-point instructions are being executed. The operation of dynamic power management is transparent to software and any external hardware.

Dynamic power management is enabled by setting HID0[DPM], as described in Section 2.1.5.1, "Hardware Implementation-Dependent Register 0 (HID0)."

10.2 Programmable Power Modes

Table 10-1 describes the four programmable power modes—full power, doze, nap, and sleep.

Table 10-1. Programmable Power Modes

Mode	Functioning Units	Activation Method	Full-Power Wake-Up Method
Full power (DPM disabled)	All units active	—	—
Full power (DPM enabled)	Requested logic by demand	Instruction dispatch	—
Doze	Bus snooping Data cache as needed Decrementer timer	Software	External asynchronous exceptions Decrementer interrupt Performance monitor interrupt Thermal management interrupt Reset

Table 10-1. Programmable Power Modes (continued)

Mode	Functioning Units	Activation Method	Full-Power Wake-Up Method
Nap	Snooping enabled by negating $\overline{\text{QACK}}$ Decrementer timer	Hardware and software	External asynchronous exceptions Decrementer interrupt Performance monitor interrupt Thermal management interrupt Reset
Sleep	None	Hardware and software	External asynchronous exceptions Performance monitor interrupt Thermal management interrupt Reset

Software selects these modes by setting one (and only one) of the three power saving mode bits in HID0. Hardware can enable a power management state through external asynchronous interrupts. Such a hardware interrupt transfers the program flow to the system management interrupt (SMI) exception handler, which then sets the appropriate power-saving mode. A decrementer interrupt can be used to return to full power from nap or doze mode after a predetermined duration.

Figure 10-1 shows the MPC7410 power management state diagram. Because bus snooping is disabled in sleep mode, a hardware handshake is provided to ensure coherency before the MPC7410 enters this power management mode.

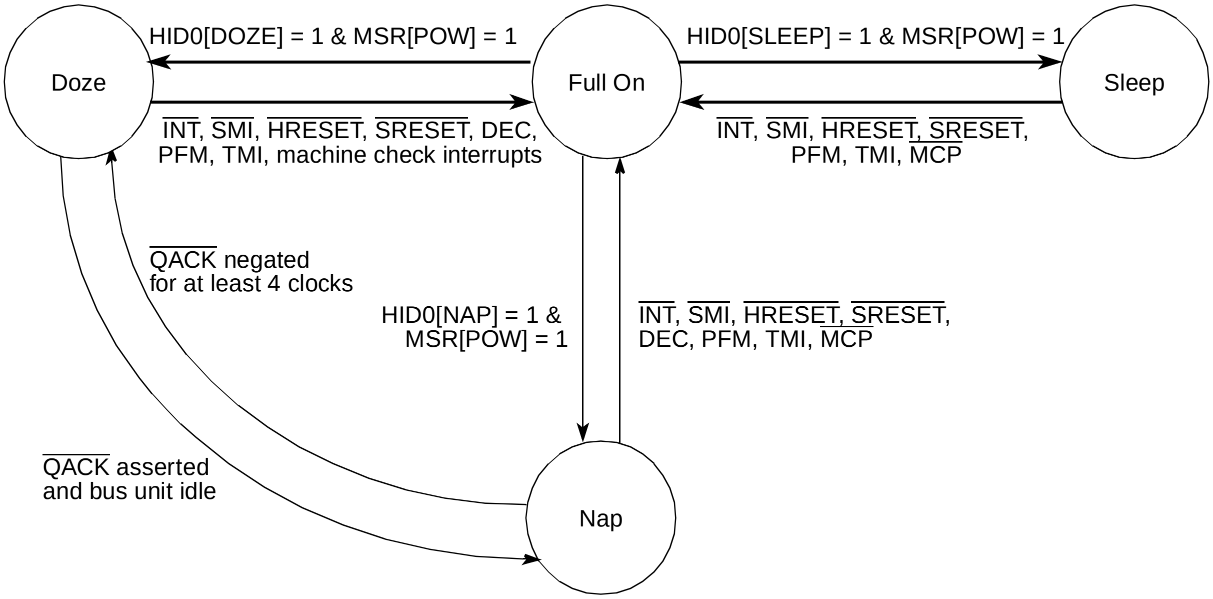

Doze
Full On
Sleep
Nap
HID0[DOZE] = 1 & MSR[POW] = 1
INT, SMI, HRESET, SRESET, DEC,
PFM, TMI, machine check interrupts
HID0[SLEEP] = 1 & MSR[POW] = 1
INT, SMI, HRESET, SRESET,
PFM, TMI, MCP
QACK negated
for at least 4 clocks
HID0[NAP] = 1 &
MSR[POW] = 1
INT, SMI, HRESET, SRESET,
DEC, PFM, TMI, MCP
QACK asserted
and bus unit idle

Figure 10-1. Power Management State Diagram

The following sections describe the characteristics of the power management modes, the requirements for entering and exiting the various modes, and the capabilities available for each mode.

10.2.1 Full-Power Mode with Dynamic Power Management Disabled

Full-power mode with dynamic power management disabled is selected when the HID0[DPM] = 0 and has the following characteristics:

- Default state after power-up and $\overline{\text{HRESET}}$
- All functional units operate at full processor speed at all times

10.2.2 Full-Power Mode with Dynamic Power Management Enabled

Full-power mode with dynamic power management enabled (HID0[DPM] = 1) provides on-chip power management without affecting functionality or performance, and has the following characteristics:

- Functional units are clocked only when needed.
- Required functional units operate at full processor speed.
- No software or hardware intervention is required after the mode is set.
- Power management is transparent to software and hardware.

10.2.3 Doze Mode

Doze mode disables most functional units but maintains cache coherency by enabling the bus interface unit and snooping (L1 and L2 caches and snooping logic). A snoop hit causes the MPC7410 to enable the caches, copy the data back to memory, disable the caches, and fully return to the doze state.

Note the following with respect to doze mode:

- Most functional units are disabled (excluding thermal assist unit, performance monitor, bus snooping, time base, and decrementer).
- Phase-locked loop (PLL) and delay-locked loop (DLL) are running and locked to SYSCLK.

10.2.3.1 Entering Doze Mode

Doze mode is entered by setting the doze bit (HID0[8] = 1), clearing the nap and sleep bits (HID0[9] and HID0[10] = 0), and setting MSR[POW]. The MPC7410 enters doze mode after several processor clocks.

10.2.3.2 Returning to Full-Power Mode from Doze Mode

The MPC7410 returns to full-power mode when $\overline{\text{INT}}$, $\overline{\text{SMI}}$, or $\overline{\text{MCP}}$ are asserted, when a decrementer or performance monitor or thermal management interrupt occurs, or when a hard or soft reset occurs. Transition to full-power state takes only a few processor cycles.

10.2.4 Nap Mode

The nap mode disables the MPC7410 but maintains clocking for the PLL and DLL outputs, the L2 clock pins (L2CLK_OUT[A–B] and L2SYNC_OUT), the time base/decrementer, thermal unit, and performance monitor. The time base can be used to restore the MPC7410 to full-power state after a programmed period.

As Figure 10-1 shows, the MPC7410 supports switching between nap and doze modes, allowing snooping to happen even if the MPC7410 spends most of its time in nap mode. This is described further in Section 10.2.4.2, "Nap Mode Bus Snooping Sequence."

To maintain data coherency, bus snooping is disabled for nap and sleep modes through a hardware handshake sequence using the quiesce request ($\overline{\text{QREQ}}$) and quiesce acknowledge ($\overline{\text{QACK}}$) signals. The MPC7410 asserts $\overline{\text{QREQ}}$ to indicate it is ready to disable bus snooping and enter nap or sleep mode. When the system ensures that snooping is no longer necessary, it asserts $\overline{\text{QACK}}$ and the MPC7410 enters nap mode. If the system determines that a bus snoop cycle is required while in nap or sleep mode, $\overline{\text{QACK}}$ is deasserted to the MPC7410 for at least 4 bus clock cycles and the MPC7410 can then respond to a snoop cycle. Asserting $\overline{\text{QACK}}$ after the snoop cycle again disables snooping; the MPC7410 returns to nap or sleep mode after the snoop completes. Power dissipation while in nap mode with $\overline{\text{QACK}}$ deasserted is the same as the power dissipation while in doze mode.

In nap mode, the DLL remains locked to enable a quick recovery to full-power mode without waiting for the DLL to relock. Additionally, the MPC7410's L2 cache interface provides the L2ZZ signal to drive external SRAM into a low-power mode when nap or sleep modes are invoked. To enable L2ZZ, set L2CR[CTL]. Note that if bus snooping is to be performed through the negation of $\overline{\text{QACK}}$, L2CR[CTL] should always be cleared.

Note the following with respect to nap mode:

- The time base, decrementer, and thermal assist unit are still enabled.
- Most functional units are disabled.
- All nonessential input receivers are disabled.
- PLL and DLL are running and locked to SYSCLK.

10.2.4.1 Entering Nap Mode

Nap mode is entered by setting the nap bit (HID0[9] = 1) and MSR[POW] and clearing the doze and sleep bits (HID0[8] and HID0[10] = 0). At this point, the MPC7410 asserts $\overline{\text{QREQ}}$, and the system asserts $\overline{\text{QACK}}$. The MPC7410 enters nap mode after several processor clocks.

10.2.4.2 Nap Mode Bus Snooping Sequence

The MPC7410 also allows dynamic switching between nap and doze mode to allow use of nap mode without sacrificing hardware snoop coherency. For this operation, negating $\overline{\text{QACK}}$ at any time for at least 4 bus cycles guarantees that the MPC7410 has changed from nap to doze mode in order to snoop. Reasserting $\overline{\text{QACK}}$ then allows the MPC7410 to return to nap mode. $\overline{\text{QACK}}$ can be reasserted one cycle (or more) after $\overline{\text{TS}}$ is asserted for the snoop. The MPC7410 re-enters nap mode after it completes all operations necessary to service the snoop (assuming that $\overline{\text{QACK}}$ was not negated for eight consecutive cycles again). Nap mode resumes within a few cycles of the snoop's address tenure if no snoop push is required. If a snoop push is required, the MPC7410 does not re-enter nap mode until a few cycles after the push's data tenure ends.

It is always necessary to negate $\overline{\text{QACK}}$ for 4 consecutive cycles before presenting a napping MPC7410 with a snoop, regardless of how many cycles $\overline{\text{QACK}}$ had been asserted previously. Because of this 4-cycle requirement, a system could try to predict bursts of snoops by negating $\overline{\text{QACK}}$ for, say, 32 or 64 clocks every time a snoop is presented. In that way, if a subsequent snoop is within tens of cycles of the previous one, the 4-cycle requirement would have already been met for that snoop.

The process for handling bus snooping is described as follows:

1. The system negates $\overline{\text{QACK}}$ for four or more bus clock cycles.
2. The MPC7410 snoops address tenures on the bus.
3. The system asserts $\overline{\text{QACK}}$ to restore full nap mode.

10.2.4.3 Returning to Full-Power Mode

The MPC7410 can be returned to full-power mode by asserting $\overline{\text{INT}}$, $\overline{\text{SMI}}$, or $\overline{\text{MCP}}$ by taking a decrementer, performance monitor, or thermal management interrupt, or by asserting hard reset or soft reset. The transition to full-power takes only a few processor cycles.

10.2.4.4 Sleep Mode

Sleep mode consumes the least amount of power of the four modes because all functional units (except the thermal assist and performance monitor units) are disabled. To maximize power conservation, the PLL may be disabled in the following two ways:

- Placing the PLL_CFG signals in PLL bypass mode and disabling SYSCLK. Note that forcing SYSCLK into a static state does not disable the MPC7410's PLL, which continues to operate internally at an undefined frequency unless placed in PLL bypass mode. Additionally, if the PLL is enabled, the L2 cache interface DLL remains locked and the five L2 clock signals remain active. The DLL is disabled by clearing L2CR[L2E].
- Placing the PLL_CFG signals in kill mode. In this case, SYSCLK can continue to operate.

Due to the fully static design of the MPC7410, the internal processor state is preserved when no internal clock is present. Because the time base and decrementer are disabled while the MPC7410 is in sleep mode, the MPC7410's time base contents must be updated from an external time base after exiting sleep mode if accurate time-of-day is required. Before entering sleep mode, the MPC7410 asserts $\overline{\text{QREQ}}$ to indicate that it is ready to disable bus snooping. When the system ensures that snooping is no longer necessary, it asserts $\overline{\text{QACK}}$ and the MPC7410 enters sleep mode.

Note the following with respect to sleep mode:

- All functional units are disabled (including bus snooping and time base).
- All nonessential input receivers are disabled:
  - — Internal clock regenerators are disabled.
  - — PLL and DLL are still running (see below).
- PLL and DLL may be disabled and SYSCLK may be removed while in sleep mode.

10.2.4.5 Entering Sleep Mode

Sleep mode is entered by setting the sleep bit (HID0[10]) and MSR[POW]) and clearing the doze and nap bits (HID0[8] and HID0[9]). The MPC7410 asserts $\overline{\text{QREQ}}$ and the system asserts $\overline{\text{QACK}}$. The MPC7410 enters sleep mode after several processor clocks.

10.2.4.6 Returning to Full-Power Mode

When the processor is recovering from sleep mode, SYSCLK must be enabled before the PLL if it had been stopped in sleep mode. If the PLL had not been disabled after PLL start-up and re-lock time (indicated in the hardware specifications) has elapsed, the system logic should assert one of the sleep mode recovery signals, such as $\overline{\text{INT}}$ or $\overline{\text{SMI}}$. After the PLL is enabled, the DLL is reconfigured and the DLL relock time has elapsed (640 L2 clock cycles), the L2 cache may be reenabled through the L2CR.

10.2.5 Power Management Software Considerations

Because the MPC7410 is a dual-issue processor with out-of-order execution capability, care must be taken in how the power management mode is entered. Furthermore, nap and sleep modes require all outstanding bus operations to be completed before these power management modes are entered. Normally, during system configuration time, a power management mode would be selected by setting the appropriate HID0 mode bit. Later on, the power management mode is invoked by setting MSR[POW]. To ensure a clean transition into and out of a power management mode, execute the following code sequence:

```
         dssall
         sync
         mtmsr[POW = 1]
         isync
loop:    b loop
```

The **dssall** instruction is needed to kill any outstanding stream touch instructions, which are not killed by a **sync**.

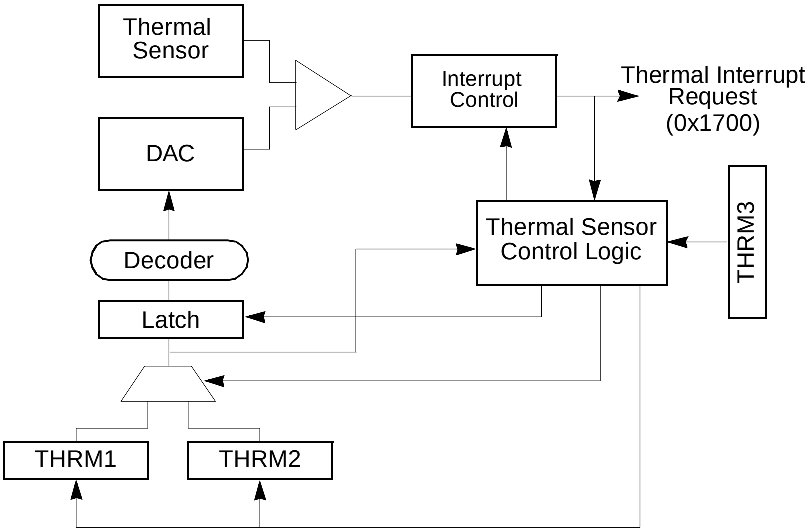
10.3 Thermal Assist Unit (TAU)—MPC7400 Only

With the increasing power dissipation of high-performance processors and operating conditions that span a wider range of temperatures than desktop systems, thermal management becomes an essential part of system design to ensure reliable operation of portable systems. One key aspect of thermal management is ensuring that the junction temperature of the microprocessor does not exceed the operating specification. While the case temperature can be measured with an external thermal sensor, the thermal constant from the junction to the case can be large, and accuracy can be a problem. This may lead to lower overall system performance due to the necessary compensation to alleviate measurement deficiencies.

The MPC7400 provides the system designer an efficient means of monitoring junction temperature through the incorporation of an on-chip thermal sensor and programmable control logic to enable a thermal management implementation tightly coupled to the processor for improved performance and reliability.

10.3.1 Thermal Assist Unit Overview

The on-chip thermal assist unit, shown in Figure 10-2, consists of a thermal sensor, a digital-to-analog converter (DAC), a comparator, control logic, and three dedicated SPRs.



Thermal Sensor
DAC
Interrupt Control
Thermal Interrupt Request (0x1700)
Thermal Sensor Control Logic
THRM3
Decoder
Latch
THRM1
THRM2

Figure 10-2. Thermal Assist Unit Block Diagram

The thermal assist unit provides thermal control by periodically comparing the MPC7400's junction temperature against user-programmed thresholds, and generating a thermal management interrupt if the threshold values are reached. The thermal assist unit also enables the user to determine the junction temperature through a software successive approximation routine.

The thermal assist unit is controlled through three supervisor-level SPRs, accessed through the **mtspr**/**mfspr** instructions. Two SPRs (THRM1 and THRM2) provide temperature threshold values that can be compared to the junction temperature value, and control bits that enable comparison and thermal interrupt generation. The third SPR (THRM3) provides a thermal assist unit enable bit and a sample interval timer. Note that all the bits in THRM1, THRM2, and THRM3 are cleared during a hard reset, and the thermal assist unit remains idle and in a low-power state until configured and enabled.

The fields in the THRM1 and THRM2 SPRs are described in Table 10-2.

Table 10-2. THRM1 and THRM2 Field Descriptions

Bits	Field	Description
0	TIN	Thermal management interrupt bit. Read only. Set if the thermal sensor output crosses the threshold specified in the SPR. The state of this bit is valid only if TIV is set. The interpretation of the TIN bit is controlled by the TID bit.
1	TIV	Thermal management interrupt valid. Read only. This bit is set by the thermal assist logic to indicate that the thermal management interrupt (TIN) state is valid.
2–8	Threshold	Threshold value that the output of the thermal sensor is compared to. The threshold range is between 0• and 127•C, and each bit represents 1•C. Note that this is not the resolution of the thermal sensor.

Table 10-2. THRM1 and THRM2 Field Descriptions

Bits	Field	Description
9–28	—	Reserved, for future use. The MPC7400 always returns zeros on a read of these bits. For maximum flexibility with other implementations, system software should clear these bits when writing to the ICTC.
29	TID	Thermal management interrupt direction bit. Selects the result of the temperature comparison to set TIN. If TID = 0, TIN is set and an interrupt occurs if the junction temperature exceeds the threshold. If TID = 1, TIN is set and an interrupt is indicated if the junction temperature is below the threshold.
30	TIE	Thermal management interrupt enable. Enables assertion of the thermal management interrupt signal. The thermal management interrupt is maskable by the MSR[EE] bit. If TIE = 0 and THRM*n* is valid, the TIN bit records the status of the junction temperature vs. threshold comparison without asserting an interrupt signal. This feature allows system software to make a successive approximation to estimate the junction temperature.
31	V	SPR valid bit. Set to indicate that the SPR contains valid threshold, TID, and TIE control bits. Setting THRM1[V], THRM2[V], and THRM3[E] enables operation of the thermal sensor.

The bit fields in the THRM3 SPR are described in Table 10-3.

Table 10-3. THRM3 Bit Field Settings

Bits	Name	Description
0–17	—	Reserved for future use. System software should clear these bits.
18–30	SITV	Sample interval timer value. Number of elapsed processor clock cycles before a junction temperature vs. threshold comparison result is sampled for TIN bit setting and interrupt generation. This is necessary due to the thermal sensor, DAC, and the analog comparator settling time being greater than the processor cycle time. The value should be configured to allow a 20-µs sampling interval or to the maximum value, whichever is lower.
31	E	Enables the thermal sensor compare operation if either THRM1[V] or THRM2[V] = 1.

10.3.2 Thermal Assist Unit Operation

The thermal assist unit can be programmed to operate in single- or dual-threshold modes, which results in the thermal assist unit generating a thermal management interrupt when one or both threshold values are crossed. In addition, with the appropriate software routine, the thermal assist unit can also directly determine the junction temperature. The following sections describe the configuration of the thermal assist unit to support these modes of operation.

10.3.2.1 Thermal Assist Unit Single-Threshold Mode

When the thermal assist unit is configured for single-threshold mode, either THRM1 or THRM2 can be used to contain the threshold value, and a thermal management interrupt is generated when the threshold value is crossed. To configure the thermal assist unit for single threshold operation, set the desired temperature threshold, TID, TIE, and V bits for either THRM1 or THRM2. The unused THRM*n* threshold SPR should be disabled by clearing the V bit. In this discussion THRM*n* refers to the THRM threshold SPR (THRM1 or THRM2) selected to contain the active threshold value.

After setting the desired operational parameters, the thermal assist unit is enabled by setting THRM3[E] and placing a value allowing the desired sample interval in THRM3[SITV]. The THRM3[SITV] setting determines the number of processor clock cycles between input to the DAC and sampling of the comparator output; accordingly, the use of a value smaller than recommended in THRM3[SITV] can cause inaccuracies in the sensed temperature.

If the junction temperature does not cross the programmed threshold, THRM*n*[TIN] is cleared to indicate that no interrupt is required and THRM*n*[TIV] is set to indicate that the TIN bit state is valid. If the threshold value has been crossed, THRM*n*[TIN] and THRM*n*[TIV] are set and a thermal management interrupt is generated if THRM*n*[TIE] and MSR[EE] = 1.

A thermal management interrupt is held asserted internally until recognized by the MPC7400's interrupt unit. Once a thermal management interrupt is recognized, further temperature sampling is suspended, and the THRM*n*[TIN] and THRM*n*[TIV] values are held until an **mtspr** instruction is executed to THRM*n*.

The execution of an **mtspr** instruction to THRM*n* anytime during thermal assist unit operation clears THRM*n*[TIV] and restarts the temperature comparison. Executing an **mtspr** instruction to THRM3 clears THRM1[TIV] and THRM2[TIV] and restarts temperature comparison in THRM*n* if THRM3[E] = 1.

Examples of valid THRM1 and THRM2 bit settings are shown in Table 10-4.

Table 10-4. Valid THRM1 and THRM2 Bit Settings

TIN[1]	TIV[1]	TID	TIE	V	Description
x	x	x	x	0	The threshold in the SPR is not used for comparison.
x	x	x	0	1	Threshold is used for comparison, thermal management interrupt assertion is disabled.
x	x	0	0	1	Set TIN and do not assert thermal management interrupt if the junction temperature exceeds the threshold.
x	x	0	1	1	Set TIN and assert thermal management interrupt if the junction temperature exceeds the threshold.
x	x	1	0	1	Set TIN and do not assert thermal management interrupt if the junction temperature is less than the threshold.
x	x	1	1	1	Set TIN and assert thermal management interrupt if the junction temperature is less than the threshold.
x	0	x	x	1	The state of the TIN bit is not valid.
0	1	0	x	1	The junction temperature is less than the threshold and as a result the thermal management interrupt is not generated for TIE = 1.
1	1	0	x	1	The junction temperature is greater than the threshold and as a result the thermal management interrupt is generated if TIE = 1.
0	1	1	x	1	The junction temperature is greater than the threshold and as a result the thermal management interrupt is not generated for TIE = 1.
1	1	1	x	1	The junction temperature is less than the threshold and as a result the thermal management interrupt is generated if TIE = 1.

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

10.3.2.2 Thermal Assist Unit Dual-Threshold Mode

The configuration and operation of dual-threshold mode is similar to single-threshold mode, except both THRM1 and THRM2 are configured with desired threshold and TID values, and the TIE and V bits are set. When THRM3[E] is set, enabling temperature measurement and comparison, the first comparison is made with THRM1. If no thermal management interrupt results from the comparison, the number of processor cycles specified in THRM3[SITV] elapses, and the next comparison is made with THRM2. If no thermal management interrupt results from the THRM2 comparison, the time specified by THRM3[SITV] again elapses, and the comparison returns to THRM1.

This sequence of comparisons continues until a thermal management interrupt occurs, or the thermal assist unit is disabled. When a comparison results in an interrupt, the comparison with the threshold SPR causing the interrupt is halted, but comparisons continue with the other threshold SPR. Following a thermal management interrupt, the interrupt service routine must read both THRM1 and THRM2 to determine which threshold was crossed. Note that it is possible for both threshold values to have been crossed, in which case the thermal assist unit stops making temperature comparisons until an **mtspr** instruction is executed to one or both of the threshold SPRs.

10.3.2.3 MPC7400 Junction Temperature Determination

While the MPC7400’s thermal assist unit does not implement an analog-to-digital converter to enable the direct determination of the junction temperature, system software can execute a simple successive approximation routine to find the junction temperature.

The thermal assist unit configuration used to approximate the junction temperature is the same required for single-threshold mode, except that the threshold SPR selected has its TIE bit cleared to disable thermal management interrupt generation. Once the thermal assist unit is enabled, the successive approximation routine loads a threshold value into the active threshold SPR, and then continuously polls the threshold SPR’s TIV bit until it is set, indicating a valid TIN bit. The successive approximation routine can then evaluate the TIN bit value, and then increment or decrement the threshold value for another comparison. This process is continued until the junction temperature is determined.

10.3.2.4 Power Saving Modes and Thermal Assist Unit Operation

The static power saving modes (nap, doze, and sleep) allow the processor temperature to be lowered quickly and can be invoked through the use of the thermal assist unit and associated thermal management interrupt. The thermal assist unit remains operational in nap and doze modes, and it remains operational in sleep mode as long as the SYSCLK input remains active. If SYSCLK is made static when sleep mode is invoked, the thermal assist unit is rendered inactive. If the MPC7400 is entering sleep mode with SYSCLK disabled, the thermal assist unit should be configured to disable thermal management interrupts to avoid an unwanted thermal management interrupt when the SYSCLK input signal is restored.

10.4 Instruction Cache Throttling

The MPC7410 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 MPC7400’s thermal assist unit and 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. While the MPC7410 lacks the thermal management unit that the MPC7400 supports, external sensors and logic may be used to invoke an exception which then uses instruction cache throttling to achieve the same goal.

The instruction cache throttling mechanism simply reduces the instruction forwarding rate from the instruction cache to the instruction dispatcher. Normally, the instruction cache forwards four instructions to the instruction dispatcher every clock cycle if all the instructions hit in the cache. For thermal management, the MPC7410 provides a supervisor-level instruction cache throttling control SPR (ICTC). The instruction forwarding rate is reduced by writing a nonzero value into ICTC[FI], and enabling instruction cache throttling by setting ICTC[E]. The overall junction temperature reduction results from dynamic power management reducing the power to the execution units while waiting for instructions to be forwarded from the instruction cache; thus, instruction cache throttling does not provide thermal reduction unless HID0[DPM] = 1. Note that during instruction cache throttling the configuration of the PLL and DLL remain unchanged.

Table 10-5 describes ICTC fields.

Table 10-5. ICTC Field Descriptions

Bits	Name	Description
0–22	—	Reserved, for future use. The MPC7410 always returns zeros on a read of these bits. For maximum flexibility with other implementations, system software should clear these bits when writing to the ICTC.
23–30	FI	Instruction forwarding interval expressed in processor clocks. 0x00—0 clock cycle 0x01—1 clock cycle : : 0xFF—255 clock cycles
31	E	Cache throttling enable 0 Disable instruction cache throttling 1 Enable instruction cache throttling

Chapter 11
Performance Monitor

The architecture defines an optional performance monitor facility that provides the ability to monitor and count predefined events such as processor clocks, misses in the instruction cache, data cache, or L2 cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (that may be an approximation) can be used to trigger the performance monitor exception. Note that some earlier processors implemented the performance monitor facility before it was defined by the architecture.

The performance monitor can be used for the following:

- To increase system performance with efficient software, especially in a multiprocessing system—Memory hierarchy behavior can be monitored and studied in order to develop algorithms that schedule tasks (and perhaps partition them) and that structure and distribute data optimally.
- To characterize processors—Some environments may not be easily characterized by a benchmark or trace.
- To help system developers bring up and debug their systems

AltiVec Technology and the Performance Monitor

The AltiVec technology features do not affect the basic implementation of the performance monitor. However, the performance monitor provides the ability to monitor the following AltiVec operations:

- Individual counts of instructions executed by the AltiVec execution units
- Completed AltiVec load instructions counts
- Individual counts of cycles during which the VSIU, VCIU, VFPU, and VPU had a valid dispatch but invalid operands
- VFPU trap counts that can be generated in Java mode
- **mtvscr** and **mtvrsave** instruction counts
- Setting counts of the VSCR saturate bit
- Completion counts of **dss** and **dssall** instructions

- **dst***x* instruction event counts—dispatches, misses, refreshes, suspensions, premature cancellations, and resumptions
- Cycle counts when the VALU has a valid **mfvscr** dispatch but cannot execute it because it is not at the bottom of the completion queue (CQ)

11.1 Overview

The performance monitor uses the following resources defined by the architecture:

- The performance monitor mark bit in the MSR (MSR[PMM]). This bit identifies programs to be monitored.
- The privilege level bit in the MSR (MSR[PR]). This bit identifies the mode the processor is in (supervisor or user).
- Special-purpose registers (SPRs):
  - The performance monitor counter registers (PMC1–PMC4) are 32-bit counters used to count the times a software-selectable event has occurred. UPMC1–UPMC4 provide user-level read access to these registers.
  - The monitor mode control registers (MMCR0–MMCR2). Control fields in the MMCR*n* registers select events to be counted, determine whether performance monitor exceptions are caused by a time-base register transition, maintain counter overflow, or when $\overline{\text{SMI}}$ is asserted, and specify the conditions under which counting is enabled. UMMCR0–UMMCR2 provide user-level read access to these registers.
  - The sampled instruction address register (SIAR) contains the effective address of the last instruction that completes before the performance monitor exception is generated. USIAR provides user-level read access to the SIAR.
  - Note that the MPC7410 does not implement the sampled data address register (SDAR) or the user-level, read-only USDAR that are defined as optional in the architecture. However, for compatibility with processors that do, those registers can be written to by boot code without causing an exception.
- The performance monitor exception follows the normal PowerPC exception model and has a defined exception vector offset (0x00F00). Its priority is below the trace exception and above the AltiVec unavailable exception.

11.2 Performance Monitor Exception

The performance monitor provides the ability to generate a performance monitor exception triggered by an enabled condition or event. This exception is triggered by an enabled condition or event defined as follows:

- A PMC*n* register overflow condition occurs:
  - — MMCR0[PMC1CE] and PMC1[OV] are both set
  - — MMCR0[PMCjCE] and PMCj[OV] are both set (j > 1)
- A time-base event—MMCR0[TBEE] = 1 and the TBL bit specified in MMCR0[TBSEL] changes from 0 to 1
- An $\overline{\text{SMI}}$ event—MMCR2[SMINTEN] = 1 and $\overline{\text{SMI}}$ is asserted.

MMCR0[PMXE] must be set for any of these conditions to signal a performance monitor exception.

Although the performance monitor exception may occur with MSR[EE] = 0, the exception is not taken until MSR[EE] = 1.

As a result of a performance monitor exception being generated, the performance monitor saves in the SIAR the effective address of the last instruction completed before the exception is taken. Note that SIAR is not updated if performance monitor counting has been disabled by setting MMCR0[0].

The performance monitor can receive a performance monitor exception request from an off-chip performance monitor or device. This is accomplished by setting the mask bit in MMCR2[SMINTEN] and asserting $\overline{\text{SMI}}$. Under this condition, the MPC7410 takes a performance monitor exception rather than an SMI exception.

The priority of the performance monitor exception is below the trace exception and above the AltiVec unavailable exception. See Section 4.2, “Exception Recognition and Priorities,” for a list of exception priorities.

Exception handling for the performance monitor exception is described in Section 4.6.13, “Performance Monitor Exception (0x00F00).”

11.2.1 Performance Monitor Signals

The $\overline{\text{PMON\_IN}}$ signal is used by the performance monitor event MMCR0[PMC1SEL] = 7 (0b000_0111) to count the number of times the $\overline{\text{PMON\_IN}}$ signal transitions from negated to asserted. Note that this event is enabled in the performance monitor control registers (MMCR0, or MMCR1, PMC*n*) and must be enabled in order for this event to be monitored.

The $\overline{\text{PMON\_OUT}}$ signal is asserted by the performance monitor when a performance monitor threshold or negative counter condition has been reached whether or not performance monitor exceptions are enabled; that is, the setting of MMSR0[PMXE] does not affect the function of this signal.

11.2.2 Using Timebase Event to Trigger or Freeze a Counter or Generate an Exception

The use of the trigger and freeze counter conditions depends on these enabled conditions and events. When MMCR0[TBEE] (timebase enable event) is 1, a timebase transition is generated to the performance monitor if the TBL bit specified in MMCR0[TBSEL] changes from 0 to 1. Timebase transition events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or generate an exception (MMCR0[PMXE]).

Changing the bits specified in MMCR0[TBSEL] while MMCR0[TBEE] is set may cause a false 0 to 1 transition that signals the specified action (freeze, trigger, or exception) to occur immediately.

11.3 Performance Monitor Registers

The following sections describe the registers used by the performance monitor. Note that reading and writing to the performance monitor SPRs do not synchronize the processor.

An explicit synchronization instruction, such as **sync**, should be placed before and after an **mfspr** or **mtspr** of one of these registers to guarantee an accurate count.

11.3.1 Performance Monitor Special-Purpose Registers

The performance monitor incorporates the SPRs listed in Table 11-1 and Table 11-2. The supervisor-level registers in Table 11-1 are accessed through the **mtspr** and **mfspr** instructions.

Table 11-1. Performance Monitor SPRs—Supervisor Level

SPR Number	spr[5–9] || spr[0–4]	Register Name
944	0b11101 10000	Monitor mode control register 2—MMCR2
951	0b11101 10111	Breakpoint address mask register—BAMR
952	0b11101 11000	Monitor mode control register 0—MMCR0
953	0b11101 11001	Performance monitor counter register 1—PMC1
954	0b11101 11010	Performance monitor counter register 2—PMC2
955	0b11101 11011	Sampled instruction address register—SIAR
956	0b11101 11100	Monitor mode control register 1—MMCR1
957	0b11101 11101	Performance monitor counter register 3—PMC3
958	0b11101 11110	Performance monitor counter register 4—PMC4

The user-level registers in Table 11-2 are read-only and are accessed with the **mfspr** instruction. Attempting to write to one of these registers in either supervisor or user mode causes a program exception.

Table 11-2. Performance Monitor SPRs—User Level (Read-Only)

SPR Number	spr[5–9] || spr[0–4]	Register Name
928	0b11101 00000	User monitor mode control register 2—UMMCR2
935	0b11101 00111	User breakpoint address mask register—UBAMR
936	0b11101 01000	User monitor mode control register 0—UMMCR0
937	0b11101 01001	User performance monitor counter register 1—UPMC1
938	0b11101 01010	User performance monitor counter register 2—UPMC2
939	0b11101 01011	User sampled instruction address register—USIAR
940	0b11101 01100	User monitor mode control register 1—UMMCR1
941	0b11101 01101	User performance monitor counter register 3—UPMC3
942	0b11101 01110	User performance monitor counter register 4—UPMC4

11.3.2 Monitor Mode Control Register 0 (MMCR0)

The monitor mode control register 0 (MMCR0), shown in Figure 11-1 is a 32-bit SPR provided to specify events to be counted and recorded. If the state of MSR[PR] and MSR[PMM] matches a state specified in MMCR0, then counting is enabled; see Section 11.4, “Event Counting,” for further details. The MMCR0 can be accessed only in supervisor mode. User-level software can read the contents of MMCR0 by issuing an **mfspr** instruction to UMMCR0, described in Section 11.3.2.1, “User Monitor Mode Control Register 0 (UMMCR0).”

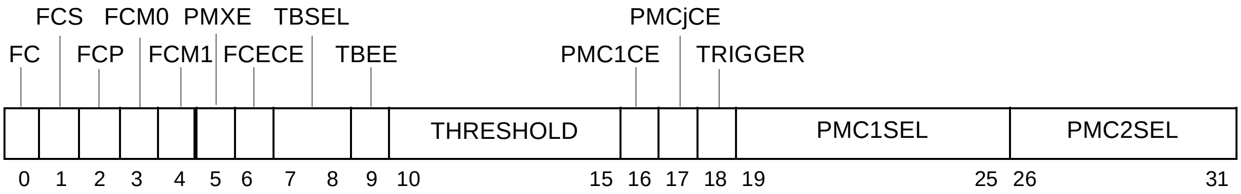


Figure 11-1. Monitor Mode Control Register 0 (MMCR0)

This register is automatically cleared at power-up. Reading this register does not change its contents. Table 11-3 describes the fields of MMCR0.

Table 11-3. MMCR0 Field Descriptions

Bits	Name	Description
0	FC	Freeze counters 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented (performance monitor counting is disabled). The processor automatically sets this bit when an enabled condition or event occurs and MMCR0[FCECE] = 1. Note that SIAR is not updated if performance monitor counting is disabled.
1	FCS	Freeze counters in supervisor mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 0.
2	FCP	Freeze counters in user mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 1.
3	FCM1	Freeze counters while mark = 1 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 1.
4	FCM0	Freeze counters while mark = 0 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 0.
5	PMXE	Performance monitor exception enable 0 Performance monitor exceptions are disabled. 1 Performance monitor exceptions are enabled until a performance monitor exception occurs; at that time, MMCR0[PMXE] is automatically cleared. Software can clear PMXE to prevent performance monitor exceptions. Software can also set PMXE and then poll it to determine whether an enabled condition or event occurred.
6	FCECE	Freeze counters on enabled condition or event 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are incremented (if permitted by other MMCR bits) until an enabled condition or event occurs when MMCR0[TRIGGER] = 0, at that time MMCR0[FC] is set. If the enabled condition or event occurs when MMCR0[TRIGGER] = 1, FCECE is treated as if it were 0. The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in Section 11.2, "Performance Monitor Exception."
7–8	TBSEL	Time-base selector. Selects the time-base bit that can cause a time-base transition event (the event occurs when the selected bit changes from 0 to 1). 00 TBL[31] 01 TBL[23] 10 TBL[19] 11 TBL[15] Time-base transition events can be used to periodically collect information about processor activity. In multiprocessor systems in that 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.

Table 11-3. MMCR0 Field Descriptions (continued)

Bits	Name	Description
9	TBEE	Time-base event enable 0 Time-base transition events are disabled. 1 Time-base transition events are enabled. A time-base transition is generated to the performance monitor if the TB bit specified in MMCR0[TBSEL] changes from 0 to 1. Time-base transition events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]). Changing the bits specified in MMCR0[TBSEL] while MMCR0[TBEE] is enabled may cause a false 0 to 1 transition that signals the specified action (freeze, trigger, or exception) to occur immediately.
10–15	THRESHOLD	Threshold Contains a threshold value, such that only events that exceed the value are counted (PMC1 events 11, 19, and 20). By varying the threshold value, software can obtain a profile of the characteristics of the events subject to the threshold. For example, if PMC1 counts cache misses that the duration exceeds the threshold value, software can obtain the distribution of cache miss durations for a given program by monitoring the program repeatedly using a different threshold value each time. Note that MMCR2[THRESHMULT] selects whether this value is multiplied by 2 or 32.
16	PMC1CE	PMC1 condition enable. Controls whether counter negative conditions due to a negative value in PMC1 are enabled. 0 Counter negative conditions for PMC1 are disabled. 1 Counter negative conditions for PMC1 are enabled. These events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]).
17	PMCjCE	PMCj condition enable. Controls whether the counter negative conditions due to a negative value in any PMCj (that is, in any PMC except PMC1) are enabled. 0 Counter negative conditions for all PMCjs are disabled. 1 Counter negative conditions for all PMCjs are enabled. These events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]).

Table 11-3. MMCR0 Field Descriptions (continued)

Bits	Name	Description
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 PMCjs are not incremented until PMC1 is negative or an enabled condition or event occurs, at that time the PMCjs 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 PMCjs when PMC1 becomes negative without causing a performance monitor exception. Then freeze all PMCs (and optionally cause a performance monitor exception) when a PMCj becomes negative. The PMCjs then reflect the events that occurred after PMC1 became negative and before PMCj becomes negative. This use requires the following MMCR0 bit settings: –TRIGGER = 1 –PMC1CE = 0 –PMCjCE = 1 –TBEE = 0 –FCECE = 1 –PMXE = 1 (if a performance monitor exception is desired) • Resume counting in the PMCjs when PMC1 becomes negative and cause a performance monitor exception without freezing any PMCs. The PMCjs then reflect the events that occurred between the time PMC1 became negative and the time the exception handler reads them. This use requires the following MMCR0 bit settings: –TRIGGER = 1 –PMC1CE = 1 –TBEE = 0 –FCECE = 0 –PMXE = 1 The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in Section 11.2, “Performance Monitor Exception.”
19–25	PMC1SEL	PMC1 selector. Contains a code (one of at most 128 values) that identifies the event to be counted in PMC1. See Table 11-9.
26–31	PMC2SEL	PMC2 selector. Contains a code (one of at most 64 values) that identifies the event to be counted in PMC2. See Table 11-10.

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

11.3.2.1 User Monitor Mode Control Register 0 (UMMCR0)

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

11.3.3 Monitor Mode Control Register 1 (MMCR1)

The monitor mode control register 1 (MMCR1) functions as an event selector for performance monitor counter registers 3and 4 (PMC3 and, PMC4). The MMCR1 register is shown in Figure 11-2.

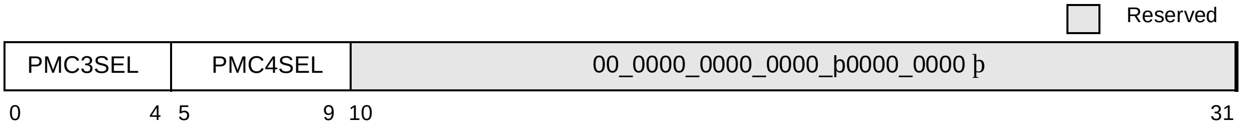


Figure 11-2. Monitor Mode Control Register 1 (MMCR1)

Bit settings for MMCR1 are shown in Table 11-4. The corresponding events are described in Section 11.3.6, “Performance Monitor Counter Registers (PMC1–PMC4).”

Table 11-4. MMCR1 Field Descriptions

Bits	Name	Description
0–4	PMC3SEL	PMC3 selector. Contains a code (one of at most 32 values) that identifies the event to be counted in PMC3. See Table 11-11.
5–9	PMC4SEL	PMC4 selector. Contains a code (one of at most 32 values) that identifies the event to be counted in PMC4. See Table 11-12.
10–31	—	Reserved

MMCR1 can be accessed with the **mtspr** and **mfspr** instructions using SPR 956. User-level software can read the contents of MMCR1 by issuing an **mfspr** instruction to UMMCR1, described in Section 11.3.4.1, “User Monitor Mode Control Register 2 (UMMCR2).”

11.3.3.1 User Monitor Mode Control Register 1 (UMMCR1)

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

11.3.4 Monitor Mode Control Register 2 (MMCR2)

The monitor mode control register 2 (MMCR2) contains only one bit. This bit is used to scale the value in the MMCR0[THRESHOLD] field for certain threshold events. If MMCR2[THRESMULT] = 0, it scales the value by two times. If MMCR2[THRESMULT] = 1, it scales the value by 32 times.The MMCR2 register is shown in Figure 11-3.

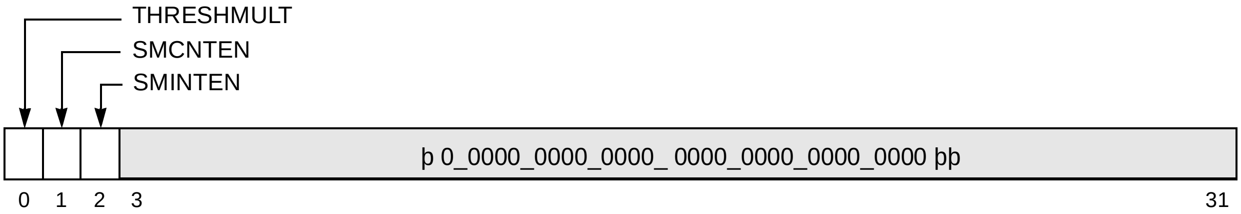


Figure 11-3. Monitor Mode Control Register 2 (MMCR2)

Table 11-5 describes MMCR2 fields.

Table 11-5. MMCR2 Field Descriptions

Bits	Name	Description
0	THRESHMULT	Threshold multiplier. Used to extend the range of the THRESHOLD field, MMCR0[10–15]. 0 Threshold field is multiplied by 2. 1 Threshold field is multiplied by 32.
1	SMCNTEN	SMCNTEN is used to mask the request from a peripheral performance monitor. 0 Ignore PMON_IN. 1 Start counting when PMON_IN is asserted. Note that counting is subject to other enabling control bits in MMCR0.
2	SMINTEN	SMINTEN is used to mask the performance monitor exception request from a peripheral performance monitor. 0 Ignore SMI. 1 When SMI is asserted, take a performance monitoring exception if enabled in MMCR0 and MSR[EE]. This event can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an exception (MMCR0[PMXE]). When SMINTEN = 1, the MPC7410 never takes an SMI.

MMCR2 can be accessed with **mtspr** and **mfspr** using SPR 944. User-level software can read the contents of MMCR2 by issuing an **mfspr** instruction to UMMCR2, described in Section 11.3.4.1, “User Monitor Mode Control Register 2 (UMMCR2).”

11.3.4.1 User Monitor Mode Control Register 2 (UMMCR2)

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

11.3.5 Breakpoint Address Mask Register (BAMR)

The breakpoint address mask register (BAMR), shown in Figure 11-4, is used in conjunction with the events that monitor IABR and DABR hits.

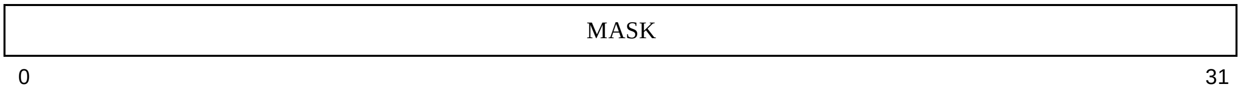


Figure 11-4. Breakpoint Address Mask Register (BAMR)

Table 11-6 describes BAMR fields.

Table 11-6. BAMR Field Descriptions

Bit	Name	Description
0–31	MASK	Used with events (PMC1 events 9 and 10) that monitor IABR and DABR hits The addresses to be compared for an IABR or DABR match are affected by the value in BAMR: • IABR hit (PMC1, event 8) 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] • DABR hit (PMC1, event 9) occurs if DABR_CMP (that is, DABR AND BAMR) = effective_address_compare (that is, EA AND BAMR). DABR_CMP[0–28] = DABR[0–28] AND BAMR[0–28] effective_addr_cmp[0–28] = effective_addr[0–28] AND BAMR[0–28] Be aware that breakpoint events 9 and 10 of PMC1 can be used to trigger ISI and DSI exceptions when the performance monitor detects an enabled overflow. This feature supports debug purposes and occurs only when IABR[30] or DABR[30–31] are set. To avoid taking one of these exceptions, IABR[30] or DABR[30–31] should be cleared.

BAMR can be accessed with **mtspr** and **mfspr** using SPR 951. User-level software can read the contents of BAMR by issuing an **mfspr** instruction to UBAMR, described in Section 11.3.5.1, “User Breakpoint Address Mask Register (UBAMR).”

11.3.5.1 User Breakpoint Address Mask Register (UBAMR)

The contents of BAMR are reflected to UBAMR, that can be read by user-level software. UBAMR can be accessed with the **mfspr** instructions using SPR 935.For synchronization requirements on the BAMR register see Table 2-22.

11.3.6 Performance Monitor Counter Registers (PMC1–PMC4)

PMC1–PMC4, shown in Figure 11-5, are 32-bit counters that can be programmed to generate a performance monitor exception when they overflow.

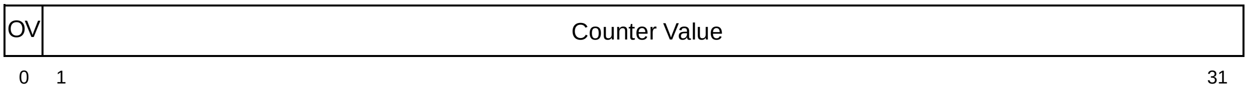


Figure 11-5. Performance Monitor Counter Registers (PMC1–PMC4)

The bits contained in the PMC registers are described in Table 11-7.

Table 11-7. PMCj 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[PMCjCE] 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 PMCj (j > 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[PMCjCE] 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

11.3.6.1 User Performance Monitor Counter Registers (UPMC1–UPMC4)

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

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

11.3.7 Sampled Instruction Address Register (SIAR)

The sampled instruction address register (SIAR) is a supervisor-level register that contains the effective address of the last instruction to complete before the performance monitor exception is generated. The SIAR is shown in Figure 11-6.

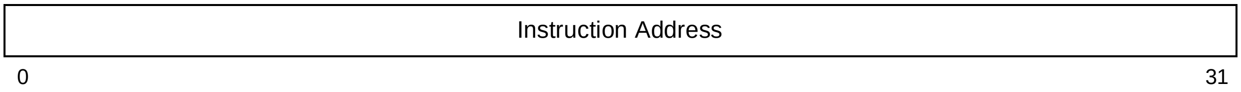


Figure 11-6. Sampled Instruction Address Register (SIAR)

Note that SIAR is not updated:

- if performance monitor counting has been disabled by setting MMCR0[FC] or
- if the performance monitor exception has been disabled by clearing MMCR0[PMXE].

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

11.3.7.1 User Sampled Instruction Address Register (USIAR)

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

11.4 Event Counting

Counting can be enabled if conditions in the processor state match a software-specified condition. Because a software task scheduler may switch a processor’s execution among multiple processes and because statistics only on a particular process may be of interest, a facility is provided to mark a process. The performance monitor bit, MSR[PMM], is used for this purpose. System software may set this bit when a marked process is running. This enables statistics to be gathered only during the execution of the marked process. The states of MSR[PR] and MSR[PMM] together define a state that the processor (supervisor or user) and the process (marked or unmarked) may be in at any time. If this state matches a state specified in the MMCR (the state in which monitoring is enabled), counting is enabled. Table 11-8 lists the states that can be monitored.

Table 11-8. Monitorable States

MSR[PR]	MSR[PMM]	Process Marked	Process UnMarked	User Mode	Supervisor Mode
0	—	—	—	√	√
1	—	—	—	√	—
—	1	√	—	—	—
—	0	—	√	—	—
0	0	—	√	√	√
0	1	√	—	√	√

Table 11-8. Monitorable States

MSR[PR]	MSR[PMM]	Process Marked	Process UnMarked	User Mode	Supervisor Mode
1	0	—	√	√	—
1	1	√	—	√	—

In addition, one of two unconditional counting modes may be specified:

- Counting is unconditionally enabled regardless of the states of MSR[PMM] and MSR[PR]. This can be accomplished by clearing MMCR0[0–4].
- Counting is unconditionally disabled regardless of the states of MSR[PMM] and MSR[PR]. This is done by setting MMCR0[0] = 1. Note that SIAR is not updated if MMCR0[0] = 1.

11.5 Event Selection

Event selection is handled through the MMCR*n* registers, described in subsequent tables.

- The four event-select fields in MMCR0 and MMCR1 are as follows:
  - — MMCR0[PMC1SEL]—PMC1 input selector, 128 events selectable. See Table 11-9.
  - — MMCR0[PMC2SEL]—PMC2 input selector, 64 events selectable. See Table 11-10.
  - — MMCR1[PMC3SEL]—PMC3 input selector, 32 events selectable. See Table 11-11.
  - — MMCR1[PMC4SEL]—PMC4 input selector, 32 events selectable. See Table 11-12.
- In Table 11-9 through Table 11-12, a correlation is established between each counter, events to be traced, and the pattern required for the desired selection.
- As shown in Table 11-9 through Table 11-12, the first five events are common to all four counters and are considered to be reference events. These are as follows:
  - — 00000—Register holds current value
  - — 00001—Number of processor cycles
  - — 00010—Number of completed instructions, not including folded branches
  - — 00011—Number of times the TBL bit transitions from 0 to 1. The TBL bit is specified through MMCR0[TBSEL]
    - – 00 = uses the TBL[31] bit to count
    - – 01 = uses the TBL[23] bit to count
    - – 10 = uses the TBL[19] bit to count
    - – 11 = uses the TBL[15] bit to count
  - — 00100—Number of instructions dispatched: 0, 1,or 2, per cycle

11.5.1 PMC1 Events

The event to be monitored can be chosen by setting MMCR0[PMC1SEL]. The selected events are counted beginning when MMCR0 is set until either MMCR0 is reset or a performance monitor exception is generated. Table 11-9 lists the selectable events and their encodings.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings

Number	Event	Description
0 (000_0000)	Nothing	Register counter holds current value
1 (000_0001)	Processor cycles	Counts every processor cycle
2 (000_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Does not include folded branches. 0, 1, or 2 instructions per cycle.
3 (000_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL]. 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (000_0100)	Instructions dispatched	Counts dispatched instructions. 0, 1, or 2 instructions per cycle.
5 (000_0101)	**eieio** instructions completed	Counts **eieio** instructions completed
6 (000_0110)	ITLB table search cycles	Counts the number of cycles spent performing ITLB table search operations
7 (000_0111)	VPU instructions completed	Counts instructions completed by the VPU (0, 1, or 2 instructions per cycle)
8 (000_1000)	AltiVec ALU's simple integer subunit wait	Counts the number of cycles the ALU's simple integer subunit had a valid dispatch but invalid operands.
9 (000_1001)	Instruction breakpoint matches	Counts the number of times that the address of an instruction being completed matches the address in the IABR. Both the address in the IABR and the completed instruction address are masked by BAMR. Some instruction addresses (such as folded branches) never produce hits. IABR_hit = [(IABR & BAMR) == (EA & BAMR)] See Table 11-6 for more information on this event.
10 (000_1010)	Data breakpoint matches	Counts the number of matches. DABR hit = [(DABR[0–28] & BAMR[0–28]) == (EA & BAMR)]. See Table 11-6 for more information on this event.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
11 (000_1011)	Load miss latency over threshold	Counts the number of times a load that misses in the L1 data cache needs more than the threshold cycles to write back to the rename registers. The cycle count compared to the threshold value represents the number of cycles starting with allocation in the load-fold-queue or data reload table. Note that MMCR2[THRESMULT] determines whether the value in the MMCR0[THRESHOLD] field is multiplied by 2 or 32. Note that misaligned loads require two cache accesses. The miss latency threshold is measured from the first access that misses up until both halves of load are written to the rename registers. Load multiple/string instructions can generate many separate cache accesses. For these loads, the threshold is measured starting with the first access that misses until the entire multiple or string operation is complete.
12 (000_1100)	Unresolved branches	Counts branches that are unresolved when processed.
13 (000_1101)	Dispatch stall due to speculative branches	Counts the number of cycles the dispatcher halts due to a second unresolved branch in the instruction stream
14 (000_1110)	**mfvscr** synchronization	Counts the number of cycles the VALU had a valid **mfvscr** dispatch but could not execute it because it is not at the bottom three entries of the completion queue.
15 (000_1111)	mtvscr instructions completed	Counts completed **mtvscr** instructions
16 (001_0000)	mtvrsave instructions completed	Counts completed **mtvrsave** instructions
17 (001_0001)	VSCR[SAT] set	Counts whenever VSCR[SAT] goes from 0 to 1
18 (001_0010)	Clean L1 castouts to L2	Counts the number of times the L1 casts out clean data to L2.
19 (001_0011)	Instruction TLB search latency over threshold	Counts when an instruction TLB search requires more than the threshold cycles to complete. Includes search operations that do not find a matching PTE. Note that MMCR2[0] determines whether the value in the MMCR0 threshold field is multiplied by 2 or 32.
20 (001_0100)	Data TLB search latency over threshold	Counts when a data TLB search requires more than the threshold cycles to complete. Includes search operations that do not find a matching PTE and search operations caused by **dst***x* instructions. Note that MMCR2[0] determines whether the value in the MMCR0 threshold field is multiplied by 2 or 32.
21 (001_0101)	Instruction TLB search latency over threshold	Counts when a instruction TLB search requires more than the threshold cycles to complete. Includes search operations that do not find a matching PTE and search operations caused by **dst***x* instructions. Note that MMCR2[0] determines whether the value in the MMCR0 threshold field is multiplied by 2 or 32.
22 (001_0110)	L1 data cache load hits	Counts each L1 data cache load hit; misaligned load accesses are counted separately. Does not include MMU table search lookups.
23 (001_0111)	L1 data store hit	Counts write-back store hit attempts that successfully hit the L1 data cache. Does not count if store hit on shared cache block.
24 (001_1000)	L1 data hits	Counts L1 data load, store, and touch hits.

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
25 (001_1001)	L2 tag lookup	Counts once per each L2 tag lookup. Does not include snoop look-ups.
26 (001_1010)	L2 tag accesses	Counts L2 tag accesses, including L2 tag lookups, writes, snoop look-ups, and snoop writes
27 (001_1011)	BIU non-ARTRYed data $\overline{\text{TS}}$ is asserted	Counts read and write transactions performed on the system interface. Does not count address-only transaction types.
28 (001_1100)	BIU single-beat read cycles	Counts when $\overline{\text{TA}}$ generates single-beat reads
29 (001_1101)	LFQ entries	Load fold queue entries. Counts loads that miss in the L1 data cache but hit on an already outstanding reload to that same cache block in the data reload table.
30 (001_1110)	L1 data write-hit-on-shared	Counts write-back stores that hit in a shared cache block in the L1 data cache and begin to claim the cache block as exclusive.
31 (001_1111)	Write through stores	Counts write-through stores written to bus. If store gathering is occurring, counts stores after gathering.
32 (010_0000)	L1 instruction cache miss fetches	Counts L1 instruction cache miss fetches (both cacheable and cache-inhibited)
33 (010_0001)	L2 data-side hits	Counts L1 data cacheable load/store misses that hit in the L2 cache. Does not include hits due to L1 data cache castouts, cache operations, or write-through stores.
34 (010_0010)	L2 instruction-side misses	Counts cacheable instruction misses in the L2 (L2 reloads due to instruction reload table)
35 (010_0011)	L1 data castout hits to L2	Counts L1 data castouts to L2 that hit in on a valid sector in the L2 cache
36 (010_0100)	L2 allocations	Counts when a new valid tag is allocated in the L2. This is sector-independent.
37 (010_0101)	Speculative stalled BIU cycles	Counts the number of bus cycles in the BIU that cannot request the address bus because it is waiting for a guarded load to become nonspeculative
38 (010_0110)	**dst** instructions dispatched	Counts **dst** instructions dispatched to the vector touch queue engine. Includes speculative **dst** instructions that are cancelled.
39 (010_0111)	**dst** stream 0 cache line fetches	Counts **dst** stream 0 cache line fetches from the data stream engine (VT0) within the vector-touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache
40 (010_1000)	Refreshed **dst** instructions	Counts each new **dst** instruction issued to already active stream, overwriting the old stream
41 (010_1001)	**dst** suspensions due to change of context	Counts **dst** instructions suspended due to change of context; that is, the times any number of streams pause due to change in MSR[PR] or MSR[DR]
42 (010_1010)	Raw snoop request	External snoop requests
43 (010_1011)	WOOP push address tenures	Window of opportunity push address tenures. Doesn’t count if $\overline{\text{ARTRY}}$ is asserted for WR/KILL.
44 (010_1100)	L2CO snoop hits	Counts snoop requests that hit in the L2 cast-out buffer

Table 11-9. PMC1 Events—MMCR0[PMC1SEL] Select Encodings (continued)

Number	Event	Description
45 (010_1101)	Hit-style intervention data-only transfers	Counts data-only transfers (data intervention) that can occur when MPX bus mode is enabled. Includes all three types (exclusive, modified, and shared (R)) of cache-to-cache, data-intervention transfers.
46 (010_1110)	LFQ touch entries	Counts load-fold queue entries that fold into a touch-initiated L1 data cache miss (data reload table entry). Occurs when load hits on an outstanding cache line prefetch initiated by a **dst***x*, **dcbt**, or **dcbtst** instruction. This event indicates that the touch prefetch is in progress when the demand load occurred but the prefetch is not sufficiently ahead of the load for the load to achieve a L1 data cache hit.
47 (010_1111)	L1 operations queue snoop hits	Counts snoop requests that hit in the L1 operations queue (contains L1 data cache castouts)
48 (011_0000)	AltiVec loads completed	Counts AltiVec load instructions completed (0, 1, or 2 instructions per cycle)
95-127	—	Reserved

11.5.2 PMC2 Events

MMCR0[PMC2SEL] specify the events associated with PMC2, as shown in Table 11-10.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings

Number	Event	Description
0 (00_0000)	Nothing	Register counter holds current value.
1 (00_0001)	Processor cycles	Counts every processor cycle.
2 (000_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Does not include folded branches. 0, 1, or 2 instructions per cycle.
3 (00_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL] 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (00_0100)	Instructions dispatched	Counts dispatched instructions (0, 1, or 2 instructions per cycle)
5 (00_0101)	Fall-through branches	Counts the branches that were resolved as a not-taken branch.
6 (00_0110)	ITLB misses	Counts the number of times that a requested address translation was not in the ITLB
7 (00_0111)	Instructions completed in VIU1	Counts the instructions completed by the ALU's simple integer subunit. The counter can increment by 0, 1, 2, or 3 depending on the number of completed instructions per cycle.
8 (00_1000)	Instructions completed in AltiVec ALU vector complex integer sub-unit	Counts the instructions completed by the AltiVec ALU complex integer subunit. The counter can increment by 0, 1, 2, or 3 depending on the number of completed instructions per cycle.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings (continued)

Number	Event	Description
9 (00_1001)	User/supervisor mode switches	Counts the number of times MSR[PR] toggles
10 (00_1010)	Reserved loads	Counts completed reserved load instructions
11 (00_1011)	Loads	Counts completed load instructions (0, 1, or 2 instructions per cycle)
12 (00_1100)	Snoops serviced	Counts snoops serviced to the L1 and the L2
13 (00_1101)	Dirty L1 castouts to L2	Incremented whenever the L1 casts out dirty data to the L2. A dirty castout is a cache block whose D bit is 1.
14 (00_1110)	SRU instructions completed	Counts completed instructions executed by the system register unit (SRU)
15 (00_1111)	L1 data load miss	Counts speculative load attempts that missed in the L1 data cache and were queued either in the data reload table or load fold queue. Does not include MMU.
16 (01_0000)	L1 data store misses	Counts write-back store attempts that missed in the L1 data cache and queued up a new entry in the data reload table or store-miss-merged to a current data reload table entry
17 (01_0001)	L1 data misses	Counts L1 data cache load, store, and touch misses
18 (01_0010)	L2 tag writes	Counts whenever an L2 tag needed to be written to a new state. Does not include writes due to snoop state updates.
19 (01_0011)	L2SRAM read cycles	Incremented for each beat of a read from the L2 SRAMs. A beat consists of 8 bytes for 64-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b00). For 32-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b10), each beat consists of 4 bytes. Note that for designs that use pipeline burst SRAMs (PB2) event 19 does not count the Rdrv cycle as shown in Figure 3-36.
20 (01_0100)	Generates BIU ARTRYed $\overline{\text{TS}}$	Counts transactions initiated by this processor and retried on the system interface.
21 (01_0101)	BIU multi-beat read cycles	Counts when $\overline{\text{TA}}$ is generated for multiple beat reads
22 (01_0110)	Data reload table store-miss merges	Counts data reload table store-miss merges
23 (01_0111)	L2 write-hit-on-SHD occurrence	Counts store misses from the data reload table that hit in a shared cache line in the L2 tag
24 (01_1000)	Cache-inhibited stores	Counts cache-inhibited stores written to bus. If store gathering is occurring, this counts stores after gathering
25 (01_1001)	L1 instruction cache reloads	Counts 32-byte cache blocks loaded into the L1 instruction cache
26 (01_1010)	L2 data side misses	Counts the number of times a load or store miss from the data reload table missed in the L2 tag. Does not include hits due to L1 data cache castouts, cache operations, or write-through stores.
27 (01_1011)	L2 instruction cache side hits	L1 instruction cacheable misses that hit in the L2 cache.

Table 11-10. PMC2 Events—MMCR0[PMC2SEL] Select Encodings (continued)

Number	Event	Description
28 (01_1100)	L1 data castouts to L2 misses	L1 data cache castouts to L2 that miss in the L2 cache.
29 (01_1101)	L2 sectors castout	Counts 0, 1, 2, 3, or 4 for a given L2 sector cast out.
30 (01_1110)	**dst** L1 cache line misses	Counts **dst** and **dstst** instructions that miss in the L1 data cache and queue in the data reload table. This event Counts the number of times the **dst** cache line fetch precedes the demand load or store by enough cycles to access the L1 data cache first.
31 (01_1111)	**dst** stream 1 cache line fetches	Counts **dst** stream 1 cache line fetches from the data stream engine (VT1) within the vector touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache.
32 (10_0000)	**dst** cache line fetches	**dst** cache line fetches—Includes all streams
33 (10_0001)	**dst** stream prematurely cancelled	Incremented when a stream is cancelled due to branch speculation cancel, inappropriate translation protection, or WIMG. Includes whenever a **dst** stream is cancelled for any reason other than reaching the end of the stream, refresh, or **dss**. Can count 0, 1, 2, 3, or 4 at a time.
34 (10_0010)	VFPU traps	Counts VFPU traps. These conditions are generated from certain operations when in Java mode.
35 (10_0011)	dRLT snoop hits	Counts serviced snoop requests that also hit on a data reload table entry
36 (10_0100)	Hit-style modified intervention data-only transfers	Increments when modified data is provided to the system interface using data intervention (MPX bus mode is enabled) due to a snoop
37 (10_0101)	Snoop hits	Counts serviced snoop requests that hit in one or more caches or buffers
38 (10_0110)	First speculative branch predicted correctly	Counts correctly predicted branches in the first speculative stream
39 (10_0111)	**dst** resumptions due to change of context	Counts the number of times any **dst** streams resume due to a change in MSR[PR] or MSR[DR]
40 (10_1000)	TLBI instructions	Counts completed **tlbi** instructions
41 (10_1001)	Snooped TLB invalidations	Counts TLB invalidations performed due to another master's TLBIE broadcast
42 (10_1010)	Generate BIU $\overline{\text{TA}}$ cycles	Counts when $\overline{\text{TA}}$ generates (single-beat or multi-beat, read or write). Indicates system interface bandwidth consumed when compared to the number of elapsed bus cycles.

11.5.3 PMC3 Events

Bits MMCR1[PMC3SEL] specify events associated with PMC3, as shown in Table 11-11.

Table 11-11. PMC3 Events—MMCR1[PMC3SEL] Select Encodings

Number	Event	Description
0 (0_0000)	Nothing	Register counter holds current value.
1 (0_0001)	Processor cycles	Counts every processor clock cycle.
2 (0_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Does not include folded branches. 0, 1, or 2 instructions per cycle.
3 (0_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL] 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (0_0100)	Instructions dispatched	Counts dispatched instructions (0, 1, or 2 instructions per cycle)
5 (0_0101)	Branches taken	Counts branches resolved as taken
6 (0_0110)	DTLB misses	Counts DTLB misses. Indicates the times that a requested data address translation was not in the DTLB.
7 (0_0111)	VCIU instructions completed	Counts completed VCIU instructions (0, 1, or 2 instructions per cycle)
8 (0_1000)	VFPU wait	Counts the number of cycles the vector FPU had a valid dispatch, but not valid operands
9 (0_1001)	—	Reserved
10 (0_1010)	**tlbsync** instructions	Counts completed **tlbsync** instructions.
11 (0_1011)	FPU instructions	Counts completed FPU instructions (0, 1, or 2 instructions per cycle)
12 (0_1100)	Conditional stores	Counts completed conditional store instructions.
13 (0_1101)	L2 snoop interventions	Counts snoop interventions from L2; that is, times when a 32-byte cache block is supplied to the system interface from the L2SRAMs.
14 (0_1110)	Second speculative branch predicted correctly	Counts correctly predicted branches in the second speculative stream.
15 (0_1111)	Stall on LR/CTR dependency.	Counts the number of cycles the BPU stalls due to the LR or CTR being unresolved.
16 (1_0000)	L1 data touch hit	Counts once per **dcbt** or **dcbtst** instruction or **dst***x* cache line fetch that hits in the L1 data cache.
17 (1_0001)	Miscellaneous L1 data cache operation cycles	Counts the number of cycles spent on **dcbf**, **dcbst**, **dcbi**, **sync**, **eieio**, **icbi**, **tlbi**, **tlbsync**, write-hit-shared, MMU loads, retried operations, etc., that are not counted elsewhere. These are the miscellaneous L1 data cache overhead operations.
18 (1_0010)	L1 data cycles used	Counts the number of cycles when the L1 data cache is used for any reason. Indicates L1 data cache bandwidth consumed when compared to the number of processor cycles elapsed.

Table 11-11. PMC3 Events—MMCR1[PMC3SEL] Select Encodings (continued)

Number	Event	Description
19 (1_0011)	L2 tag snoop lookup	Counts once per snoop query of the L2 tag
20 (1_0100)	L2SRAM write cycles	Counts once for each beat of a write to the L2SRAM. A beat is 8 bytes if L2CR[L2DSIZ] = 0, otherwise it is 16 bytes.
21 (1_0101)	dRLT SMM to 32 bytes	Counts when the store-miss merging mechanism merges misses to a full 32-byte cache block. If merged before the RWITM bus transaction obtains a bus grant, the RWITM converts to an address-only KILL transaction.
22 (1_0110)	**dst** cache line fetch dRLT hit	Counts **dst***x* cache line fetches that miss in the L1 data cache and hit in the data reload table. This event indicates another load, store, or touch miss to the same cache line is already in the data reload table. When this occurs the **dst** cache line fetch has occurred slightly too late to prefetch the cache line before the demand load or store is attempted at the L1 data cache.
23 (1_0111)	**dst** stream 2 cache line fetches	Counts **dst** stream 2 cache line fetches from the data stream engine (VT2) within the vector-touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache.
24 (1_1000)	**dss** instructions completed	Counts single-stream **dss** instructions (does not count **dssall** instructions). 0, 1, or 2 instructions per cycle.
25 (1_1001)	—	Reserved
26 (1_1010)	Snoop busies	Counts when the processor could not service a snoop request and asserts $\overline{\text{ARTRY}}$ as a snoop response.
27 (1_1011)	L2 snoop hits	Counts snoop requests that hit in the L2 tag.
28 (1_1100)	Hit-style exclusive intervention data-only transfers	Counts the number of times exclusive data is provided to the system interface using data intervention (MPX bus mode enabled) due to a snoop.
29 (1_1101)	BIU single-beat write cycles	Counts when $\overline{\text{TA}}$ is generated for single-beat writes
30 (1_1110)	L1 data reloads	Counts the number of times that the L1 data cache is reloaded with a new cache block from the data reload table.
All others	—	Reserved

11.5.4 PMC4 Events

Bits MMCR1PMC4SEL] specify events associated with PMC4, as shown in Table 11-12.

Table 11-12. PMC4 Events—MMCR1[PMC4SEL] Select Encodings

Number	Event	Description
0 (0_0000)	Nothing	Register counter holds current value.
1 (0_0001)	Processor cycles	Counts every processor cycle.
2 (0_0010)	Instructions completed	Counts all completed PowerPC and AltiVec instructions. Does not include folded branches. 0, 1, or 2 instructions per cycle.

Table 11-12. PMC4 Events—MMCR1[PMC4SEL] Select Encodings (continued)

Number	Event	Description
3 (0_0011)	TBL bit transitions	Counts transitions from 0 to 1 of TBL bits specified through MMCR0[TBSEL] 00 = uses the TBL[31] bit to count 01 = uses the TBL[23] bit to count 10 = uses the TBL[19] bit to count 11 = uses the TBL[15] bit to count
4 (0_0100)	Instructions dispatched	Counts dispatched instructions. 0, 1, or 2 instructions per cycle.
5 (0_0101)	Mispredicted branches	Counts the number of mispredicted branches
6 (0_0110)	DTLB table search operation cycles	Counts the number of cycles spent performing table search operations for the DTLB. Does not include cycles performing **dst***x*-initiated table search operations.
9 (0_1001)	—	Reserved
10 (0_1010)	Conditional stores successful	Counts completed store conditionals with reservation intact
11 (0_1011)	sync instructions	Counts completed **sync** instructions
12 (0_1100)	BIU KILLs	Counts successful (that is, non-ARTRYed) KILL transactions.
13 (0_1101)	Integer operations	Counts completed integer operations. 0, 1, or 2 instructions per cycle.
14 (0_1110)	Speculative branches fetch stall	Counts the number of cycles the branch unit cannot process new branches due to having two unresolved branches.
15 (0_1111)	L1 data touch miss	Counts once for every **dst***x* cache line fetch, **dcbt**, or **dcbtst** L1 data cache miss that causes a L1 data cache reload
16 (1_0000)	L1 data snoop intervention cycles	Counts the number of cycles that the L1 data cache is occupied reading data from the data cache for a snoop intervention. Two cycles per intervention.
17 (1_0001)	L2 tag snoop write	Counts L2 tag writes due to snoop state updates.
18 (1_0010)	L2SRAM cycles used	Counts L2SRAM read cycles, L2SRAM write cycles, and turnaround dead cycles required between reads and writes. Indicates L2SRAM bandwidth consumed when compared to the number of L2 clock cycles elapsed. Note that for designs that use pipeline burst SRAMs (PB2) event 18 also counts the Rdrv cycle as shown in Figure 3-36.
19 (1_0011)	L1 data castouts	Counts castouts from the L1 data cache to the L2.
20 (1_0100)	**dst** cache line fetch L1 data hits	Counts **dst***x* cache line fetches that hit in the L1 data cache. This event indicates that another load, store, or touch already brought the cache line into the L1 data cache. This occurs when the **dst** cache line fetch occurs too late to prefetch the cache line and wasted a cycle of L1 data cache bandwidth.
21 (1_0101)	**dst** stream 3 cache line fetches	Counts **dst** stream 3 cache line fetches from the data stream engine (VT3) within the vector-touch queue (VTQ). This includes accesses that hit or miss in the L1 data cache.
22 (1_0110)	**dssall** instructions	Counts completed **dssall** instructions
23 (1_0111)	L1 data snoop hits	Counts snoop requests that hit in the L1 data cache

Table 11-12. PMC4 Events—MMCR1[PMC4SEL] Select Encodings (continued)

Number	Event	Description
24 (1_1000)	Hit-style shared intervention data-only transfers	Counts the number of times shared (R) data is provided to the system interface using data intervention (MPX bus mode is enabled) due to a snoop.
25 (1_1001)	BIU multi-beat write cycles	Counts when $\overline{TA}$ is asserted for multiple beat writes
26 (1_1010)	L1 data snoop interventions	Counts the number of times a 32-byte cache line of data is supplied to the system interface from the L1 data cache for a snoop intervention.
27 (1_1011)	**dst***x*-initiated successful table search operations	Incremented whenever a **dst** instruction initiates a data address translation table search that results in a successful match in the page table.
28 (1_1100)	Hit-style exclusive intervention data-only transfers	Counts the number of times exclusive data is provided to the system interface using data intervention (MPX bus mode enabled) due to a snoop.
29 (1_1101)	BIU single-beat write transactions	Counts when $\overline{TA}$ is generated for single-beat writes
30 (1_1110)	L1 data reloads	Counts the number of times that the L1 data cache is reloaded with a new cache block from the data reload table.
31 (1_1111)	—	Reserved

Appendix A
MPC7410 Instruction Set Listings

This appendix lists the MPC7410 microprocessor’s instruction set as well as the additional PowerPC instructions not implemented in the MPC7410. Instructions are sorted by mnemonic, opcode, function, and form. Also included in this appendix is a quick reference table that contains general information, such as the architecture level, privilege level, and form, and indicates if the instruction is 64-bit and optional. Note that the MPC7410 is a 32-bit microprocessor, and does not implement any 64-bit instructions.

Note that split fields, that represent the concatenation of sequences from left to right, are shown in lowercase. For more information refer to Chapter 8, “Instruction Set,” in *The Programming Environments Manual*.

A.1 Instructions Sorted by Mnemonic (Decimal and Hexidecimal)

Table A-1 shows the instructions implemented in the MPC7410 in alphabetical order by their mnemonic name. The primary opcode (0–5) and secondary opcode (21-31) are decimal and hexidecimal values.

Key:

▨ Reserved bits

Table A-1. Instructions by Mnemonic (Dec, Hex)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
add*x*	31 (0x1F)	D	A	B	OE	0266 (0x10A)	Rc
addc*x*	31 (0x1F)	D	A	B	OE	0010 (0x00A)	Rc
adde*x*	31 (0x1F)	D	A	B	OE	0138 (0x08A)	Rc
addi	14 (0xE)	D	A	SIMM			
addic	12 (0xC)	D	A	SIMM			
addic.	13 (0xD)	D	A	SIMM			
addis	15 (0xF)	D	A	SIMM			
addme*x*	31 (0x1F)	D	A	0_0000	OE	0234 (0x0EA)	Rc
addze*x*	31 (0x1F)	D	A	0_0000	OE	0202 (0x0CA)	Rc
and*x*	31 (0x1F)	S	A	B	0028 (0x01C)		Rc
andc*x*	31 (0x1F)	S	A	B	0060 (0x03C)		Rc
andi.	28 (0x1C)	S	A	UIMM			
andis.	29 (0x1D)	S	A	UIMM			

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 … 5	6	7 8	9	10	11 … 15	16 … 20	21	22 … 29	30	31
b*x*	18 (0x12)	LI								AA	LK
bc*x*	16 (0x10)	BO				BI	BD			AA	LK
bcctr*x*	19 (0x13)	BO				BI	0_0000	00528 (0x210)			LK
bclr*x*	19 (0x13)	BO				BI	0_0000	0016 (0x010)			LK
cmp	31 (0x1F)	crfD		0	L	A	B	0000 (0x000)			0
cmpi	11 (0x0B)	crfD		0	L	A	SIMM				
cmpl	31 (0x1F)	crfD		0	L	A	B	0032 (0x020)			0
cmpli	10 (0x0A)	crfD		0	L	A	UIMM				
cntlzw*x*	31 (0x1F)	S				A	0_0000	0026 (0x01A)			Rc
crand	19 (0x13)	crbD				crbA	crbB	0257 (0x101)			0
crandc	19 (0x13)	crbD				crbA	crbB	0129 (0x081)			0
creqv	19 (0x13)	crbD				crbA	crbB	0289 (0x121)			0
crnand	19 (0x13)	crbD				crbA	crbB	0225 (0x0E1)			0
crnor	19 (0x13)	crbD				crbA	crbB	0033 (0x21)			0
cror	19 (0x13)	crbD				crbA	crbB	0449 (0x1C1)			0
crorc	19 (0x13)	crbD				crbA	crbB	0417 (0x1A1)			0
crxor	19 (0x13)	crbD				crbA	crbB	0193 (0C1)			0
dcba[1]	31 (0x1F)	000_00				A	B	0758 (0x2F6)			0
dcbf	31 (0x1F)	000_00				A	B	0086 (0x056)			0
dcbi[2]	31 (0x1F)	000_00				A	B	0470 (0x1D6)			0
dcbst	31 (0x1F)	000_00				A	B	0054 (0x036)			0
dcbt	31 (0x1F)	000_00				A	B	0278 (0x116)			0
dcbtst	31 (0x1F)	000_00				A	B	0246 (0x0F6)			0
dcbz	31 (0x1F)	000_00				A	B	1014 (0x3F6)			0
divw*x*	31 (0x1F)	D				A	B	OE	0491 (0x1EB)		Rc
divwu*x*	31 (0x1F)	D				A	B	OE	0459 (0x1CB)		Rc
dss[3]	31 (0x1F)	0	00	STRM		00_000	0_0000	0822 (0x336)			0
dssall[3]	31 (0x1F)	1	00	STRM		00_000	0_0000	0822 (0x336)			0
dst[3]	31 (0x1F)	0	00	STRM		A	B	0342 (0x156)			0
dstst[3]	31 (0x1F)	0	00	STRM		A	B	0374 (0x09C)			0
dststt[3]	31 (0x1F)	1	00	STRM		A	B	0374 (0x176)			0
dstt[3]	31 (0x1F)	1	00	STRM		A	B	0342 (0x0B0)			0
eciwx[1]	31 (0x1F)	D				A	B	0310 (0x136)			0

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
ecowx[1]	31 (0x1F)	S		A	B	0438 (0x1B6)		0
eieio	31 (0x1F)	000_00		00_000	0_0000	0854 (0x356)		0
eqv_x_	31 (0x1F)	S		A	B	0284 (0x11C)		Rc
extsb_x_	31 (0x1F)	S		A	0_0000	0954 (0x3BA)		Rc
extsh_x_	31 (0x1F)	S		A	0_0000	0922 (0x39A)		Rc
fabs_x_	63 (0x3F)	D		00_000	B	0264 (0x108)		Rc
fadd_x_	63 (0x3F)	D		A	B	0000_0	0021 (0x015)	Rc
fadds_x_	59 (0x3B)	D		A	B	0000_0	0021 (0x015)	Rc
fcmpo	63 (0x3F)	crfD	00	A	B	0032 (0x020)		0
fcmpu	63 (0x3F)	crfD	00	A	B	0000 (0x000)		0
fctiw_x_	63 (0x3F)	D		00_000	B	0014 (0x00E)		Rc
fctiwz_x_	63 (0x3F)	D		00_000	B	0015 (0x00F)		Rc
fdiv_x_	63 (0x3F)	D		A	B	0000_0	0018 (0x012)	Rc
fdivs_x_	59 (0x3B)	D		A	B	0000_0	0018 (0x012)	Rc
fmadd_x_	63 (0x3F)	D		A	B	C	0029 (0x01D)	Rc
fmadds_x_	59 (0x3B)	D		A	B	C	0029 (0x01D)	Rc
fmr_x_	63 (0x3F)	D		00_000	B	0072 (0x48)		Rc
fmsub_x_	63 (0x3F)	D		A	B	C	0028 (0x01C)	Rc
fmsubs_x_	59 (0x3B)	D		A	B	C	0028 (0x01C)	Rc
fmul_x_	63 (0x3F)	D		A	0_0000	C	0025 (0x019)	Rc
fmuls_x_	59 (0x3B)	D		A	0_0000	C	0025 (0x019)	Rc
fnabs_x_	63 (0x3F)	D		00_000	B	0136 (0x88)		Rc
fneg_x_	63 (0x3F)	D		00_000	B	0040 (0x28)		Rc
fnmadd_x_	63 (0x3F)	D		A	B	C	0031 (0x01F)	Rc
fnmadds_x_	59 (0x3B)	D		A	B	C	0031 (0x01F)	Rc
fnmsub_x_	63 (0x3F)	D		A	B	C	0030 (0x01E)	Rc
fnmsubs_x_	59 (0x3B)	D		A	B	C	0030 (0x01E)	Rc
fres_x_ [1]	59 (0x3B)	D		00_000	B	0000_0	0024 (0x018)	Rc
frsp_x_	63 (0x3F)	D		00_000	B	0012 (0xC)		Rc
frsqrte_x_ [1]	63 (0x3F)	D		00_000	B	0000_0	0026 (0x01A)	Rc
fsel_x_ [1]	63 (0x3F)	D		A	B	C	0023 (0x017)	Rc
fsqrt_x_ [4]	63 (0x3F)	D		00_000	B	0000_0	0022 (0x016)	Rc
fsqrts_x_ [4]	59 (0x3B)	D		00_000	B	0000_0	0022 (0x016)	Rc

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fsub*x*	63 (0x3F)	D	A	B	0000_0	0020 (0x014)	Rc
fsubs*x*	59 (0x3B)	D	A	B	0000_0	0020 (0x014)	Rc
icbi	31 (0x1F)	000_00	A	B	0982 (0x3D6)		0
isync	19 (0x13)	000_00	00_000	0_0000	0150 (0x096)		0
lbz	34 (0x22)	D	A	d			
lbzu	35 (0x23)	D	A	d			
lbzux	31 (0x1F)	D	A	B	0119 (0x077)		0
lbzx	31 (0x1F)	D	A	B	087 (0x057)		0
lfd	50 (0x32)	D	A	d			
lfdu	51 (0x33)	D	A	d			
lfdux	31 (0x1F)	D	A	B	0631 (0x277)		0
lfdx	31 (0x1F)	D	A	B	0599 (0x257)		0
lfs	48 (0x30)	D	A	d			
lfsu	49 (0x31)	D	A	d			
lfsux	31 (0x1F)	D	A	B	0567 (0x237)		0
lfsx	31 (0x1F)	D	A	B	0535 (0x217)		0
lha	42 (0x2A)	D	A	d			
lhau	43 (0x2B)	D	A	d			
lhaux	31 (0x1F)	D	A	B	0375 (0x177)		0
lhax	31 (0x1F)	D	A	B	0343 (0x157)		0
lhbrx	31 (0x1F)	D	A	B	0790 (0x316)		0
lhz	40 (0x28)	D	A	d			
lhzu	41 (0x29)	D	A	d			
lhzux	31 (0x1F)	D	A	B	0311 (0x137)		0
lhzx	31 (0x1F)	D	A	B	0279 (0x117)		0
lmw[5]	46 (0x2E)	D	A	d			
lswi[5]	31 (0x1F)	D	A	NB	0597 (0x255)		0
lswx[5]	31 (0x1F)	D	A	B	0533 (0x215)		0
lvebx[3]	31 (0x1F)	**v**D	A	B	0007 (0x007)		0
lvehx[3]	31 (0x1F)	**v**D	A	B	0039 (0x027)		0
lvewx[3]	31 (0x1F)	**v**D	A	B	0071 (0x047)		0
lvsl[3]	31 (0x1F)	**v**D	A	B	0006 (0x006)		0
lvsr[3]	31 (0x1F)	**v**D	A	B	0038 (0x026)		0

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

<table>
<tr><th>Name</th><th>0 5</th><th>6</th><th>7 8</th><th>9 10</th><th>11</th><th>12 13</th><th>14</th><th>15</th><th>16 17 18 19</th><th>20</th><th>21</th><th>22 23 24 25 26 27 28 29 30</th><th>31</th></tr>
<tr><td>lvx[3]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="4">A</td><td colspan="2">B</td><td colspan="2">0103 (0x067)</td><td>0</td></tr>
<tr><td>lvxl[3]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="4">A</td><td colspan="2">B</td><td colspan="2">0359 (0x167)</td><td>0</td></tr>
<tr><td>lwarx</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="2">B</td><td colspan="2">0020 (0x014)</td><td>0</td></tr>
<tr><td>lwbrx</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="2">B</td><td colspan="2">0534 (0x216)</td><td>0</td></tr>
<tr><td>lwz</td><td>32 (0x20)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="5">d</td></tr>
<tr><td>lwzu</td><td>33 (0x21)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="5">d</td></tr>
<tr><td>lwzux</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="2">B</td><td colspan="2">0055 (0x037)</td><td>0</td></tr>
<tr><td>lwzx</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="2">B</td><td colspan="2">0023 (0x017)</td><td>0</td></tr>
<tr><td>mcrf</td><td>19 (0x13)</td><td colspan="2">crfD</td><td>00</td><td colspan="2">crfS</td><td colspan="2">00</td><td colspan="2">0_0000</td><td colspan="2">0000 (0x000)</td><td>0</td></tr>
<tr><td>mcrfs</td><td>63 (0x3F)</td><td colspan="2">crfD</td><td>00</td><td colspan="2">crfS</td><td colspan="2">00</td><td colspan="2">0_0000</td><td colspan="2">0064 (0x040)</td><td>0</td></tr>
<tr><td>mcrxr</td><td>31 (0x1F)</td><td colspan="2">crfD</td><td>00</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">0512 (0x200)</td><td>0</td></tr>
<tr><td>mfcr</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">0019 (0x013)</td><td>0</td></tr>
<tr><td>mffsx</td><td>63 (0x3F)</td><td colspan="3">D</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">0583 (0x247)</td><td>Rc</td></tr>
<tr><td>mfmsr[2]</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">0083 (0x053)</td><td>0</td></tr>
<tr><td>mfspr[6]</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="6">spr</td><td colspan="2">0339 (0x153)</td><td>0</td></tr>
<tr><td>mfsr[2]</td><td>31 (0x1F)</td><td colspan="3">D</td><td>0</td><td colspan="3">SR</td><td colspan="2">0_0000</td><td colspan="2">0595 (0x099)</td><td>0</td></tr>
<tr><td>mfsrin[2]</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">00_000</td><td colspan="2">B</td><td colspan="2">0659 (0x293)</td><td>0</td></tr>
<tr><td>mftb</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="6">tbr</td><td colspan="2">0371 (0x173)</td><td>0</td></tr>
<tr><td>mfvscr[3]</td><td>04 (0x04)</td><td colspan="3">vD</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">1540 (0x604)</td><td>0</td></tr>
<tr><td>mtcrf</td><td>31 (0x1F)</td><td colspan="3">S</td><td>0</td><td colspan="4">CRM</td><td>0</td><td colspan="2">0144 (0x090)</td><td>0</td></tr>
<tr><td>mtfsb0x</td><td>63 (0x3F)</td><td colspan="3">crbD</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">0070 (0x046)</td><td>Rc</td></tr>
<tr><td>mtfsb1x</td><td>63 (0x3F)</td><td colspan="3">crbD</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">0038 (0x026)</td><td>Rc</td></tr>
<tr><td>mtfsfx</td><td>63 (0x3F)</td><td>0</td><td colspan="5">FM</td><td>0</td><td colspan="2">B</td><td colspan="2">0711 (0x2C7)</td><td>Rc</td></tr>
<tr><td>mtfsfix</td><td>63 (0x3F)</td><td colspan="2">crfD</td><td>00</td><td colspan="4">00_000</td><td>IMM</td><td>0</td><td colspan="2">0134 (0x086)</td><td>Rc</td></tr>
<tr><td>mtmsr[2]</td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="4">00_000</td><td colspan="2">0_0000</td><td colspan="2">0146 (0x092)</td><td>0</td></tr>
<tr><td>mtspr[6]</td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="6">spr</td><td colspan="2">0467 (0x1D3)</td><td>0</td></tr>
<tr><td>mtsr[2]</td><td>31 (0x1F)</td><td colspan="3">S</td><td>0</td><td colspan="3">SR</td><td colspan="2">0_0000</td><td colspan="2">0210 (0x001)</td><td>0</td></tr>
<tr><td>mtsrin[2]</td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="4">00_000</td><td colspan="2">B</td><td colspan="2">0242 (0x0F2)</td><td>0</td></tr>
<tr><td>mtvscr[3]</td><td>04 (0x04)</td><td colspan="3">000_00</td><td colspan="4">00_000</td><td colspan="2">vB</td><td colspan="2">1604 (0x644)</td><td>0</td></tr>
<tr><td>mulhwx</td><td>31(0x1F)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="2">B</td><td>0</td><td>0075 (0x04B)</td><td>Rc</td></tr>
<tr><td>mulhwux</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="2">B</td><td>0</td><td>0011 (0x00B)</td><td>Rc</td></tr>
<tr><td>mulli</td><td>07 (0x07)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="5">SIMM</td></tr>
<tr><td>mullwx</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="4">A</td><td colspan="2">B</td><td>OE</td><td>0235 (0x0EB)</td><td>Rc</td></tr>
</table>

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
nand*x*	31 (0x1F)	S	A	B	0476 (0x1DC)		Rc
neg*x*	31 (0x1F)	D	A	0_0000	OE	0104 (0x068)	Rc
nor*x*	31 (0x1F)	S	A	B	0124 (0x07C)		Rc
or*x*	31 (0x1F)	S	A	B	0444 (0x1BC)		Rc
orc*x*	31 (0x1F)	S	A	B	0412 (0x19C)		Rc
ori	24 (0x18)	S	A	UIMM			
oris	25 (0x19)	S	A	UIMM			
rfi[2]	19 (0x13)	000_00	00_000	0_0000	0050 (0x032)		0
rlwimi*x*	20 (0x14)	S	A	SH	MB	ME	Rc
rlwinm*x*	21 (0x15)	S	A	SH	MB	ME	Rc
rlwnm*x*	23 (0x17)	S	A	B	MB	ME	Rc
sc	17 (0x11)	000_0000_0000_0000_0000_0000_00				1	0
slw*x*	31 (0x1F)	S	A	B	0024 (0x018)		Rc
sraw*x*	31 (0x1F)	S	A	B	0792 (0x318)		Rc
srawi*x*	31 (0x1F)	S	A	SH	0824 (0x338)		Rc
srw*x*	31 (0x1F)	S	A	B	0536 (0x218)		Rc
stb	38 (0x26)	S	A	d			
stbu	39 (0x27)	S	A	d			
stbux	31 (0x1F)	S	A	B	0247 (0x0F7)		0
stbx	31 (0x1F)	S	A	B	0215 (0x0D7)		0
stfd	54 (0x36)	S	A	d			
stfdu	55 (0x37)	S	A	d			
stfdux	31 (0x1F)	S	A	B	0759 (0x2F7)		0
stfdx	31 (0x1F)	S	A	B	0727 (0x2D7)		0
stfiwx[1]	31 (0x1F)	S	A	B	0983 (0x3D7)		0
stfs	52 (0x34)	S	A	d			
stfsu	53 (0x35)	S	A	d			
stfsux	31 (0x1F)	S	A	B	0695 (0x2B7)		0
stfsx	31 (0x1F)	S	A	B	0663 (0x297)		0
sth	44 (0x2C)	S	A	d			
sthbrx	31 (0x1F)	S	A	B	0918 (0x396)		0
sthu	45 (0x2D)	S	A	d			
sthux	31 (0x1F)	S	A	B	0439 (0x1B7)		0

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
sthx	31 (0x1F)	S	A	B	0407 (0x197)		0
stmw[5]	47 (0x2F)	S	A	d			
stswi[5]	31 (0x1F)	S	A	NB	0725 (0x2D5)		0
stswx[5]	31 (0x1F)	S	A	B	0661 (0x295)		0
stvebx[3]	31 (0x1F)	**v**S	A	B	0135 (0x127)		0
stvehx[3]	31 (0x1F)	**v**S	A	B	0167 (0x0A7)		0
stvewx[3]	31 (0x1F)	**v**S	A	B	0199 (0x0C7)		0
stvx[3]	31 (0x1F)	**v**S	A	B	0231 (0x01F)		0
stvxl[3]	31 (0x1F)	**v**S	A	B	0487 (0x1E7)		0
stw	36 (0x24)	S	A	d			
stwbrx	31 (0x1F)	S	A	B	0662 (0x296)		0
stwcx.	31 (0x1F)	S	A	B	0150 (0x096)		1
stwu	37 (0x25)	S	A	d			
stwux	31 (0x1F)	S	A	B	0183 (0x0B7)		0
stwx	31 (0x1F)	S	A	B	0151 (0x097)		0
subf*x*	31 (0x1F)	D	A	B	OE	0040 (0x028)	Rc
subfc*x*	31 (0x1F)	D	A	B	OE	0008 (0x008)	Rc
subfe*x*	31 (0x1F)	D	A	B	OE	0136 (0x088)	Rc
subfic	08 (0x08)	D	A	SIMM			
subfme*x*	31 (0x1F)	D	A	0_0000	OE	0232 (0x0E8)	Rc
subfze*x*	31 (0x1F)	D	A	0_0000	OE	0200 (0x0C8)	Rc
sync	31 (0x1F)	000_00	00_000	0_0000	0598 (0x256)		0
tlbia[4]	31 (0x1F)	000_00	00_000	0_0000	0370 (0x172)		0
tlbie[1, 2]	31 (0x1F)	000_00	00_000	B	0306 (0x132)		0
tlbsync[1, 2]	31 (0x1F)	000_00	00_000	0_0000	0566 (0x236)		0
tw	31 (0x1F)	TO	A	B	0004 (0x004)		0
twi	03 (0x03)	TO	A	SIMM			
vaddcuw[3]	04 (0x04)	**v**D	**v**A	**v**B	084 (0x180)		0
vaddfp[3]	04 (0x04)	**v**D	**v**A	**v**B	0010 (0x0B4)		0
vaddsbs[3]	04 (0x04)	**v**D	**v**A	**v**B	0768 (0x300)		0
vaddshs[3]	04 (0x04)	**v**D	**v**A	**v**B	0832 (0x340)		0
vaddsws[3]	04 (0x04)	**v**D	**v**A	**v**B	0896 (0x154)		0
vaddubm[3]	04 (0x04)	**v**D	**v**A	**v**B	0000 (0x000)		0

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30	31
vaddubs[3]	04 (0x04)	vD	vA	vB	0512 (0x200)			0
vadduhm[3]	04 (0x04)	vD	vA	vB	0064 (0x040)			0
vadduhs[3]	04 (0x04)	vD	vA	vB	0576 (0x240)			0
vadduwm[3]	04 (0x04)	vD	vA	vB	0128 (0x0F0)			0
vadduws[3]	04 (0x04)	vD	vA	vB	0640 (0x280)			0
vand[3]	04 (0x04)	vD	vA	vB	1028 (0x118)			0
vandc[3]	04 (0x04)	vD	vA	vB	1092 (0x444)			0
vavgsb[3]	04 (0x04)	vD	vA	vB	1282 (0x502)			0
vavgsh[3]	04 (0x04)	vD	vA	vB	1346 (0x542)			0
vavgsw[3]	04 (0x04)	vD	vA	vB	1410 (0x582)			0
vavgub[3]	04 (0x04)	vD	vA	vB	1026 (0x402)			0
vavguh[3]	04 (0x04)	vD	vA	vB	1090 (0x442)			0
vavguw[3]	04 (0x04)	vD	vA	vB	1154 (0x482)			0
vcfsx[3]	04 (0x04)	vD	UIMM	vB	0842 (0x1E2)			
vcfux[3]	04 (0x04)	vD	UIMM	vB	0778 (0x30A)			0
vcmpbfp*x*[3]	04 (0x04)	vD	vA	vB	Rc	0966 (0x3C6)		
vcmpeqfp*x*[3]	04 (0x04)	vD	vA	vB	Rc	0198 (0x0C6)		
vcmpequb*x*[3]	04 (0x04)	vD	vA	vB	Rc	0006 (0x006)		
vcmpequh*x*[3]	04 (0x04)	vD	vA	vB	Rc	0070 (0x046)		
vcmpequw*x*[3]	04 (0x04)	vD	vA	vB	Rc	0134 (0x086)		
vcmpgefp*x*[3]	04 (0x04)	vD	vA	vB	Rc	0454 (0x1C6)		
vcmpgtfp*x*[3]	04 (0x04)	vD	vA	vB	Rc	0710 (0x2C6)		
vcmpgtsb*x*[3]	04 (0x04)	vD	vA	vB	Rc	0774 (0x306)		
vcmpgtsh*x*[3]	04 (0x04)	vD	vA	vB	Rc	0838 (0x346)		
vcmpgtsw*x*[3]	04 (0x04)	vD	vA	vB	Rc	0902 (0x386)		
vcmpgtub*x*[3]	04 (0x04)	vD	vA	vB	Rc	0518 (0x206)		
vcmpgtuh*x*[3]	04 (0x04)	vD	vA	vB	Rc	0582 (0x246)		
vcmpgtuw*x*[3]	04 (0x04)	vD	vA	vB	Rc	0646 (0x286)		
vctsxs[3]	04 (0x04)	vD	UIMM	vB	0970 (0x3CA)			
vctuxs[3]	04 (0x04)	vD	UIMM	vB	0906 (0x38A)			
vexptefp[3]	04 (0x04)	vD	00_000	vB	0394 (0x18A)			
vlogefp[3]	04 (0x04)	vD	00_000	vB	0458 (0x1CA)			
vmaddfp[3]	04 (0x04)	vD	vA	vB	vC		0046 (0x002E)	

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30 31
vmaxfp[3]	04 (0x04)	vD	vA	vB	1034 (0x040A)	
vmaxsb[3]	04 (0x04)	vD	vA	vB	0258 (0x028)	
vmaxsh[3]	04 (0x04)	vD	vA	vB	0322 (0x01C)	
vmaxsw[3]	04 (0x04)	vD	vA	vB	0386 (0x182)	
vmaxub[3]	04 (0x04)	vD	vA	vB	0002 (0x002)	
vmaxuh[3]	04 (0x04)	vD	vA	vB	0066 (0x042)	
vmaxuw[3]	04 (0x04)	vD	vA	vB	0130 (0x082)	
vmhaddshs[3]	04 (0x04)	vD	vA	vB	vC	0032 (0x020)
vmhraddshs[3]	04 (0x04)	vD	vA	vB	vC	0033 (0x021)
vminfp[3]	04 (0x04)	vD	vA	vB	1098 (0x44A)	
vminsb[3]	04 (0x04)	vD	vA	vB	0770 (0x302)	
vminsh[3]	04 (0x04)	vD	vA	vB	0834 (0x342)	
vminsw[3]	04 (0x04)	vD	vA	vB	0898 (0x382)	
vminub[3]	04 (0x04)	vD	vA	vB	0514 (0x202)	
vminuh[3]	04 (0x04)	vD	vA	vB	0578 (0x242)	
vminuw[3]	04 (0x04)	vD	vA	vB	0642 (0x282)	
vmladduhm[3]	04 (0x04)	vD	vA	vB	vC	0034 (0x022)
vmrghb[3]	04 (0x04)	vD	vA	vB	0012 (0x00C)	
vmrghh[3]	04 (0x04)	vD	vA	vB	0076 (0x04C)	
vmrghw[3]	04 (0x04)	vD	vA	vB	0140 (0x08C)	
vmrglb[3]	04 (0x04)	vD	vA	vB	0268 (0x008)	
vmrglh[3]	04 (0x04)	vD	vA	vB	0332 (0x14C)	
vmrglw[3]	04 (0x04)	vD	vA	vB	0396 (0x18C)	
vmsummbm[3]	04 (0x04)	vD	vA	vB	vC	0037 (0x025)
vmsumshm[3]	04 (0x04)	vD	vA	vB	vC	0040 (0x028)
vmsumshs[3]	04 (0x04)	vD	vA	vB	vC	0041 (0x029)
vmsumubm[3]	04 (0x04)	vD	vA	vB	vC	0036 (0x024)
vmsumuhm[3]	04 (0x04)	vD	vA	vB	vC	0038 (0x026)
vmsumuhs[3]	04 (0x04)	vD	vA	vB	vC	0039 (0x027)
vmulesb[3]	04 (0x04)	vD	vA	vB	0776 (0x308)	
vmulesh[3]	04 (0x04)	vD	vA	vB	0840 (0x348)	
vmuleub[3]	04 (0x04)	vD	vA	vB	0520 (0x208)	
vmuleuh[3]	04 (0x04)	vD	vA	vB	0584 (0x248)	

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30 31
vmulosb[3]	04 (0x04)	vD	vA	vB	0264 (0x108)		
vmulosh[3]	04 (0x04)	vD	vA	vB	0328 (0x148)		
vmuloub[3]	04 (0x04)	vD	vA	vB	0008 (0x008)		
vmulouh[3]	04 (0x04)	vD	vA	vB	0072 (0x048)		
vnmsubfp[3]	04 (0x04)	vD	vA	vB	vC		0047 (0x02F)
vnor[3]	04 (0x04)	vD	vA	vB	1284 (0x504)		
vor[3]	04 (0x04)	vD	vA	vB	1156 (0x484)		
vperm[3]	04 (0x04)	vD	vA	vB	vC		0043 (0x02B)
vpkpx[3]	04 (0x04)	vD	vA	vB	0782 (0x30E)		
vpkshss[3]	04 (0x04)	vD	vA	vB	0398 (0x18E)		
vpkshus[3]	04 (0x04)	vD	vA	vB	0270 (0x012)		
vpkswss[3]	04 (0x04)	vD	vA	vB	0462 (0x00C)		
vpkswus[3]	04 (0x04)	vD	vA	vB	0334 (0x14E)		
vpkuhum[3]	04 (0x04)	vD	vA	vB	0014 (0x00E)		
vpkuhus[3]	04 (0x04)	vD	vA	vB	0142 (0x08E)		
vpkuwum[3]	04 (0x04)	vD	vA	vB	0078 (0x04E)		
vpkuwus[3]	04 (0x04)	vD	vA	vB	0206 (0x0CE)		
vrefp[3]	04 (0x04)	vD	00_000	vB	0266 (0x10A)		
vrfim[3]	04 (0x04)	vD	00_000	vB	0714 (0x2CA)		
vrfin[3]	04 (0x04)	vD	00_000	vB	0522 (0x20A)		
vrfip[3]	04 (0x04)	vD	00_000	vB	0650 (0x28A)		
vrfiz[3]	04 (0x04)	vD	00_000	vB	0586 (0x24A)		
vrlb[3]	04 (0x04)	vD	vA	vB	0004 (0x004)		
vrlh[3]	04 (0x04)	vD	vA	vB	0068 (0x044)		
vrlw[3]	04 (0x04)	vD	vA	vB	0132 (0x084)		
vrsqrtefp[3]	04 (0x04)	vD	00_000	vB	0330 (0x14A)		
vsel[3]	04 (0x04)	vD	vA	vB	vC		0042 (0x02A)
vsl[3]	04 (0x04)	vD	vA	vB	0452 (0x1C4)		
vslb[3]	04 (0x04)	vD	vA	vB	0260 (0x104)		
vsldoi[3]	04 (0x04)	vD	vA	vB	0	SH	0044 (0x02C)
vslh[3]	04 (0x04)	vD	vA	vB	0324 (0x144)		
vslo[3]	04 (0x04)	vD	vA	vB	1036 (0x40C)		
vslw[3]	04 (0x04)	vD	vA	vB	0388 (0x184)		

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
vspltb[3]	04 (0x04)	vD	UIMM	vB	0524 (0x20C)
vsplth[3]	04 (0x04)	vD	UIMM	vB	0588 (0x24C)
vspltisb[3]	04 (0x04)	vD	SIMM	0_0000	0780 (0x30C)
vspltish[3]	04 (0x04)	vD	SIMM	0_0000	0844 (0x34C)
vspltisw[3]	04 (0x04)	vD	SIMM	0_0000	0908 (0x38C)
vspltw[3]	04 (0x04)	vD	UIMM	vB	0652 (0x28C)
vsr[3]	04 (0x04)	vD	vA	vB	0708 (0x2C4)
vsrab[3]	04 (0x04)	vD	vA	vB	0772 (0x304)
vsrah[3]	04 (0x04)	vD	vA	vB	0836 (0x344)
vsraw[3]	04 (0x04)	vD	vA	vB	0900 (0x384)
vsrb[3]	04 (0x04)	vD	vA	vB	0516 (0x204)
vsrh[3]	04 (0x04)	vD	vA	vB	0580 (0x244)
vsro[3]	04 (0x04)	vD	vA	vB	1100 (0x44C)
vsrw[3]	04 (0x04)	vD	vA	vB	0644 (0x284)
vsubcuw[3]	04 (0x04)	vD	vA	vB	1408 (0x580)
vsubfp[3]	04 (0x04)	vD	vA	vB	0074 (0x4A)
vsubsbs[3]	04 (0x04)	vD	vA	vB	1792 (0x700)
vsubshs[3]	04 (0x04)	vD	vA	vB	1856 (0x740)
vsubsws[3]	04 (0x04)	vD	vA	vB	1920 (0x780)
vsububm[3]	04 (0x04)	vD	vA	vB	1024 (0x400)
vsububs[3]	04 (0x04)	vD	vA	vB	1536 (0x600)
vsubuhm[3]	04 (0x04)	vD	vA	vB	1088 (0x440)
vsubuhs[3]	04 (0x04)	vD	vA	vB	1600 (0x640)
vsubuwm[3]	04 (0x04)	vD	vA	vB	1152 (0x480)
vsubuws[3]	04 (0x04)	vD	vA	vB	1664 (0x680)
vsumsws[3]	04 (0x04)	vD	vA	vB	1928 (0x788)
vsum2sws[3]	04 (0x04)	vD	vA	vB	1672 (0x688)
vsum4sbs[3]	04 (0x04)	vD	vA	vB	1800 (0x708)
vsum4shs[3]	04 (0x04)	vD	vA	vB	1608 (0x648)
vsum4ubs[3]	04 (0x04)	vD	vA	vB	1544 (0x608)
vupkhpx[3]	04 (0x04)	vD	00_000	vB	0846 (0x34E)
vupkhsb[3]	04 (0x04)	vD	00_000	vB	0526 (0x20E)
vupkhsh[3]	04 (0x04)	vD	00_000	vB	0590 (0x24E)

Table A-1. Instructions by Mnemonic (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
vupklpx[3]	04 (0x04)	vD	00_000	vB	0974 (0x3CE)		
vupklsb[3]	04 (0x04)	vD	00_000	vB	0654 (0x28E)		
vupklsh[3]	04 (0x04)	vD	00_000	vB	0718 (0x2CE)		
vxor[3]	04 (0x04)	vD	vA	vB	1220 (0x4C4)		
xor*x*	31 (0x1F)	S	A	B	0316 (0x13C)		Rc
xori	26 (0x1A)	S	A	UIMM			
xoris	27 (0x1B)	S	A	UIMM			

[1] Optional to the architecture but implemented by the MPC7410
[2] Supervisor-level instruction
[3] AltiVec technology-specific instruction
[4] Optional instruction not implemented by the MPC7410
[5] Load/store string/multiple instruction
[6] Supervisor- and user-level instruction

A.2 Instructions Sorted by Primary and Secondary Opcodes (Decimal and Hexidecimal)

Table A-2 shows the instructions implemented in the MPC7410by their primary (0–5) and secondary (21-31) opcodes in decimal and hexidecimal format.

Key:

▨ Reserved bits

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
twi	03 (0x03)	TO	A	SIMM			
vaddubm[1]	04 (0x04)	vD	vA	vB	0000 (0x000)		0
vmaxub[1]	04 (0x04)	vD	vA	vB	0002 (0x002)		
vrlb[1]	04 (0x04)	vD	vA	vB	0004 (0x004)		
vcmpequb*x*[1]	04 (0x04)	vD	vA	vB	Rc	0006 (0x006)	
vmuloub[1]	04 (0x04)	vD	vA	vB	0008 (0x008)		
vaddfp[1]	04 (0x04)	vD	vA	vB	0010 (0x0B4)		0
vmrghb[1]	04 (0x04)	vD	vA	vB	0012 (0x00C)		
vpkuhum[1]	04 (0x04)	vD	vA	vB	0014 (0x00E)		
vmhaddshs[1]	04 (0x04)	vD	vA	vB	vC	0032 (0x020)	
vmhraddshs[1]	04 (0x04)	vD	vA	vB	vC	0033 (0x021)	
vmladduhm[1]	04 (0x04)	vD	vA	vB	vC	0034 (0x022)	
vmsumubm[1]	04 (0x04)	vD	vA	vB	vC	0036 (0x024)	

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30	31
vmsummbm[1]	04 (0x04)	vD	vA	vB	vC		0037 (0x025)	
vmsumuhm[1]	04 (0x04)	vD	vA	vB	vC		0038 (0x026)	
vmsumuhs[1]	04 (0x04)	vD	vA	vB	vC		0039 (0x027)	
vmsumshm[1]	04 (0x04)	vD	vA	vB	vC		0040 (0x028)	
vmsumshs[1]	04 (0x04)	vD	vA	vB	vC		0041 (0x029)	
vsel[1]	04 (0x04)	vD	vA	vB	vC		0042 (0x02A)	
vperm[1]	04 (0x04)	vD	vA	vB	vC		0043 (0x02B)	
vsldoi[1]	04 (0x04)	vD	vA	vB	0	SH	0044 (0x02C)	
vmaddfp[1]	04 (0x04)	vD	vA	vB	vC		0046 (0x002E)	
vnmsubfp[1]	04 (0x04)	vD	vA	vB	vC		0047 (0x02F)	
vadduhm[1]	04 (0x04)	vD	vA	vB	0064 (0x040)			0
vmaxuh[1]	04 (0x04)	vD	vA	vB	0066 (0x042)			
vrlh[1]	04 (0x04)	vD	vA	vB	0068 (0x044)			
vcmpequh*x*[1]	04 (0x04)	vD	vA	vB	Rc	0070 (0x046)		
vmulouh[1]	04 (0x04)	vD	vA	vB	0072 (0x048)			
vsubfp[1]	04 (0x04)	vD	vA	vB	0074 (0x4A)			
vmrghh[1]	04 (0x04)	vD	vA	vB	0076 (0x04C)			
vpkuwum[1]	04 (0x04)	vD	vA	vB	0078 (0x04E)			
vadduwm[1]	04 (0x04)	vD	vA	vB	0128 (0x0F0)			0
vmaxuw[1]	04 (0x04)	vD	vA	vB	0130 (0x082)			
vrlw[1]	04 (0x04)	vD	vA	vB	0132 (0x084)			
vcmpequw*x*[1]	04 (0x04)	vD	vA	vB	Rc	0134 (0x086)		
vmrghw[1]	04 (0x04)	vD	vA	vB	0140 (0x08C)			
vpkuhus[1]	04 (0x04)	vD	vA	vB	0142 (0x08E)			
vcmpeqfp*x*[1]	04 (0x04)	vD	vA	vB	Rc	0198 (0x0C6)		
vpkuwus[1]	04 (0x04)	vD	vA	vB	0206 (0x0CE)			
vmaxsb[1]	04 (0x04)	vD	vA	vB	0258 (0x028)			
vslb[1]	04 (0x04)	vD	vA	vB	0260 (0x104)			
vmulosb[1]	04 (0x04)	vD	vA	vB	0264 (0x108)			
vrefp[1]	04 (0x04)	vD	00_000	vB	0266 (0x10A)			
vmrglb[1]	04 (0x04)	vD	vA	vB	0268 (0x008)			
vpkshus[1]	04 (0x04)	vD	vA	vB	0270 (0x012)			
vmaxsh[1]	04 (0x04)	vD	vA	vB	0322 (0x01C)			

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
vslh[1]	04 (0x04)	**v**D	**v**A	**v**B	0324 (0x144)		
vmulosh[1]	04 (0x04)	**v**D	**v**A	**v**B	0328 (0x148)		
vrsqrtefp[1]	04 (0x04)	**v**D	00_000	**v**B	0330 (0x14A)		
vmrglh[1]	04 (0x04)	**v**D	**v**A	**v**B	0332 (0x14C)		
vpkswus[1]	04 (0x04)	**v**D	**v**A	**v**B	0334 (0x14E)		
vaddcuw[1]	04 (0x04)	**v**D	**v**A	**v**B	0384 (0x180)		0
vmaxsw[1]	04 (0x04)	**v**D	**v**A	**v**B	0386 (0x182)		
vslw[1]	04 (0x04)	**v**D	**v**A	**v**B	0388 (0x184)		
vexptefp[1]	04 (0x04)	**v**D	00_000	**v**B	0394 (0x18A)		
vmrglw[1]	04 (0x04)	**v**D	**v**A	**v**B	0396 (0x18C)		
vpkshss[1]	04 (0x04)	**v**D	**v**A	**v**B	0398 (0x18E)		
vsl[1]	04 (0x04)	**v**D	**v**A	**v**B	0452 (0x1C4)		
vcmpgefp*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0454 (0x1C6)	
vlogefp[1]	04 (0x04)	**v**D	00_000	**v**B	0458 (0x1CA)		
vpkswss[1]	04 (0x04)	**v**D	**v**A	**v**B	0462 (0x00C)		
vaddubs[1]	04 (0x04)	**v**D	**v**A	**v**B	0512 (0x200)		0
vminub[1]	04 (0x04)	**v**D	**v**A	**v**B	0514 (0x202)		
vsrb[1]	04 (0x04)	**v**D	**v**A	**v**B	0516 (0x204)		
vcmpgtub*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0518 (0x206)	
vmuleub[1]	04 (0x04)	**v**D	**v**A	**v**B	0520 (0x208)		
vrfin[1]	04 (0x04)	**v**D	00_000	**v**B	0522 (0x20A)		
vspltb[1]	04 (0x04)	**v**D	UIMM	**v**B	0524 (0x20C)		
vupkhsb[1]	04 (0x04)	**v**D	00_000	**v**B	0526 (0x20E)		
vadduhs[1]	04 (0x04)	**v**D	**v**A	**v**B	0576 (0x240)		0
vminuh[1]	04 (0x04)	**v**D	**v**A	**v**B	0578 (0x242)		
vsrh[1]	04 (0x04)	**v**D	**v**A	**v**B	0580 (0x244)		
vcmpgtuh*x*[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0582 (0x246)	
vmuleuh[1]	04 (0x04)	**v**D	**v**A	**v**B	0584 (0x248)		
vrfiz[1]	04 (0x04)	**v**D	00_000	**v**B	0586 (0x24A)		
vsplth[1]	04 (0x04)	**v**D	UIMM	**v**B	0588 (0x24C)		
vupkhsh[1]	04 (0x04)	**v**D	00_000	**v**B	0590 (0x24E)		
vadduws[1]	04 (0x04)	**v**D	**v**A	**v**B	0640 (0x280)		0
vminuw[1]	04 (0x04)	**v**D	**v**A	**v**B	0642 (0x282)		

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
vsrw[1]	04 (0x04)	**v**D	**v**A	**v**B	0644 (0x284)		
vcmpgtuw_x_[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0646 (0x286)	
vrfip[1]	04 (0x04)	**v**D	00_000	**v**B	0650 (0x28A)		
vspltw[1]	04 (0x04)	**v**D	UIMM	**v**B	0652 (0x28C)		
vupklsb[1]	04 (0x04)	**v**D	00_000	**v**B	0654 (0x28E)		
vsr[1]	04 (0x04)	**v**D	**v**A	**v**B	0708 (0x2C4)		
vcmpgtfp_x_[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0710 (0x2C6)	
vrfim[1]	04 (0x04)	**v**D	00_000	**v**B	0714 (0x2CA)		
vupklsh[1]	04 (0x04)	**v**D	00_000	**v**B	0718 (0x2CE)		
vaddsbs[1]	04 (0x04)	**v**D	**v**A	**v**B	0768 (0x300)		0
vminsb[1]	04 (0x04)	**v**D	**v**A	**v**B	0770 (0x302)		
vsrab[1]	04 (0x04)	**v**D	**v**A	**v**B	0772 (0x304)		
vcmpgtsb_x_[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0774 (0x306)	
vmulesb[1]	04 (0x04)	**v**D	**v**A	**v**B	0776 (0x308)		
vcfux[1]	04 (0x04)	**v**D	UIMM	**v**B	0778 (0x30A)		0
vspltisb[1]	04 (0x04)	**v**D	SIMM	0_0000	0780 (0x30C)		
vpkpx[1]	04 (0x04)	**v**D	**v**A	**v**B	0782 (0x30E)		
vaddshs[1]	04 (0x04)	**v**D	**v**A	**v**B	0832 (0x340)		0
vminsh[1]	04 (0x04)	**v**D	**v**A	**v**B	0834 (0x342)		
vsrah[1]	04 (0x04)	**v**D	**v**A	**v**B	0836 (0x344)		
vcmpgtsh_x_[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0838 (0x346)	
vmulesh[1]	04 (0x04)	**v**D	**v**A	**v**B	0840 (0x348)		
vcfsx[1]	04 (0x04)	**v**D	UIMM	**v**B	0842 (0x1E2)		
vspltish[1]	04 (0x04)	**v**D	SIMM	0_0000	0844 (0x34C)		
vupkhpx[1]	04 (0x04)	**v**D	00_000	**v**B	0846 (0x34E)		
vaddsws[1]	04 (0x04)	**v**D	**v**A	**v**B	0896 (0x154)		0
vminsw[1]	04 (0x04)	**v**D	**v**A	**v**B	0898 (0x382)		
vsraw[1]	04 (0x04)	**v**D	**v**A	**v**B	0900 (0x384)		
vcmpgtsw_x_[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0902 (0x386)	
vctuxs[1]	04 (0x04)	**v**D	UIMM	**v**B	0906 (0x38A)		
vspltisw[1]	04 (0x04)	**v**D	SIMM	0_0000	0908 (0x38C)		
vcmpbfp_x_[1]	04 (0x04)	**v**D	**v**A	**v**B	Rc	0966 (0x3C6)	
vctsxs[1]	04 (0x04)	**v**D	UIMM	**v**B	0970 (0x3CA)		

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
vupklpx[1]	04 (0x04)	vD	00_000	vB	0974 (0x3CE)	
vsububm[1]	04 (0x04)	vD	vA	vB	1024 (0x400)	
vavgub[1]	04 (0x04)	vD	vA	vB	1026 (0x402)	0
vand[1]	04 (0x04)	vD	vA	vB	1028 (0x118)	0
vmaxfp[1]	04 (0x04)	vD	vA	vB	1034 (0x040A)	
vslo[1]	04 (0x04)	vD	vA	vB	1036 (0x40C)	
vsubuhm[1]	04 (0x04)	vD	vA	vB	1088 (0x440)	
vavguh[1]	04 (0x04)	vD	vA	vB	1090 (0x442)	0
vandc[1]	04 (0x04)	vD	vA	vB	1092 (0x444)	0
vminfp[1]	04 (0x04)	vD	vA	vB	1098 (0x44A)	
vsro[1]	04 (0x04)	vD	vA	vB	1100 (0x44C)	
vsubuwm[1]	04 (0x04)	vD	vA	vB	1152 (0x480)	
vavguw[1]	04 (0x04)	vD	vA	vB	1154 (0x482)	0
vor[1]	04 (0x04)	vD	vA	vB	1156 (0x484)	
vxor[1]	04 (0x04)	vD	vA	vB	1220 (0x4C4)	
vavgsb[1]	04 (0x04)	vD	vA	vB	1282 (0x502)	0
vnor[1]	04 (0x04)	vD	vA	vB	1284 (0x504)	
vavgsh[1]	04 (0x04)	vD	vA	vB	1346 (0x542)	0
vsubcuw[1]	04 (0x04)	vD	vA	vB	1408 (0x580)	
vavgsw[1]	04 (0x04)	vD	vA	vB	1410 (0x582)	0
vsububs[1]	04 (0x04)	vD	vA	vB	1536 (0x600)	
mfvscr[1]	04 (0x04)	vD	00_000	0_0000	1540 (0x604)	0
vsum4ubs[1]	04 (0x04)	vD	vA	vB	1544 (0x608)	
vsubuhs[1]	04 (0x04)	vD	vA	vB	1600 (0x640)	
mtvscr[1]	04 (0x04)	000_00	00_000	vB	1604 (0x644)	0
vsum4shs[1]	04 (0x04)	vD	vA	vB	1608 (0x648)	
vsubuws[1]	04 (0x04)	vD	vA	vB	1664 (0x680)	
vsum2sws[1]	04 (0x04)	vD	vA	vB	1672 (0x688)	
vsubsbs[1]	04 (0x04)	vD	vA	vB	1792 (0x700)	
vsum4sbs[1]	04 (0x04)	vD	vA	vB	1800 (0x708)	
vsubshs[1]	04 (0x04)	vD	vA	vB	1856 (0x740)	
vsubsws[1]	04 (0x04)	vD	vA	vB	1920 (0x780)	
vsumsws[1]	04 (0x04)	vD	vA	vB	1928 (0x788)	

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mulli	07 (0x07)		D					A					SIMM															
subfic	08 (0x08)		D					A					SIMM															
cmpli	10 (0x0A)		crfD			0	L	A					UIMM															
cmpi	11 (0x0B)		crfD			0	L	A					SIMM															
addic	12 (0xC)		D					A					SIMM															
addic.	13 (0xD)		D					A					SIMM															
addi	14 (0xE)		D					A					SIMM															
addis	15 (0xF)		D					A					SIMM															
bc*x*	16 (0x10)		BO					BI					BD														AA	LK
sc	17 (0x11)		000_0000_0000_0000_0000_0000_00																								1	0
b*x*	18 (0x12)		LI																								AA	LK
mcrf	19 (0x13)		crfD			00		crfS			00		0_0000					0000 (0x000)										0
bclr*x*	19 (0x13)		BO					BI					0_0000					0016 (0x010)										LK
crnor	19 (0x13)		crbD					crbA					crbB					0033 (0x21)										0
rfi[2]	19 (0x13)		000_00					00_000					0_0000					0050 (0x032)										0
crandc	19 (0x13)		crbD					crbA					crbB					0129 (0x081)										0
isync	19 (0x13)		000_00					00_000					0_0000					0150 (0x096)										0
crxor	19 (0x13)		crbD					crbA					crbB					0193 (0C1)										0
crnand	19 (0x13)		crbD					crbA					crbB					0225 (0x0E1)										0
crand	19 (0x13)		crbD					crbA					crbB					0257 (0x101)										0
creqv	19 (0x13)		crbD					crbA					crbB					0289 (0x121)										0
crorc	19 (0x13)		crbD					crbA					crbB					0417 (0x1A1)										0
cror	19 (0x13)		crbD					crbA					crbB					0449 (0x1C1)										0
bcctr*x*	19 (0x13)		BO					BI					0_0000					0528 (0x210)										LK
rlwimi*x*	20 (0x14)		S					A					SH					MB					ME					Rc
rlwinm*x*	21 (0x15)		S					A					SH					MB					ME					Rc
rlwnm*x*	23 (0x17)		S					A					B					MB					ME					Rc
ori	24 (0x18)		S					A					UIMM															
oris	25 (0x19)		S					A					UIMM															
xori	26 (0x1A)		S					A					UIMM															
xoris	27 (0x1B)		S					A					UIMM															
andi.	28 (0x1C)		S					A					UIMM															
andis.	29 (0x1D)		S					A					UIMM															

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

<table>
<tr><th>Name</th><th>0 5</th><th>6 7 8</th><th>9</th><th>10</th><th>11</th><th>12 13 14 15</th><th>16 17 18 19</th><th>20</th><th>21</th><th>22 23 24 25 26 27 28 29 30</th><th>31</th></tr>
<tr><td>cmp</td><td>31 (0x1F)</td><td>crfD</td><td>0</td><td>L</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0000 (0x000)</td><td>0</td></tr>
<tr><td>tw</td><td>31 (0x1F)</td><td colspan="3">TO</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0004 (0x004)</td><td>0</td></tr>
<tr><td>lvsl[1]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0006 (0x006)</td><td>0</td></tr>
<tr><td>lvebx[1]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0007 (0x007)</td><td>0</td></tr>
<tr><td>subfc<i>x</i></td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>0008 (0x008)</td><td>Rc</td></tr>
<tr><td>addc<i>x</i></td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>0010 (0x00A)</td><td>Rc</td></tr>
<tr><td>mulhwu<i>x</i></td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td>0</td><td>0011 (0x00B)</td><td>Rc</td></tr>
<tr><td>mfcr</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">00_000</td><td colspan="2">0_0000</td><td colspan="2">0019 (0x013)</td><td>0</td></tr>
<tr><td>lwarx</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0020 (0x014)</td><td>0</td></tr>
<tr><td>lwzx</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0023 (0x017)</td><td>0</td></tr>
<tr><td>slw<i>x</i></td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0024 (0x018)</td><td>Rc</td></tr>
<tr><td>cntlzw<i>x</i></td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="2">A</td><td colspan="2">0_0000</td><td colspan="2">0026 (0x01A)</td><td>Rc</td></tr>
<tr><td>and<i>x</i></td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0028 (0x01C)</td><td>Rc</td></tr>
<tr><td>cmpl</td><td>31 (0x1F)</td><td>crfD</td><td>0</td><td>L</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0032 (0x020)</td><td>0</td></tr>
<tr><td>lvsr[1]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0038 (0x026)</td><td>0</td></tr>
<tr><td>lvehx[1]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0039 (0x027)</td><td>0</td></tr>
<tr><td>subf<i>x</i></td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>0040 (0x028)</td><td>Rc</td></tr>
<tr><td>dcbst</td><td>31 (0x1F)</td><td colspan="3">000_00</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0054 (0x036)</td><td>0</td></tr>
<tr><td>lwzux</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0055 (0x037)</td><td>0</td></tr>
<tr><td>andc<i>x</i></td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0060 (0x03C)</td><td>Rc</td></tr>
<tr><td>lvewx[1]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0071 (0x047)</td><td>0</td></tr>
<tr><td>mulhw<i>x</i></td><td>31(0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td>0</td><td>0075 (0x04B)</td><td>Rc</td></tr>
<tr><td>mfmsr[2]</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">00_000</td><td colspan="2">0_0000</td><td colspan="2">0083 (0x053)</td><td>0</td></tr>
<tr><td>dcbf</td><td>31 (0x1F)</td><td colspan="3">000_00</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0086 (0x056)</td><td>0</td></tr>
<tr><td>lbzx</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0087 (0x057)</td><td>0</td></tr>
<tr><td>lvx[1]</td><td>31 (0x1F)</td><td colspan="3">vD</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0103 (0x067)</td><td>0</td></tr>
<tr><td>neg<i>x</i></td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">0_0000</td><td>OE</td><td>0104 (0x068)</td><td>Rc</td></tr>
<tr><td>lbzux</td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0119 (0x077)</td><td>0</td></tr>
<tr><td>nor<i>x</i></td><td>31 (0x1F)</td><td colspan="3">S</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0124 (0x07C)</td><td>Rc</td></tr>
<tr><td>stvebx[1]</td><td>31 (0x1F)</td><td colspan="3">vS</td><td colspan="2">A</td><td colspan="2">B</td><td colspan="2">0135 (0x127)</td><td>0</td></tr>
<tr><td>subfe<i>x</i></td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>0136 (0x088)</td><td>Rc</td></tr>
<tr><td>adde<i>x</i></td><td>31 (0x1F)</td><td colspan="3">D</td><td colspan="2">A</td><td colspan="2">B</td><td>OE</td><td>0138 (0x08A)</td><td>Rc</td></tr>
<tr><td>mtcrf</td><td>31 (0x1F)</td><td colspan="3">S</td><td>0</td><td colspan="2">CRM</td><td>0</td><td colspan="2">0144 (0x090)</td><td>0</td></tr>
</table>

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6	7 8	9 10	11	12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
mtmsr[2]	31 (0x1F)	S			00_000		0_0000	0146 (0x092)		0
stwcx.	31 (0x1F)	S			A		B	0150 (0x096)		1
stwx	31 (0x1F)	S			A		B	0151 (0x097)		0
stvehx[1]	31 (0x1F)	vS			A		B	0167 (0x0A7)		0
stwux	31 (0x1F)	S			A		B	0183 (0x0B7)		0
stvewx[1]	31 (0x1F)	vS			A		B	0199 (0x0C7)		0
subfze*x*	31 (0x1F)	D			A		0_0000	OE	0200 (0x0C8)	Rc
addze*x*	31 (0x1F)	D			A		0_0000	OE	0202 (0x0CA)	Rc
mtsr[2]	31 (0x1F)	S			0	SR	0_0000	0210 (0x001)		0
stbx	31 (0x1F)	S			A		B	0215 (0x0D7)		0
stvx[1]	31 (0x1F)	vS			A		B	0231 (0x01F)		0
subfme*x*	31 (0x1F)	D			A		0_0000	OE	0232 (0x0E8)	Rc
addme*x*	31 (0x1F)	D			A		0_0000	OE	0234 (0x0EA)	Rc
mullw*x*	31 (0x1F)	D			A		B	OE	0235 (0x0EB)	Rc
mtsrin[2]	31 (0x1F)	S			00_000		B	0242 (0x0F2)		0
dcbtst	31 (0x1F)	000_00			A		B	0246 (0x0F6)		0
stbux	31 (0x1F)	S			A		B	0247 (0x0F7)		0
add*x*	31 (0x1F)	D			A		B	OE	0266 (0x10A)	Rc
dcbt	31 (0x1F)	000_00			A		B	0278 (0x116)		0
lhzx	31 (0x1F)	D			A		B	0279 (0x117)		0
eqv*x*	31 (0x1F)	S			A		B	0284 (0x11C)		Rc
tlbie[2, 3]	31 (0x1F)	000_00			00_000		B	0306 (0x132)		0
eciwx[3]	31 (0x1F)	D			A		B	0310 (0x136)		0
lhzux	31 (0x1F)	D			A		B	0311 (0x137)		0
xor*x*	31 (0x1F)	S			A		B	0316 (0x13C)		Rc
mfspr[4]	31 (0x1F)	D			spr			0339 (0x153)		0
dst[1]	31 (0x1F)	0	00	STRM	A		B	0342 (0x156)		0
dstt[1]	31 (0x1F)	1	00	STRM	A		B	0342 (0x156)		0
lhax	31 (0x1F)	D			A		B	0343 (0x157)		0
lvxl[1]	31 (0x1F)	vD			A		B	0359 (0x167)		0
tlbia[5]	31 (0x1F)	000_00			00_000		0_0000	0370 (0x172)		0
mftb	31 (0x1F)	D			tbr			0371 (0x173)		0
dstst[1]	31 (0x1F)	0	00	STRM	A		B	0374 (0x176)		0

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
dststt[1]	31 (0x1F)	1 00 STRM	A	B	0374 (0x176)	0
lhaux	31 (0x1F)	D	A	B	0375 (0x177)	0
sthx	31 (0x1F)	S	A	B	0407 (0x197)	0
orc*x*	31 (0x1F)	S	A	B	0412 (0x19C)	Rc
ecowx[3]	31 (0x1F)	S	A	B	0438 (0x1B6)	0
sthux	31 (0x1F)	S	A	B	0439 (0x1B7)	0
or*x*	31 (0x1F)	S	A	B	0444 (0x1BC)	Rc
divwu*x*	31 (0x1F)	D	A	B	OE 0459 (0x1CB)	Rc
mtspr[4]	31 (0x1F)	S	spr		0467 (0x1D3)	0
dcbi[2]	31 (0x1F)	000_00	A	B	0470 (0x1D6)	0
nand*x*	31 (0x1F)	S	A	B	0476 (0x1DC)	Rc
stvxl[1]	31 (0x1F)	vS	A	B	0487 (0x1E7)	0
divw*x*	31 (0x1F)	D	A	B	OE 0491 (0x1EB)	Rc
mcrxr	31 (0x1F)	crfD 00	00_000	0_0000	0512 (0x200)	0
lswx[6]	31 (0x1F)	D	A	B	0533 (0x215)	0
lwbrx	31 (0x1F)	D	A	B	0534 (0x216)	0
lfsx	31 (0x1F)	D	A	B	0535 (0x217)	0
srw*x*	31 (0x1F)	S	A	B	0536 (0x218)	Rc
tlbsync[2, 3]	31 (0x1F)	000_00	00_000	0_0000	0566 (0x236)	0
lfsux	31 (0x1F)	D	A	B	0567 (0x237)	0
mfsr[2]	31 (0x1F)	D	0 SR	0_0000	0595 (0x099)	0
lswi[6]	31 (0x1F)	D	A	NB	0597 (0x255)	0
sync	31 (0x1F)	000_00	00_000	0_0000	0598 (0x256)	0
lfdx	31 (0x1F)	D	A	B	0599 (0x257)	0
lfdux	31 (0x1F)	D	A	B	0631 (0x277)	0
mfsrin [2]	31 (0x1F)	D	00_000	B	0659 (0x293)	0
stswx[6]	31 (0x1F)	S	A	B	0661 (0x295)	0
stwbrx	31 (0x1F)	S	A	B	0662 (0x296)	0
stfsx	31 (0x1F)	S	A	B	0663 (0x297)	0
stfsux	31 (0x1F)	S	A	B	0695 (0x2B7)	0
stswi[6]	31 (0x1F)	S	A	NB	0725 (0x2D5)	0
stfdx	31 (0x1F)	S	A	B	0727 (0x2D7)	0
dcba[3]	31 (0x1F)	000_00	A	B	0758 (0x2F6)	0

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6	7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stfdux	31 (0x1F)	S			A	B	0759 (0x2F7)	0
lhbrx	31 (0x1F)	D			A	B	0790 (0x316)	0
sraw*x*	31 (0x1F)	S			A	B	0792 (0x318)	Rc
dss[1]	31 (0x1F)	0	00	STRM	00_000	0_0000	0822 (0x336)	0
dssall[1]	31 (0x1F)	1	00	STRM	00_000	0_0000	0822 (0x336)	0
srawi*x*	31 (0x1F)	S			A	SH	0824 (0x338)	Rc
eieio	31 (0x1F)	000_00			00_000	0_0000	0854 (0x356)	0
sthbrx	31 (0x1F)	S			A	B	0918 (0x396)	0
extsh*x*	31 (0x1F)	S			A	0_0000	0922 (0x39A)	Rc
extsb*x*	31 (0x1F)	S			A	0_0000	0954 (0x3BA)	Rc
tlbld[2, 3]	31 (0x1F)	000_00			00_000	B	0978 (0x3D2)	0
icbi	31 (0x1F)	000_00			A	B	0982 (0x3D6)	0
stfiwx[3]	31 (0x1F)	S			A	B	0983 (0x3D7)	0
tlbli[2, 3]	31 (0x1F)	000_00			00_000	B	1010 (0x3F2)	0
dcbz	31 (0x1F)	000_00			A	B	1014 (0x3F6)	0
lwz	32 (0x20)	D			A	d		
lwzu	33 (0x21)	D			A	d		
lbz	34 (0x22)	D			A	d		
lbzu	35 (0x23)	D			A	d		
stw	36 (0x24)	S			A	d		
stwu	37 (0x25)	S			A	d		
stb	38 (0x26)	S			A	d		
stbu	39 (0x27)	S			A	d		
lhz	40 (0x28)	D			A	d		
lhzu	41 (0x29)	D			A	d		
lha	42 (0x2A)	D			A	d		
lhau	43 (0x2B)	D			A	d		
sth	44 (0x2C)	S			A	d		
sthu	45 (0x2D)	S			A	d		
lmw[6]	46 (0x2E)	D			A	d		
stmw[6]	47 (0x2F)	S			A	d		
lfs	48 (0x30)	D			A	d		
lfsu	49 (0x31)	D			A	d		

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
lfd	50 (0x32)	D		A	d			
lfdu	51 (0x33)	D		A	d			
stfs	52 (0x34)	S		A	d			
stfsu	53 (0x35)	S		A	d			
stfd	54 (0x36)	S		A	d			
stfdu	55 (0x37)	S		A	d			
fdivs*x*	59 (0x3B)	D		A	B	0000_0	0018 (0x012)	Rc
fsubs*x*	59 (0x3B)	D		A	B	0000_0	0020 (0x014)	Rc
fadds*x*	59 (0x3B)	D		A	B	0000_0	0021 (0x015)	Rc
fsqrts*x*[5]	59 (0x3B)	D		00_000	B	0000_0	0022 (0x016)	Rc
fres*x*[3]	59 (0x3B)	D		00_000	B	0000_0	0024 (0x018)	Rc
fmuls*x*	59 (0x3B)	D		A	0_0000	C	0025 (0x019)	Rc
fmsubs*x*	59 (0x3B)	D		A	B	C	0028 (0x01C)	Rc
fmadds*x*	59 (0x3B)	D		A	B	C	0029 (0x01D)	Rc
fnmsubs*x*	59 (0x3B)	D		A	B	C	0030 (0x01E)	Rc
fnmadds*x*	59 (0x3B)	D		A	B	C	0031 (0x01F)	Rc
fcmpu	63 (0x3F)	crfD	00	A	B	0000 (0x000)		0
frsp*x*	63 (0x3F)	D		00_000	B	0012 (0xC)		Rc
fctiw*x*	63 (0x3F)	D		00_000	B	0014 (0x00E)		Rc
fctiwz*x*	63 (0x3F)	D		00_000	B	0015 (0x00F)		Rc
fdiv*x*	63 (0x3F)	D		A	B	0000_0	0018 (0x012)	Rc
fsub*x*	63 (0x3F)	D		A	B	0000_0	0020 (0x014)	Rc
fadd*x*	63 (0x3F)	D		A	B	0000_0	0021 (0x015)	Rc
fsqrt*x*[5]	63 (0x3F)	D		00_000	B	0000_0	0022 (0x016)	Rc
fsel*x*[3]	63 (0x3F)	D		A	B	C	0023 (0x017)	Rc
fmul*x*	63 (0x3F)	D		A	0_0000	C	0025 (0x019)	Rc
frsqrte*x*[3]	63 (0x3F)	D		00_000	B	0000_0	0026 (0x01A)	Rc
fmsub*x*	63 (0x3F)	D		A	B	C	0028 (0x01C)	Rc
fmadd*x*	63 (0x3F)	D		A	B	C	0029 (0x01D)	Rc
fnmsub*x*	63 (0x3F)	D		A	B	C	0030 (0x01E)	Rc
fnmadd*x*	63 (0x3F)	D		A	B	C	0031 (0x01F)	Rc
fcmpo	63 (0x3F)	crfD	00	A	B	0032 (0x020)		0
mtfsb1*x*	63 (0x3F)	crbD		00_000	0_0000	0038 (0x026)		Rc

Table A-2. Instructions by Primary and Secondary Opcodes (Dec, Hex) (Continued)

Name	0	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
fneg*x*	63 (0x3F)		D					00_000					B					0040 (0x28)										Rc
mcrfs	63 (0x3F)		crfD			00		crfS			00		0_0000					0064 (0x040)										0
mtfsb0*x*	63 (0x3F)		crbD					00_000					0_0000					0070 (0x046)										Rc
fmr*x*	63 (0x3F)		D					00_000					B					0072 (0x48)										Rc
mtfsfi*x*	63 (0x3F)		crfD			00		00_000					IMM				0	0134 (0x086)										Rc
fnabs*x*	63 (0x3F)		D					00_000					B					0136 (0x88)										Rc
fabs*x*	63 (0x3F)		D					00_000					B					0264 (0x108)										Rc
mffs*x*	63 (0x3F)		D					00_000					0_0000					0583 (0x247)										Rc
mtfsf*x*	63 (0x3F)		0	FM								0	B					0711 (0x2C7)										Rc

[1] AltiVec technology-specific instruction

[2] Supervisor-level instruction

[3] Optional to the architecture but implemented by the MPC7410

[4] Supervisor- and user-level instructions

[5] Optional instruction not implemented by the MPC7410

[6] Load/store string/multiple instruction

A.3 Instructions Sorted by Mnemonic (Binary)

Table A-3 shows instructions listed in alphabetical order by mnemonic with binary values.

Key:

▨ Reserved bits

Table A-3. Instructions by Mnemonic (Bin)

Name	05	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
add*x*	011111	D					A					B					OE	100001 010									Rc
addc*x*	011111	D					A					B					OE	000001010									Rc
adde*x*	011111	D					A					B					OE	010001010									Rc
addi	001110	D					A					SIMM															
addic	001100	D					A					SIMM															
addic.	001101	D					A					SIMM															
addis	001111	D					A					SIMM															
addme*x*	011111	D					A					0_0000					OE	11101010									Rc
addze*x*	011111	D					A					0_0000					OE	11001010									Rc
and*x*	011111	S					A					B					000011100										Rc
andc*x*	011111	S					A					B					000111100										Rc
andi.	011100	S					A					UIMM															
andis.	011101	S					A					UIMM															
b*x*	010010	LI																								AA	LK
bc*x*	010000	BO					BI					BD														AA	LK
bcctr*x*	010011	BO					BI					0_0000					1000010000										LK
bclr*x*	010011	BO					BI					0_0000					0000010000										LK
cmp	011111	crfD			0	L	A					B					0000000000										0
cmpi	001011	crfD			0	L	A					SIMM															
cmpl	011111	crfD			0	L	A					B					0000100000										0
cmpli	001010	crfD			0	L	A					UIMM															

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6	7 8	9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
cntlzw*x*	011111	S			A	0_0000	0000011010		Rc
crand	010011	crbD			crbA	crbB	0100000001		0
crandc	010011	crbD			crbA	crbB	0010000001		0
creqv	010011	crbD			crbA	crbB	0100100001		0
crnand	010011	crbD			crbA	crbB	0011100001		0
crnor	010011	crbD			crbA	crbB	0000100001		0
cror	010011	crbD			crbA	crbB	0111000001		0
crorc	010011	crbD			crbA	crbB	0110100001		0
crxor	010011	crbD			crbA	crbB	0011000001		0
dcba[1]	011111	000_00			A	B	1011110110		0
dcbf	011111	000_00			A	B	0001010110		0
dcbi[2]	011111	000_00			A	B	0111010110		0
dcbst	011111	000_00			A	B	0000110110		0
dcbt	011111	000_00			A	B	0100010110		0
dcbtst	011111	000_00			A	B	0011110110		0
dcbz	011111	000_00			A	B	1111110110		0
divw*x*	011111	D			A	B	OE	11110 1011	Rc
divw*x*	011111	D			A	B	OE	11100 1011	Rc
dss[3]	011111	A	00	STRM	00_000	0_0000	1100110110		0
dssall[3]	011111	A	00	STRM	00_000	0_0000	1100110110		0
dst[3]	011111	T	00	STRM	A	B	0101010110		0
dstst[3]	011111	T	00	STRM	A	B	0101110110		0
dststt[3]	011111	1	00	STRM	A	B	0101110110		0
dstt[3]	011111	1	00	STRM	A	B	0101010110		0
eciwx[1]	011111	D			A	B	0100110110		0
ecowx[1]	011111	S			A	B	0110110110		0
eieio	011111	000_00			00_000	0_0000	1101010110		0
eqv*x*	011111	S			A	B	0100011100		Rc
extsb*x*	011111	S			A	0_0000	1110111010		Rc

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
extsh*x*	011111	S		A	0_0000	1110011010		Rc
fabs*x*	111111	D		00_000	B	0100001000		Rc
fadd*x*	111111	D		A	B	0000_0	1 0101	Rc
fadds*x*	111011	D		A	B	0000_0	1 0101	Rc
fcmpo	111111	crfD	00	A	B	0000100000		0
fcmpu	111111	crfD	00	A	B	0000000000		0
fctiw*x*	111111	D		00_000	B	0000001110		Rc
fctiwz*x*	111111	D		00_000	B	0000001111		Rc
fdiv*x*	111111	D		A	B	0000_0	1 0010	Rc
fdivs*x*	111011	D		A	B	0000_0	1 0010	Rc
fmadd*x*	111111	D		A	B	C	1 1101	Rc
fmadds*x*	111011	D		A	B	C	1 1101	Rc
fmr*x*	111111	D		00_000	B	0001001000		Rc
fmsub*x*	111111	D		A	B	C	1 1100	Rc
fmsubs*x*	111011	D		A	B	C	1 1100	Rc
fmul*x*	111111	D		A	0_0000	C	1 1001	Rc
fmuls*x*	111011	D		A	0_0000	C	1 1001	Rc
fnabs*x*	111111	D		00_000	B	0010001000		Rc
fneg*x*	111111	D		00_000	B	0000101000		Rc
fnmadd*x*	111111	D		A	B	C	1 1111	Rc
fnmadds*x*	111011	D		A	B	C	1 1111	Rc

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fnmsub*x*	111111	D	A	B	C	1 1110	Rc
fnmsubs*x*	111011	D	A	B	C	1 1110	Rc
fres*x*[1]	111011	D	00_000	B	0000_0	1 1000	Rc
frsp*x*	111111	D	00_000	B	0000001100		Rc
frsqrte*x* [1]	111111	D	00_000	B	0000_0	1 1010	Rc
fsel*x* [1]	111111	D	A	B	C	1 0111	Rc
fsqrt*x* [4]	111111	D	00_000	B	0000_0	1 0110	Rc
fsqrts*x* [4]	111011	D	00_000	B	0000_0	1 0110	Rc
fsub*x*	111111	D	A	B	0000_0	1 0100	Rc
fsubs*x*	111011	D	A	B	0000_0	1 0100	Rc
icbi	011111	000_00	A	B	1111010110		0
isync	010011	000_00	00_000	0_0000	0010010110		0
lbz	100010	D	A	d			
lbzu	100011	D	A	d			
lbzux	011111	D	A	B	0001110111		0
lbzx	011111	D	A	B	0001010111		0
lfd	110010	D	A	d			
lfdu	110011	D	A	d			
lfdux	011111	D	A	B	1001110111		0
lfdx	011111	D	A	B	1001010111		0
lfs	110000	D	A	d			
lfsu	110001	D	A	d			
lfsux	011111	D	A	B	1000110111		0
lfsx	011111	D	A	B	1000010111		0
lha	101010	D	A	d			
lhau	101011	D	A	d			
lhaux	011111	D	A	B	0101110111		0

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
lhax	011111	D					A					B					0101010111										0
lhbrx	011111	D					A					B					1100010110										0
lhz	101000	D					A					d															
lhzu	101001	D					A					d															
lhzux	011111	D					A					B					0100110111										0
lhzx	011111	D					A					B					0100010111										0
lmw[5]	101110	D					A					d															
lswi[5]	011111	D					A					NB					1001010101										0
lswx[5]	011111	D					A					B					1000010101										0
lvebx[3]	011111	vD					A					B					0000000111										0
lvehx[3]	011111	vD					A					B					0000100111										0
lvewx[3]	011111	vD					A					B					0001000111										0
lvsl[3]	011111	vD					A					B					0000000110										0
lvsr[3]	011111	vD					A					B					0000100110										0
lvx[3]	011111	vD					A					B					0001100111										0
lvxl[3]	011111	vD					A					B					0101100111										0
lwarx	011111	D					A					B					0000010100										0
lwbrx	011111	D					A					B					1000010110										0
lwz	100000	D					A					d															
lwzu	100001	D					A					d															
lwzux	011111	D					A					B					0000110111										0
lwzx	011111	D					A					B					0000010111										0
mcrf	010011	crfD			00		crfS			00		0_0000					000000000										0
mcrfs	111111	crfD			00		crfS			00		0_0000					001000000										0
mcrxr	011111	crfD			00		00_000					0_0000					1000000000										0
mfcr	011111	D					00_000					0_0000					0000010011										0
mffs*x*	111111	D					00_000					0_0000					1001000111										Rc
mfmsr[2]	011111	D					00_000					0_0000					0001010011										0
mfspr[6]	011111	D					spr										0101010011										0
mfsr[2]	011111	D					0	SR				0_0000					1001010011										0
mfsrin[2]	011111	D					00_000					B					1010010011										0
mftb	011111	D					tbr										0101110011										0

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
mfvscr[3]	000100	**v**D					00_000					0_0000					11000000100										0
mtcrf	011111	S					0	CRM								0	0010010000										0
mtfsb0*x*	111111	crbD					00_000					0_0000					0001000110										Rc
mtfsb1*x*	111111	crbD					00_000					0_0000					0000100110										Rc
mtfsf*x*	111111	0	FM								0	B					1011000111										Rc
mtfsfi*x*	111111	crfD			00		00_000					IMM				0	0010000110										Rc
mtmsr[2]	011111	S					00_000					0_0000					0010010010										0
mtspr[6]	011111	S					spr										0111010011										0
mtsr[2]	011111	S					0	SR				0_0000					0011010010										0
mtsrin [2]	011111	S					00_000					B					0011110010										0
mtvscr[3]	000100	000_00					00_000					**v**B					11001000100										0
mulhw*x*	011111	D					A					B					0	001001011									Rc
mulhwu*x*	011111	D					A					B					0	000001011									Rc
mulli	000111	D					A					SIMM															
mullw*x*	011111	D					A					B					OE	011101011									Rc
nand*x*	011111	S					A					B					0111011100										Rc
neg*x*	011111	D					A					0_0000					OE	001101000									Rc
nor*x*	011111	S					A					B					0001111100										Rc
or*x*	011111	S					A					B					0110111100										Rc
orc*x*	011111	S					A					B					0110011100										Rc
ori	011000	S					A					UIMM															
oris	011001	S					A					UIMM															
rfi[2]	010011	000_00					00_000					0_0000					0000110010										0
rlwimi*x*	010100	S					A					SH					MB					ME					Rc

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
rlwinm*x*	010101	S	A	SH	MB	ME	Rc
rlwnm*x*	010111	S	A	B	MB	ME	Rc
sc	010001	000_0000_0000_0000_0000_0000_00				1	0
slw*x*	011111	S	A	B	0000011000		Rc
sraw*x*	011111	S	A	B	1100011000		Rc
srawi*x*	011111	S	A	SH	1100011000		Rc
srw*x*	011111	S	A	B	1000011000		Rc
stb	100110	S	A	d			
stbu	100111	S	A	d			
stbux	011111	S	A	B	0011110111		0
stbx	011111	S	A	B	0011010111		0
stfd	110110	S	A	d			
stfdu	110111	S	A	d			
stfdux	011111	S	A	B	1011110111		0
stfdx	011111	S	A	B	1011010111		0
stfiwx[1]	011111	S	A	B	1111010111		0
stfs	110100	S	A	d			
stfsu	110101	S	A	d			
stfsux	011111	S	A	B	1010110111		0
stfsx	011111	S	A	B	1010010111		0
sth	101100	S	A	d			
sthbrx	011111	S	A	B	1110010110		0
sthu	101101	S	A	d			
sthux	011111	S	A	B	110110111		0
sthx	011111	S	A	B	110010111		0
stmw[5]	101111	S	A	d			
stswi[5]	011111	S	A	NB	1011010101		0
stswx[5]	011111	S	A	B	1010010101		0
stvebx[3]	011111	vS	A	B	0010000111		0

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
stvehx[3]	011111	vS	A	B	0010100111		0
stvewx[3]	011111	vS	A	B	0011000111		0
stvx[3]	011111	vS	A	B	0011100111		0
stvxl[3]	011111	vS	A	B	0111100111		0
stw	100100	S	A	d			
stwbrx	011111	S	A	B	1010010110		0
stwcx.	011111	S	A	B	10010110		1
stwu	100101	S	A	d			
stwux	011111	S	A	B	10110111		0
stwx	011111	S	A	B	10010111		0
subf*x*	011111	D	A	B	OE	000101000	Rc
subfc*x*	011111	D	A	B	OE	000001000	Rc
subfe*x*	011111	D	A	B	OE	010001000	Rc
subfic	001000	D	A	SIMM			
subfme*x*	011111	D	A	0_0000	OE	011101000	Rc
subfze*x*	011111	D	A	0_0000	OE	011001000	Rc
sync	011111	000_00	00_000	0_0000	1001010110		0
tlbia [4]	011111	000_00	00_000	0_0000	0101110010		0
tlbie[1, 2]	011111	000_00	00_000	B	0100110010		0
tlbsync[1, 2]	011111	000_00	00_000	0_0000	1000110110		0
tw	011111	TO	A	B	0000000100		0
twi	000011	TO	A	SIMM			
vaddcuw[3]	000100	vD	vA	vB	0110000000		0
vaddfp[3]	000100	vD	vA	vB	0000001010		0
vaddsbs[3]	000100	vD	vA	vB	1100000000		0
vaddshs[3]	000100	vD	vA	vB	1101000000		0
vaddsws[3]	000100	vD	vA	vB	1110000000		0
vaddubm[3]	000100	vD	vA	vB	0000000000		0
vaddubs[3]	000100	vD	vA	vB	1000000000		0
vadduhm[3]	000100	vD	vA	vB	001000000		0

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
vadduhs[3]	000100	vD	vA	vB	1001000000		0
vadduwm[3]	000100	vD	vA	vB	0010000000		0
vadduws[3]	000100	vD	vA	vB	1010000000		0
vand[3]	000100	vD	vA	vB	10000000100		0
vandc[3]	000100	vD	vA	vB	10001000100		0
vavgsb[3]	000100	vD	vA	vB	10100000010		0
vavgsh[3]	000100	vD	vA	vB	10101000010		0
vavgsw[3]	000100	vD	vA	vB	10110000010		0
vavgub[3]	000100	vD	vA	vB	10000000010		0
vavguh[3]	000100	vD	vA	vB	10001000010		0
vavguw[3]	000100	vD	vA	vB	10010000010		0
vcfsx[3]	000100	vD	UIMM	vB	01101001010		
vcfux[3]	000100	vD	UIMM	vB	1100001010		0
vcmpbfp*x*[3]	000100	vD	vA	vB	Rc	1111000110	
vcmpeqfp*x*[3]	000100	vD	vA	vB	Rc	0011000110	
vcmpequb*x*[3]	000100	vD	vA	vB	Rc	0000000110	
vcmpequh*x*[3]	000100	vD	vA	vB	Rc	0001000110	
vcmpequw*x*[3]	000100	vD	vA	vB	Rc	0010000110	
vcmpgefp*x*[3]	000100	vD	vA	vB	Rc	0111000110	
vcmpgtfp*x*[3]	000100	vD	vA	vB	Rc	1011000110	
vcmpgtsb*x*[3]	000100	vD	vA	vB	Rc	1100000110	
vcmpgtsh*x*[3]	000100	vD	vA	vB	Rc	1101000110	
vcmpgtsw*x*[3]	000100	vD	vA	vB	Rc	1110000110	
vcmpgtub*x*[3]	000100	vD	vA	vB	Rc	1000000110	
vcmpgtuh*x*[3]	000100	vD	vA	vB	Rc	1001000110	

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30 31
vcmpgtuw*x*[3]	000100	vD	vA	vB	Rc	1010000110	
vctsxs[3]	000100	vD	UIMM	vB	1111001010		
vctuxs[3]	000100	vD	UIMM	vB	1110001010		
vexptefp[3]	000100	vD	00_000	vB	110001010		
vlogefp[3]	000100	vD	00_000	vB	111001010		
vmaddfp[3]	000100	vD	vA	vB	vC		101110
vmaxfp[3]	000100	vD	vA	vB	10000001010		
vmaxsb[3]	000100	vD	vA	vB	0100000010		
vmaxsh[3]	000100	vD	vA	vB	0101000010		
vmaxsw[3]	000100	vD	vA	vB	0110000010		
vmaxub[3]	000100	vD	vA	vB	0000000010		
vmaxuh[3]	000100	vD	vA	vB	0001000010		
vmaxuw[3]	000100	vD	vA	vB	0010000010		
vmhaddshs[3]	000100	vD	vA	vB	vC		100000
vmhraddshs[3]	000100	vD	vA	vB	vC		100001
vminfp[3]	000100	vD	vA	vB	10001001010		
vminsb[3]	000100	vD	vA	vB	1100000010		
vminsh[3]	000100	vD	vA	vB	1101000010		
vminsw[3]	000100	vD	vA	vB	1110000010		
vminub[3]	000100	vD	vA	vB	1000000010		
vminuh[3]	000100	vD	vA	vB	1001000010		
vminuw[3]	000100	vD	vA	vB	1010000010		
vmladduhm[3]	000100	vD	vA	vB	vC		100010
vmrghb[3]	000100	vD	vA	vB	0000001100		
vmrghh[3]	000100	vD	vA	vB	0001001100		
vmrghw[3]	000100	vD	vA	vB	0010001100		
vmrglb[3]	000100	vD	vA	vB	0100001100		
vmrglh[3]	000100	vD	vA	vB	0101001100		
vmrglw[3]	000100	vD	vA	vB	0110001100		
vmsummbm[3]	000100	vD	vA	vB	vC		100101
vmsumshm[3]	000100	vD	vA	vB	vC		101000
vmsumshs[3]	000100	vD	vA	vB	vC		101001

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30 31
vmsumubm[3]	000100	vD	vA	vB	vC	100100
vmsumuhm[3]	000100	vD	vA	vB	vC	100110
vmsumuhs[3]	000100	vD	vA	vB	vC	100111
vmulesb[3]	000100	vD	vA	vB	0100001000	
vmulesh[3]	000100	vD	vA	vB	1101001000	
vmuleub[3]	000100	vD	vA	vB	1000001000	
vmuleuh[3]	000100	vD	vA	vB	1001001000	
vmulosb[3]	000100	vD	vA	vB	0100001000	
vmulosh[3]	000100	vD	vA	vB	0101001000	
vmuloub[3]	000100	vD	vA	vB	0000001000	
vmulouh[3]	000100	vD	vA	vB	0001001000	
vnmsubfp[3]	000100	vD	vA	vB	vC	101111
vnor[3]	000100	vD	vA	vB	10100000100	
vor[3]	000100	vD	vA	vB	10010000100	
vperm[3]	000100	vD	vA	vB	vC	101011
vpkpx[3]	000100	vD	vA	vB	1100001110	
vpkshss[3]	000100	vD	vA	vB	0110001110	
vpkshus[3]	000100	vD	vA	vB	0100001110	
vpkswss[3]	000100	vD	vA	vB	0111001110	
vpkswus[3]	000100	vD	vA	vB	0101001110	
vpkuhum[3]	000100	vD	vA	vB	0000001110	
vpkuhus[3]	000100	vD	vA	vB	0010001110	
vpkuwum[3]	000100	vD	vA	vB	0001001110	
vpkuwus[3]	000100	vD	vA	vB	0011001110	
vrefp[3]	000100	vD	00_000	vB	0100001010	
vrfim[3]	000100	vD	00_000	vB	1011001010	
vrfin[3]	000100	vD	00_000	vB	1000001010	
vrfip[3]	000100	vD	00_000	vB	1010001010	
vrfiz[3]	000100	vD	00_000	vB	1001001010	
vrlb[3]	000100	vD	vA	vB	0000000100	
vrlh[3]	000100	vD	vA	vB	0001000100	
vrlw[3]	000100	vD	vA	vB	0010000100	
vrsqrtefp[3]	000100	vD	00_000	vB	0101001010	

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30 31
vsel^3	000100	vD	vA	vB	vC		101010
vsl^3	000100	vD	vA	vB	0111000100		
vslb^3	000100	vD	vA	vB	0100000100		
vsldoi^3	000100	vD	vA	vB	0	SH	101100
vslh^3	000100	vD	vA	vB	0101000100		
vslo^3	000100	vD	vA	vB	10000001100		
vslw^3	000100	vD	vA	vB	0110000100		
vspltb^3	000100	vD	UIMM	vB	1000001100		
vsplth^3	000100	vD	UIMM	vB	1001001100		
vspltisb^3	000100	vD	SIMM	0_0000	1100001100		
vspltish^3	000100	vD	SIMM	0_0000	1101001100		
vspltisw^3	000100	vD	SIMM	0_0000	1110001100		
vspltw^3	000100	vD	UIMM	vB	1010001100		
vsr^3	000100	vD	vA	vB	1011000100		
vsrab^3	000100	vD	vA	vB	1100000100		
vsrah^3	000100	vD	vA	vB	1101000100		
vsraw^3	000100	vD	vA	vB	1110000100		
vsrb^3	000100	vD	vA	vB	1000000100		
vsrh^3	000100	vD	vA	vB	1001000100		
vsro^3	000100	vD	vA	vB	10001001100		
vsrw^3	000100	vD	vA	vB	1010000100		
vsubcuw^3	000100	vD	vA	vB	10110000000		
vsubfp^3	000100	vD	vA	vB	0001001010		
vsubsbs^3	000100	vD	vA	vB	11100000000		
vsubshs^3	000100	vD	vA	vB	11101000000		
vsubsws^3	000100	vD	vA	vB	11110000000		
vsububm^3	000100	vD	vA	vB	10000000000		
vsububs^3	000100	vD	vA	vB	11000000000		
vsubuhm^3	000100	vD	vA	vB	10001000000		
vsubuhs^3	000100	vD	vA	vB	11001000000		
vsubuwm^3	000100	vD	vA	vB	10010000000		
vsubuws^3	000100	vD	vA	vB	11010000000		
vsumsws^3	000100	vD	vA	vB	11110001000		

Table A-3. Instructions by Mnemonic (Bin) (Continued)

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
vsum2sws[3]	000100	**v**D	**v**A	**v**B	11010001000	
vsum4sbs[3]	000100	**v**D	**v**A	**v**B	11100001000	
vsum4shs[3]	000100	**v**D	**v**A	**v**B	11001001000	
vsum4ubs[3]	000100	**v**D	**v**A	**v**B	11000001000	
vupkhpx[3]	000100	**v**D	00_000	**v**B	1101001110	
vupkhsb[3]	000100	**v**D	00_000	**v**B	1000001110	
vupkhsh[3]	000100	**v**D	00_000	**v**B	1001001110	
vupklpx[3]	000100	**v**D	00_000	**v**B	1111001110	
vupklsb[3]	000100	**v**D	00_000	**v**B	1010001110	
vupklsh[3]	000100	**v**D	00_000	**v**B	1011001110	
vxor[3]	000100	**v**D	**v**A	**v**B	10011000100	
xor*x*	011111	S	A	B	0100111100	Rc
xori	011010	S	A	UIMM		
xoris	011011	S	A	UIMM		

[1] Optional to the architecture but implemented by the MPC7410

[2] Supervisor-level instruction

[3] AltiVec technology-specific instruction

[4] Optional instruction not implemented by the MPC7410

[5] Load/store string/multiple instruction

[6] Supervisor- and user-level instruction

A.4 Instructions Sorted by Opcode (Binary)

Table A-4 lists the instructions implemented in the MPC7410 in binary numerical order by opcode.

Key:

▨ Reserved bits

Table A-4. Instructions by Primary and Secondary Opcode (Bin)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30	31
twi	000011	TO	A	SIMM				
vaddubm[1]	000100	vD	vA	vB	0000000000			0
vmaxub[1]	000100	vD	vA	vB	0000000010			
vrlb[1]	000100	vD	vA	vB	0000000100			
vcmpequb*x*[1]	000100	vD	vA	vB	Rc	0000000110		
vmuloub[1]	000100	vD	vA	vB	0000001000			
vaddfp[1]	000100	vD	vA	vB	0000001010			0
vmrghb[1]	000100	vD	vA	vB	0000001100			
vpkuhum[1]	000100	vD	vA	vB	0000001110			
vmhaddshs[1]	000100	vD	vA	vB	vC		100000	
vmhraddshs[1]	000100	vD	vA	vB	vC		100001	
vmladduhm[1]	000100	vD	vA	vB	vC		100010	
vmsumubm[1]	000100	vD	vA	vB	vC		100100	
vmsummbm[1]	000100	vD	vA	vB	vC		100101	
vmsumuhm[1]	000100	vD	vA	vB	vC		100110	
vmsumuhs[1]	000100	vD	vA	vB	vC		100111	
vmsumshm[1]	000100	vD	vA	vB	vC		101000	
vmsumshs[1]	000100	vD	vA	vB	vC		101001	
vsel[1]	000100	vD	vA	vB	vC		101010	
vperm[1]	000100	vD	vA	vB	vC		101011	
vsldoi[1]	000100	vD	vA	vB	0	SH	101100	
vmaddfp[1]	000100	vD	vA	vB	vC		101110	
vnmsubfp[1]	000100	vD	vA	vB	vC		101111	
vadduhm[1]	000100	vD	vA	vB	001000000			0
vmaxuh[1]	000100	vD	vA	vB	0001000010			
vrlh[1]	000100	vD	vA	vB	0001000100			
vcmpequh*x*[1]	000100	vD	vA	vB	Rc	0001000110		
vmulouh[1]	000100	vD	vA	vB	0001001000			

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
vsubfp[1]	000100	vD	vA	vB	0001001010		
vmrghh[1]	000100	vD	vA	vB	0001001100		
vpkuwum[1]	000100	vD	vA	vB	0001001110		
vadduwm[1]	000100	vD	vA	vB	0010000000		0
vmaxuw[1]	000100	vD	vA	vB	0010000010		
vrlw[1]	000100	vD	vA	vB	0010000100		
vcmpequw*x*[1]	000100	vD	vA	vB	Rc	0010000110	
vmrghw[1]	000100	vD	vA	vB	0010001100		
vpkuhus[1]	000100	vD	vA	vB	0010001110		
vcmpeqfp*x*[1]	000100	vD	vA	vB	Rc	0011000110	
vpkuwus[1]	000100	vD	vA	vB	0011001110		
vmaxsb[1]	000100	vD	vA	vB	0100000010		
vslb[1]	000100	vD	vA	vB	0100000100		
vmulosb[1]	000100	vD	vA	vB	0100001000		
vrefp[1]	000100	vD	00_000	vB	0100001010		
vmrglb[1]	000100	vD	vA	vB	0100001100		
vpkshus[1]	000100	vD	vA	vB	0100001110		
vmaxsh[1]	000100	vD	vA	vB	0101000010		
vslh[1]	000100	vD	vA	vB	0101000100		
vmulosh[1]	000100	vD	vA	vB	0101001000		
vrsqrtefp[1]	000100	vD	00_000	vB	0101001010		
vmrglh[1]	000100	vD	vA	vB	0101001100		
vpkswus[1]	000100	vD	vA	vB	0101001110		
vaddcuw[1]	000100	vD	vA	vB	0110000000		0
vmaxsw[1]	000100	vD	vA	vB	0110000010		
vslw[1]	000100	vD	vA	vB	0110000100		
vexptefp[1]	000100	vD	00_000	vB	110001010		
vmrglw[1]	000100	vD	vA	vB	0110001100		
vpkshss[1]	000100	vD	vA	vB	0110001110		
vsl[1]	000100	vD	vA	vB	0111000100		
vcmpgefp*x*[1]	000100	vD	vA	vB	Rc	0111000110	
vlogefp[1]	000100	vD	00_000	vB	111001010		
vpkswss[1]	000100	vD	vA	vB	0111001110		

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 … 5	6 … 10	11 … 15	16 … 20	21	22 … 30	31
vaddubs[1]	000100	vD	vA	vB	1000000000		0
vminub[1]	000100	vD	vA	vB	1000000010		
vsrb[1]	000100	vD	vA	vB	1000000100		
vcmpgtub*x*[1]	000100	vD	vA	vB	Rc	1000000110	
vmuleub[1]	000100	vD	vA	vB	1000001000		
vrfin[1]	000100	vD	00_000	vB	1000001010		
vspltb[1]	000100	vD	UIMM	vB	1000001100		
vupkhsb[1]	000100	D	00_000	B	1000001110		
vadduhs[1]	000100	vD	vA	vB	1001000000		0
vminuh[1]	000100	vD	vA	vB	1001000010		
vsrh[1]	000100	vD	vA	vB	1001000100		
vcmpgtuh*x* [1]	000100	vD	vA	vB	Rc	1001000110	
vmuleuh[1]	000100	vD	vA	vB	1001001000		
vrfiz[1]	000100	vD	00_000	vB	1001001010		
vsplth[1]	000100	vD	UIMM	vB	1001001100		
vupkhsh[1]	000100	D	00_000	B	1001001110		
vadduws[1]	000100	vD	vA	vB	1010000000		0
vminuw[1]	000100	vD	vA	vB	1010000010		
vsrw[1]	000100	vD	vA	vB	1010000100		
vcmpgtuw*x*[1]	000100	vD	vA	vB	Rc	1010000110	
vrfip[1]	000100	vD	00_000	vB	1010001010		
vspltw[1]	000100	vD	UIMM	vB	1010001100		
vupklsb[1]	000100	D	00_000	B	1010001110		
vsr[1]	000100	vD	vA	vB	1011000100		
vcmpgtfp*x*[1]	000100	vD	vA	vB	Rc	1011000110	
vrfim[1]	000100	vD	00_000	vB	1011001010		
vupklsh[1]	000100	D	00_000	B	1011001110		
vaddsbs[1]	000100	vD	vA	vB	1100000000		0
vminsb[1]	000100	vD	vA	vB	1100000010		
vsrab[1]	000100	vD	vA	vB	1100000100		
vcmpgtsb*x*[1]	000100	vD	vA	vB	Rc	1100000110	
vmulesb[1]	000100	vD	vA	vB	1100001000		
vcfux[1]	000100	vD	UIMM	vB	1100001010		0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29	30	31
vspltisb[1]	000100	**v**D	SIMM	0_0000	1100001100			
vpkpx[1]	000100	**v**D	**v**A	**v**B	1100001110			
vaddshs[1]	000100	**v**D	**v**A	**v**B	1101000000			0
vminsh[1]	000100	**v**D	**v**A	**v**B	1101000010			
vsrah[1]	000100	**v**D	**v**A	**v**B	1101000100			
vcmpgtsh*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1101000110		
vmulesh[1]	000100	**v**D	**v**A	**v**B	1101001000			
vcfsx[1]	000100	**v**D	UIMM	**v**B	01101001010			
vspltish[1]	000100	**v**D	SIMM	0_0000	1101001100			
vupkhpx[1]	000100	**v**D	00_000	**v**B	1101001110			
vaddsws[1]	000100	**v**D	**v**A	**v**B	1110000000			0
vminsw[1]	000100	**v**D	**v**A	**v**B	1110000010			
vsraw[1]	000100	**v**D	**v**A	**v**B	1110000100			
vcmpgtsw*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1110000110		
vctuxs[1]	000100	**v**D	UIMM	**v**B	1110001010			
vspltisw[1]	000100	**v**D	SIMM	0_0000	1110001100			
vcmpbfp*x*[1]	000100	**v**D	**v**A	**v**B	Rc	1111000110		
vctsxs[1]	000100	**v**D	UIMM	**v**B	1111001010			
vupklpx[1]	000100	**v**D	00_000	**v**B	1111001110			
vsububm[1]	000100	**v**D	**v**A	**v**B	10000000000			
vavgub[1]	000100	**v**D	**v**A	**v**B	10000000010			0
vand[1]	000100	**v**D	**v**A	**v**B	10000000100			0
vmaxfp[1]	000100	**v**D	**v**A	**v**B	10000001010			
vslo[1]	000100	**v**D	**v**A	**v**B	10000001100			
vsubuhm[1]	000100	**v**D	**v**A	**v**B	10001000000			
vavguh[1]	000100	**v**D	**v**A	**v**B	10001000010			0
vandc[1]	000100	**v**D	**v**A	**v**B	10001000100			0
vminfp[1]	000100	**v**D	**v**A	**v**B	10001001010			
vsro[1]	000100	**v**D	**v**A	**v**B	10001001100			
vsubuwm[1]	000100	**v**D	**v**A	**v**B	10010000000			
vavguw[1]	000100	**v**D	**v**A	**v**B	10010000010			0
vor[1]	000100	**v**D	**v**A	**v**B	10010000100			
vxor[1]	000100	**v**D	**v**A	**v**B	10011000100			

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8	9	10	11 12 13	14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29	30	31
vavgsb[1]	000100	vD			vA		vB	10100000010		0
vnor[1]	000100	vD			vA		vB	10100000100		
vavgsh[1]	000100	vD			vA		vB	10101000010		0
vsubcuw[1]	000100	vD			vA		vB	10110000000		
vavgsw[1]	000100	vD			vA		vB	10110000010		0
vsububs[1]	000100	vD			vA		vB	11000000000		
mfvscr[1]	000100	vD			00_000		0_0000	11000000100		0
vsum4ubs[1]	000100	vD			vA		vB	11000001000		
vsubuhs[1]	000100	vD			vA		vB	11001000000		
mtvscr[1]	000100	000_00			00_000		vB	11001000100		0
vsum4shs[1]	000100	vD			vA		vB	11001001000		
vsubuws[1]	000100	vD			vA		vB	11010000000		
vsum2sws[1]	000100	vD			vA		vB	11010001000		
vsubsbs[1]	000100	vD			vA		vB	11100000000		
vsum4sbs[1]	000100	vD			vA		vB	11100001000		
vsubshs[1]	000100	vD			vA		vB	11101000000		
vsubsws[1]	000100	vD			vA		vB	11110000000		
vsumsws[1]	000100	vD			vA		vB	11110001000		
mulli	000111	D			A		SIMM			
subfic	001000	D			A		SIMM			
cmpli	001010	crfD	0	L	A		UIMM			
cmpi	001011	crfD	0	L	A		SIMM			
addic	001100	D			A		SIMM			
addic.	001101	D			A		SIMM			
addi	001110	D			A		SIMM			
addis	001111	D			A		SIMM			
bc*x*	010000	BO			BI		BD		AA	LK
sc	010001	000_0000_0000_0000_0000_0000_00							1	0
b*x*	010010	LI							AA	LK
mcrf	010011	crfD	00		crfS	00	0_0000	000000000		0
bclr*x*	010011	BO			BI		0_0000	0000010000		LK
crnor	010011	crbD			crbA		crbB	0000100001		0
rfi[2]	010011	000_00			00_000		0_0000	0000110010		0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8	9	10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30	31
crandc	010011	crbD			crbA	crbB	0010000001			0
isync	010011	000_00			00_000	0_0000	0010010110			0
crxor	010011	crbD			crbA	crbB	0011000001			0
crnand	010011	crbD			crbA	crbB	0011100001			0
crand	010011	crbD			crbA	crbB	0100000001			0
creqv	010011	crbD			crbA	crbB	0100100001			0
crorc	010011	crbD			crbA	crbB	0110100001			0
cror	010011	crbD			crbA	crbB	0111000001			0
bcctr*x*	010011	BO			BI	0_0000	1000010000			LK
rlwimi*x*	010100	S			A	SH	MB		ME	Rc
rlwinm*x*	010101	S			A	SH	MB		ME	Rc
rlwnm*x*	010111	S			A	B	MB		ME	Rc
ori	011000	S			A	UIMM				
oris	011001	S			A	UIMM				
xori	011010	S			A	UIMM				
xoris	011011	S			A	UIMM				
andi.	011100	S			A	UIMM				
andis.	011101	S			A	UIMM				
cmp	011111	crfD	0	L	A	B	0000000000			0
tw	011111	TO			A	B	0000000100			0
lvsl[1]	011111	**vD**			A	B	0000000110			0
lvebx[1]	011111	**vD**			A	B	0000000111			0
subfc*x*	011111	D			A	B	OE	000001000		Rc
addc*x*	011111	D			A	B	OE	000001010		Rc
mulhwu*x*	011111	D			A	B	0	000001011		Rc
mfcr	011111	D			00_000	0_0000	0000010011			0
lwarx	011111	D			A	B	0000010100			0
lwzx	011111	D			A	B	0000010111			0
slw*x*	011111	S			A	B	0000011000			Rc
cntlzw*x*	011111	S			A	0_0000	0000011010			Rc
and*x*	011111	S			A	B	000011100			Rc
cmpl	011111	crfD	0	L	A	B	0000100000			0
lvsr[1]	011111	**vD**			A	B	0000100110			0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lvehx[1]	011111	vD	A	B	0000100111	0
subf*x*	011111	D	A	B	OE 000101000	Rc
dcbst	011111	000_00	A	B	0000110110	0
lwzux	011111	D	A	B	0000110111	0
andc*x*	011111	S	A	B	000111100	Rc
lvewx[1]	011111	vD	A	B	0001000111	0
mulhw*x*	011111	D	A	B	0 001001011	Rc
mfmsr[2]	011111	D	00_000	0_0000	0001010011	0
dcbf	011111	000_00	A	B	0001010110	0
lbzx	011111	D	A	B	0001010111	0
lvx[1]	011111	vD	A	B	0001100111	0
neg*x*	011111	D	A	0_0000	OE 001101000	Rc
lbzux	011111	D	A	B	0001110111	0
nor*x*	011111	S	A	B	0001111100	Rc
stvebx[1]	011111	vS	A	B	0010000111	0
subfe*x*	011111	D	A	B	OE 010001000	Rc
adde*x*	011111	D	A	B	OE 010001010	Rc
mtcrf	011111	S	0 CRM	0	0010010000	0
mtmsr[2]	011111	S	00_000	0_0000	0010010010	0
stwcx.	011111	S	A	B	10010110	1
stwx	011111	S	A	B	10010111	0
stvehx[1]	011111	vS	A	B	0010100111	0
stwux	011111	S	A	B	10110111	0
stvewx[1]	011111	vS	A	B	0011000111	0
subfze*x*	011111	D	A	0_0000	OE 011001000	Rc
addze*x*	011111	D	A	0_0000	OE 11001010	Rc
mtsr[2]	011111	S	0 SR	0_0000	0011010010	0
stbx	011111	S	A	B	0011010111	0
stvx[1]	011111	vS	A	B	0011100111	0
subfme*x*	011111	D	A	0_0000	OE 011101000	Rc
addme*x*	011111	D	A	0_0000	OE 11101010	Rc
mullw*x*	011111	D	A	B	OE 011101011	Rc
mtsrin[2]	011111	S	00_000	B	0011110010	0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6	7 8	9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
dcbtst	011111	000_00			A	B	0011110110		0
stbux	011111	S			A	B	0011110111		0
add*x*	011111	D			A	B	OE	100 001 010	Rc
dcbt	011111	000_00			A	B	0100010110		0
lhzx	011111	D			A	B	0100010111		0
eqv*x*	011111	S			A	B	0100011100		Rc
tlbie[2, 3]	011111	000_00			00_000	B	0100110010		0
eciwx[3]	011111	D			A	B	0100110110		0
lhzux	011111	D			A	B	0100110111		0
xor*x*	011111	S			A	B	0100111100		Rc
mfspr[4]	011111	D			spr		0101010011		0
dst[1]	011111	T	00	STRM	A	B	0101010110		0
dstt[1]	011111	1	00	STRM	A	B	0101010110		0
lhax	011111	D			A	B	0101010111		0
lvxl[1]	011111	**v**D			A	B	0101100111		0
tlbia[5]	011111	000_00			00_000	0_0000	0101110010		0
mftb	011111	D			tbr		0101110011		0
dstst[1]	011111	T	00	STRM	A	B	0101110110		0
dststt[1]	011111	1	00	STRM	A	B	0101110110		0
lhaux	011111	D			A	B	0101110111		0
sthx	011111	S			A	B	110010111		0
orc*x*	011111	S			A	B	0110011100		Rc
ecowx[3]	011111	S			A	B	0110110110		0
sthux	011111	S			A	B	110110111		0
or*x*	011111	S			A	B	0110111100		Rc
divwu*x*	011111	D			A	B	OE	1 1100 1011	Rc
mtspr[4]	011111	S			spr		0111010011		0
dcbi[2]	011111	000_00			A	B	0111010110		0
nand*x*	011111	S			A	B	0111011100		Rc
stvxl[1]	011111	**v**S			A	B	0111100111		0
divw*x*	011111	D			A	B	OE	1 1110 1011	Rc
mcrxr	011111	crfD		00	00_000	0_0000	1000000000		0
lswx[6]	011111	D			A	B	1000010101		0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lwbrx	011111	D	A	B	1000010110	0
lfsx	011111	D	A	B	1000010111	0
srw*x*	011111	S	A	B	1000011000	Rc
tlbsync[2, 3]	011111	000_00	00_000	0_0000	1000110110	0
lfsux	011111	D	A	B	1000110111	0
mfsr[2]	011111	D	0 SR	0_0000	1001010011	0
lswi[6]	011111	D	A	NB	1001010101	0
sync	011111	000_00	00_000	0_0000	1001010110	0
lfdx	011111	D	A	B	1001010111	0
lfdux	011111	D	A	B	1001110111	0
mfsrin[2]	011111	D	00_000	B	1010010011	0
stswx[6]	011111	S	A	B	1010010101	0
stwbrx	011111	S	A	B	1010010110	0
stfsx	011111	S	A	B	1010010111	0
stfsux	011111	S	A	B	1010110111	0
stswi[6]	011111	S	A	NB	1011010101	0
stfdx	011111	S	A	B	1011010111	0
dcba[3]	011111	000_00	A	B	1011110110	0
stfdux	011111	S	A	B	1011110111	0
lhbrx	011111	D	A	B	1100010110	0
sraw*x*	011111	S	A	B	1100011000	Rc
dss[1]	011111	A 00 STRM	00_000	0_0000	1100110110	0
dssall[1]	011111	A 00 STRM	00_000	0_0000	1100110110	0
srawi*x*	011111	S	A	SH	1100011000	Rc
eieio	011111	000_00	00_000	0_0000	1101010110	0
sthbrx	011111	S	A	B	1110010110	0
extsh*x*	011111	S	A	0_0000	1110011010	Rc
extsb*x*	011111	S	A	0_0000	1110111010	Rc
icbi	011111	000_00	A	B	1111010110	0
stfiwx[3]	011111	S	A	B	1111010111	0
dcbz	011111	000_00	A	B	1111110110	0
lwz	100000	D	A	d		
lwzu	100001	D	A	d		

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
lbz	100010	D		A	d			
lbzu	100011	D		A	d			
stw	100100	S		A	d			
stwu	100101	S		A	d			
stb	100110	S		A	d			
stbu	100111	S		A	d			
lhz	101000	D		A	d			
lhzu	101001	D		A	d			
lha	101010	D		A	d			
lhau	101011	D		A	d			
sth	101100	S		A	d			
sthu	101101	S		A	d			
lmw[6]	101110	D		A	d			
stmw[6]	101111	S		A	d			
lfs	110000	D		A	d			
lfsu	110001	D		A	d			
lfd	110010	D		A	d			
lfdu	110011	D		A	d			
stfs	110100	S		A	d			
stfsu	110101	S		A	d			
stfd	110110	S		A	d			
stfdu	110111	S		A	d			
fdivs*x*	111011	D		A	B	0000_0	1 0010	Rc
fsubs*x*	111011	D		A	B	0000_0	1 0100	Rc
fadds*x*	111011	D		A	B	0000_0	1 0101	Rc
fsqrts*x*[5]	111011	D		00_000	B	0000_0	1 0110	Rc
fres*x*[3]	111011	D		00_000	B	0000_0	1 1000	Rc
fmuls*x*	111011	D		A	0_0000	C	1 1001	Rc
fmsubs*x*	111011	D		A	B	C	1 1100	Rc
fmadds*x*	111011	D		A	B	C	1 1101	Rc
fnmsubs*x*	111011	D		A	B	C	1 1110	Rc
fnmadds*x*	111011	D		A	B	C	1 1111	Rc
fcmpu	111111	crfD	00	A	B	0000000000		0

Table A-4. Instructions by Primary and Secondary Opcode (Bin) (Continued)

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
frsp*x*	111111	D	00_000	B	0000001100		Rc
fctiw*x*	111111	D	00_000	B	0000001110		Rc
fctiwz*x*	111111	D	00_000	B	0000001111		Rc
fdiv*x*	111111	D	A	B	0000_0	1 0010	Rc
fsub*x*	111111	D	A	B	0000_0	1 0100	Rc
fadd*x*	111111	D	A	B	0000_0	1 0101	Rc
fsqrt*x* [5]	111111	D	00_000	B	0000_0	1 0110	Rc
fsel*x*[3]	111111	D	A	B	C	1 0111	Rc
fmul*x*	111111	D	A	0_0000	C	1 1001	Rc
frsqrte*x* [3]	111111	D	00_000	B	0000_0	1 1010	Rc
fmsub*x*	111111	D	A	B	C	1 1100	Rc
fmadd*x*	111111	D	A	B	C	1 1101	Rc
fnmsub*x*	111111	D	A	B	C	1 1110	Rc
fnmadd*x*	111111	D	A	B	C	1 1111	Rc
fcmpo	111111	crfD 00	A	B	0000100000		0
mtfsb1*x*	111111	crbD	00_000	0_0000	0000100110		Rc
fneg*x*	111111	D	00_000	B	0000101000		Rc
mcrfs	111111	crfD 00	crfS 00	0_0000	001000000		0
mtfsb0*x*	111111	crbD	00_000	0_0000	0001000110		Rc
fmr*x*	111111	D	00_000	B	0001001000		Rc
mtfsfi*x*	111111	crfD 00	00_000	IMM 0	0010000110		Rc
fnabs*x*	111111	D	00_000	B	0010001000		Rc
fabs*x*	111111	D	00_000	B	0100001000		Rc
mffs*x*	111111	D	00_000	0_0000	1001000111		Rc
mtfsf*x*	111111	0 FM	0	B	1011000111		Rc

[1] **AltiVec technology-specific instruction**

[2] Supervisor-level instruction

[3] Optional to the architecture but implemented by the MPC7410

[4] Supervisor- and user-level instruction

[5] Optional instruction not implemented by the MPC7410

[6] Load/store string/multiple instruction

A.5 Instructions Grouped by Functional Categories

Table A-5 through Table A-45 list the MPC7410 instructions grouped by function.

Key: ▨ Reserved bits

Table A-5. Integer Arithmetic Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
add*x*	31	D	A	B	OE	266	Rc
addc*x*	31	D	A	B	OE	10	Rc
adde*x*	31	D	A	B	OE	138	Rc
addi	14	D	A	SIMM			
addic	12	D	A	SIMM			
addic.	13	D	A	SIMM			
addis	15	D	A	SIMM			
addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	Rc
mulli	07	D	A	SIMM			
mullw*x*	31	D	A	B	OE	235	Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104	Rc
subf*x*	31	D	A	B	OE	40	Rc
subfc*x*	31	D	A	B	OE	8	Rc
subfic*x*	08	D	A	SIMM			
subfe*x*	31	D	A	B	OE	136	Rc
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc

Table A-6. Integer Compare Instructions

Name	0 5	6 7 8	9	10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
cmp	31	crfD	0	L	A	B	0 0 0 0 0 0 0 0 0 0	0
cmpi	11	crfD	0	L	A	SIMM		
cmpl	31	crfD	0	L	A	B	32	0
cmpli	10	crfD	0	L	A	UIMM		

Table A-7. Integer Logical Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
and*x*	31	S	A	B	28	Rc
andc*x*	31	S	A	B	60	Rc
andi.	28	S	A	UIMM		
andis.	29	S	A	UIMM		
cntlzw*x*	31	S	A	0 0 0 0 0	26	Rc
eqv*x*	31	S	A	B	284	Rc
extsb*x*	31	S	A	0 0 0 0 0	954	Rc
extsh*x*	31	S	A	0 0 0 0 0	922	Rc
nand*x*	31	S	A	B	476	Rc
nor*x*	31	S	A	B	124	Rc
or*x*	31	S	A	B	444	Rc
orc*x*	31	S	A	B	412	Rc
ori	24	S	A	UIMM		
oris	25	S	A	UIMM		
vand[1]	04	**v**D	**v**A	**v**B	1028	0
vandc [1]	04	**v**D	**v**A	**v**B	1092	0
vnor [1]	04	**v**D	**v**A	**v**B	1284	
vor [1]	04	**v**D	**v**A	**v**B	1156	
vxor [1]	04	D	A	B	1220	
xor*x*	31	S	A	B	316	Rc
xori	26	S	A	UIMM		
xoris	27	S	A	UIMM		

[1] AltiVec technology-specific instruction

Table A-8. Integer Rotate Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
rlwimi*x*	22	S	A	SH	MB	ME	Rc
rlwinm*x*	20	S	A	SH	MB	ME	Rc
rlwnm*x*	21	S	A	SH	MB	ME	Rc

Table A-9. Integer Shift Instruction

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
slw*x*	31	S	A	B	24	Rc
sraw*x*	31	S	A	B	792	Rc
srawi*x*	31	S	A	SH	824	Rc
srw*x*	31	S	A	B	536	Rc

Table A-10. Floating-Point Arithmetic Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fadd*x*	63	D	A	B	0 0 0 0 0	21	Rc
fadds*x*	59	D	A	B	0 0 0 0 0	21	Rc
fdiv*x*	63	D	A	B	0 0 0 0 0	18	Rc
fdivs*x*	59	D	A	B	0 0 0 0 0	18	Rc
fmul*x*	63	D	A	0 0 0 0 0	C	25	Rc
fmuls*x*	59	D	A	0 0 0 0 0	C	25	Rc
fres*x* [1]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte*x* [1]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsub*x*	63	D	A	B	0 0 0 0 0	20	Rc
fsubs*x*	59	D	A	B	0 0 0 0 0	20	Rc
fsel*x*	63	D	A	B	C	23	Rc
fsqrt*x* [2]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x* [2]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
vaddfp[3]	04	**v**D	**v**A	**v**B	10		0
vmaxfp [3]	04	**v**D	**v**A	**v**B	1034		
vminfp [3]	04	**v**D	**v**A	**v**B	1098		
vsubfp [3]	04	**v**D	**v**A	**v**B	74		

[1] Optional to the architecture but implemented by the MPC7410

[2] Optional instruction not implemented by the MPC7410

[3] AltiVec technology-specific instruction

Table A-11. Floating-Point Multiply-Add Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fmadd*x*	63	D	A	B	C	29	Rc
fmadds*x*	59	D	A	B	C	29	Rc
fmsub*x*	63	D	A	B	C	28	Rc

Table A-11. Floating-Point Multiply-Add Instructions (Continued)

fmsubs*x*	59	D	A	B	C	28	Rc
fnmadd*x*	63	D	A	B	C	31	Rc
fnmadds*x*	59	D	A	B	C	31	Rc
fnmsub*x*	63	D	A	B	C	30	Rc
fnmsubs*x*	59	D	A	B	C	30	Rc
vmaddfp[1]	04	**v**D	**v**A	**v**B	**v**C	46	
vnmsubfp [1]	04	**v**D	**v**A	**v**B	**v**C	47	

[1] AltiVec technology-specific instruction

Table A-12. Floating-Point Rounding and Conversion Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
fctiw*x*	63	D	0 0 0 0 0	B	14	Rc
fctiwz*x*	63	D	0 0 0 0 0	B	15	Rc
frsp*x*	63	D	0 0 0 0 0	B	12	Rc
vcfsx[1]	04	**v**D	UIMM	**v**B	842	
vcfux [1]	04	**v**D	UIMM	**v**B	778	0
vctsxs [1]	04	**v**D	UIMM	**v**B	970	
vctuxs [1]	04	**v**D	UIMM	**v**B	906	
vrfim [1]	04	**v**D	0 0 0 0 0	**v**B	714	
vrfin [1]	04	**v**D	0 0 0 0 0	**v**B	522	
vrfip [1]	04	**v**D	0 0 0 0 0	**v**B	650	
vrfiz [1]	04	**v**D	0 0 0 0 0	**v**B	586	

[1] AltiVec technology-specific instruction

Table A-13. Floating-Point Compare Instructions

Name	0 5	6 7 8	9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
fcmpo	63	crfD	0 0	A	B	32		0
fcmpu	63	crfD	0 0	A	B	0		0
vcmpbfp*x*[1]	04	**v**D		**v**A	**v**B	Rc	966	
vcmpeqfp*x* [1]	04	**v**D		**v**A	**v**B	Rc	198	
vcmpgefp*x* [1]	04	**v**D		**v**A	**v**B	Rc	454	
vcmpgtfp*x* [1]	04	**v**D		**v**A	**v**B	Rc	710	

[1] AltiVec technology-specific instruction

Table A-14. Floating-Point Status and Control Register Instructions

Name	0 5	6	7 8	9 10	11 12 13	14	15	16 17 18 19	20	21 22 23 24 25 26 27 28 29 30	31
mcrfs	63	crfD		0 0	crfS	0 0		0 0 0 0 0		64	0
mffs*x*	63	D			0 0 0 0 0			0 0 0 0 0		583	Rc
mtfsb0*x*	63	crbD			0 0 0 0 0			0 0 0 0 0		70	Rc
mtfsb1*x*	63	crbD			0 0 0 0 0			0 0 0 0 0		38	Rc
mtfsf*x*	31	0	FM				0	B		711	Rc
mtfsfi*x*	63	crfD		0 0	0 0 0 0 0			IMM	0	134	Rc

Table A-15. Integer Load Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lbz	34	D	A	d		
lbzu	35	D	A	d		
lbzux	31	D	A	B	119	0
lbzx	31	D	A	B	87	0
lha	42	D	A	d		
lhau	43	D	A	d		
lhaux	31	D	A	B	375	0
lhax	31	D	A	B	343	0
lhz	40	D	A	d		
lhzu	41	D	A	d		
lhzux	31	D	A	B	311	0
lhzx	31	D	A	B	279	0
lvebx[1]	31	**v**D	A	B	7	0
lvehx [1]	31	**v**D	A	B	39	0
lvewx [1]	31	**v**D	A	B	71	0
lvx [1]	31	**v**D	A	B	103	0
lvxl [1]	31	**v**D	A	B	359	0
lwz	32	D	A	d		
lwzu	33	D	A	d		
lwzux	31	D	A	B	55	0
lwzx	31	D	A	B	23	0

[1] AltiVec technology-specific instruction

Table A-16. Integer Store Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stb	38	S	A	d		
stbu	39	S	A	d		
stbux	31	S	A	B	247	0
stbx	31	S	A	B	215	0
sth	44	S	A	d		
sthu	45	S	A	d		
sthux	31	S	A	B	439	0
sthx	31	S	A	B	407	0
stw	36	S	A	d		
stwu	37	S	A	d		
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0

Table A-17. Integer Load and Store with Byte Reverse Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lhbrx	31	D	A	B	790	0
lwbrx	31	D	A	B	534	0
sthbrx	31	S	A	B	918	0
stwbrx	31	S	A	B	662	0

Table A-18. Integer Load and Store Multiple Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lmw[1]	46	D	A	d		
stmw[1]	47	S	A	d		

[1] Load/store string/multiple instruction

Table A-19. Integer Load and Store String Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lswi[1]	31	D	A	NB	597	0
lswx[1]	31	D	A	B	533	0
stswi[1]	31	S	A	NB	725	0
stswx[1]	31	S	A	B	661	0

[1] Load/store string/multiple instruction

Table A-20. Memory Synchronization Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
eieio	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	854	0
isync	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	150	0
lwarx	31	D	A	B	20	0
stwcx.	31	S	A	B	150	1
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0

Table A-21. Floating-Point Load Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lfd	50	D	A	d		
lfdu	51	D	A	d		
lfdux	31	D	A	B	631	0
lfdx	31	D	A	B	599	0
lfs	48	D	A	d		
lfsu	49	D	A	d		
lfsux	31	D	A	B	567	0
lfsx	31	D	A	B	535	0

Table A-22. Floating-Point Store Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stfd	54	S	A	d		
stfdu	55	S	A	d		
stfdux	31	S	A	B	759	0
stfdx	31	S	A	B	727	0
stfiwx[1]	31	S	A	B	983	0
stfs	52	S	A	d		
stfsu	53	S	A	d		
stfsux	31	S	A	B	695	0
stfsx	31	S	A	B	663	0

[1] Optional to the architecture but implemented by the MPC7410

Table A-23. Floating-Point Move Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
fabs*x*	63	D	0 0 0 0 0	B	264	Rc
fmr*x*	63	D	0 0 0 0 0	B	72	Rc
fnabs*x*	63	D	0 0 0 0 0	B	136	Rc
fneg*x*	63	D	0 0 0 0 0	B	40	Rc

Table A-24. Branch Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29	30	31
b*x*	18	LI				AA	LK
bc*x*	16	BO	BI	BD		AA	LK
bcctr*x*	19	BO	BI	0 0 0 0 0	528		LK
bclr*x*	19	BO	BI	0 0 0 0 0	16		LK

Table A-25. Condition Register Logical Instructions

Name	0 5	6 7 8	9 10	11 12 13	14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
crand	19	crbD		crbA		crbB	257	0
crandc	19	crbD		crbA		crbB	129	0
creqv	19	crbD		crbA		crbB	289	0
crnand	19	crbD		crbA		crbB	225	0
crnor	19	crbD		crbA		crbB	33	0
cror	19	crbD		crbA		crbB	449	0
crorc	19	crbD		crbA		crbB	417	0
crxor	19	crbD		crbA		crbB	193	0
mcrf	19	crfD	0 0	crfS	0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0

Table A-26. System Linkage Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29	30	31
rfi[1]	19	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	50		0
sc	17	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0		1	0

[1] Supervisor-level instruction

Table A-27. Trap Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
tw	31	TO	A	B	4	0
twi	03	TO	A	SIMM		

Table A-28. Processor Control Instructions

Name	0 5	6 7 8	9 10	11	12 13 14 15	16 17 18 19	20	21 22 23 24 25 26 27 28 29 30	31
mcrxr	31	crfS	0 0	0 0 0 0 0		0 0 0 0 0		512	0
mfcr	31	D		0 0 0 0 0		0 0 0 0 0		19	0
mfmsr[1]	31	D		0 0 0 0 0		0 0 0 0 0		83	0
mfspr[2]	31	D		spr				339	0
mftb	31	D		tpr				371	0
mtcrf	31	S		0	CRM		0	144	0
mtmsr[1]	31	S		0 0 0 0 0		0 0 0 0 0		146	0
mtspr[2]	31	D		spr				467	0

[1] Supervisor-level instruction

[2] Supervisor- and user-level instruction

Table A-29. Cache Management Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
dcba[1]	31	0 0 0 0 0	A	B	758	0
dcbf	31	0 0 0 0 0	A	B	86	0
dcbi[2]	31	0 0 0 0 0	A	B	470	0
dcbst	31	0 0 0 0 0	A	B	54	0
dcbt	31	0 0 0 0 0	A	B	278	0
dcbtst	31	0 0 0 0 0	A	B	246	0
dcbz	31	0 0 0 0 0	A	B	1014	0
icbi	31	0 0 0 0 0	A	B	982	0

[1] Optional to the architecture but implemented by the MPC7410

[2] Supervisor-level instruction

Table A-30. Segment Register Manipulation Instructions

Name	0 5	6 7 8 9 10	11	12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
mfsr[1]	31	D	0	SR	0 0 0 0 0	595	0
mfsrin[1]	31	D	0 0 0 0 0		B	659	0

mtsr[1]	31	S	0	SR	0 0 0 0 0	210	0
mtsrin[1]	31	S	0 0 0 0 0		B	242	0

[1] Supervisor-level instruction

Table A-31. Lookaside Buffer Management Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
tlbia[1]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie[2,3]	31	0 0 0 0 0	0 0 0 0 0	B	306	0
tlbsync[2,3]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0

[1] Optional instruction not implemented by the MPC7410

[2] Optional to the architecture but implemented by the MPC7410

[3] Supervisor-level instruction

Table A-32. External Control Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
eciwx[1]	31	D	A	B	310	0
ecowx[1]	31	S	A	B	438	0

[1] Optional to the architecture but implemented by the MPC7410

Table A-33. Vector Integer Arithmetic Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
vaddcuw[1]	04	vD	vA	vB	384	0
vaddsbs [1]	04	vD	vA	vB	768	0
vaddshs [1]	04	vD	vA	vB	832	0
vaddsws [1]	04	vD	vA	vB	896	0
vaddubm [1]	04	vD	vA	vB	0	0
vaddubs [1]	04	vD	vA	vB	512	0
vadduhm [1]	04	vD	vA	vB	64	0
vadduhs [1]	04	vD	vA	vB	576	0
vadduwm [1]	04	vD	vA	vB	128	0
vadduws [1]	04	vD	vA	vB	640	0
vavgsb [1]	04	vD	vA	vB	1282	0
vavgsh [1]	04	vD	vA	vB	1346	0
vavgsw [1]	04	vD	vA	vB	1410	0

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
vavgub [1]	04	vD	vA	vB	1026		0
vavguh [1]	04	vD	vA	vB	1090		0
vavguw [1]	04	vD	vA	vB	1154		0
vmaxsb [1]	04	vD	vA	vB	258		
vmaxsh [1]	04	vD	vA	vB	322		
vmaxsw [1]	04	vD	vA	vB	386		
vmaxub [1]	04	vD	vA	vB	2		
vmaxuh [1]	04	vD	vA	vB	66		
vmaxuw [1]	04	vD	vA	vB	130		
vmhaddshs [1]	04	vD	vA	vB	vC	32	
vmhraddshs [1]	04	vD	vA	vB	vC	33	
vminsb [1]	04	vD	vA	vB	770		
vminsh [1]	04	vD	vA	vB	834		
vminsw [1]	04	vD	vA	vB	898		
vminub [1]	04	vD	vA	vB	514		
vminuh [1]	04	vD	vA	vB	578		
vminuw [1]	04	vD	vA	vB	642		
vmladduhm [1]	04	vD	vA	vB	vC	34	
vmsummbm [1]	04	vD	vA	vB	vC	37	
vmsumshm [1]	04	vD	vA	vB	vC	40	
vmsumshs [1]	04	vD	vA	vB	vC	41	
vmsumubm [1]	04	vD	vA	vB	vC	36	
vmsumuhm [1]	04	vD	vA	vB	vC	38	
vmsumuhs [1]	04	vD	vA	vB	vC	39	
vmulesb [1]	04	vD	vA	vB	776		
vmulesh [1]	04	vD	vA	vB	840		
vmuleub [1]	04	vD	vA	vB	520		
vmuleuh [1]	04	vD	vA	vB	584		
vmulosb [1]	04	vD	vA	vB	264		
vmulosh [1]	04	vD	vA	vB	328		
vmuloub [1]	04	vD	vA	vB	8		
vmulouh [1]	04	vD	vA	vB	72		

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
vsubcuw [1]	04	**v**D	**v**A	**v**B	1408
vsubsbs [1]	04	**v**D	**v**A	**v**B	1792
vsubshs [1]	04	**v**D	**v**A	**v**B	1856
vsubsws [1]	04	**v**D	**v**A	**v**B	1920
vsububm [1]	04	**v**D	**v**A	**v**B	1024
vsububs [1]	04	**v**D	**v**A	**v**B	1536
vsubuhm [1]	04	**v**D	**v**A	**v**B	1088
vsubuhs [1]	04	**v**D	**v**A	**v**B	1600
vsubuwm [1]	04	**v**D	**v**A	**v**B	1152
vsubuws [1]	04	**v**D	**v**A	**v**B	1664
vsumsws [1]	04	**v**D	**v**A	**v**B	1928
vsum2sws [1]	04	D	A	B	1672
vsum4sbs [1]	04	D	A	B	1800
vsum4shs [1]	04	D	A	B	1608
vsum4ubs [1]	04	D	A	B	1544

[1] AltiVec technology-specific instruction

Table A-34. Floating-Point Compare Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30 31
vcmpbfp*x*	04	vD	vA	vB	Rc	966
vcmpeqfp*x*	04	vD	vA	vB	Rc	198
vcmpgefp*x*	04	vD	vA	vB	Rc	454
vcmpgtfp*x*	04	vD	vA	vB	Rc	710

Table A-35. Floating-Point Estimate Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
vexptefp	04	vD	0 0 0 0 0	vB	394
vlogefp	04	vD	0 0 0 0 0	vB	458
vrefp	04	vD	0 0 0 0 0	vB	266
vrsqrtefp	04	vD	0 0 0 0 0	vB	330

Table A-36. Vector Load Instructions Supporting Alignment

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
lvsl	31	vD	A	B	6	0
lvsr	31	vD	A	B	38	0

Table A-37. Integer Store Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
stvebx	31	S	A	B	135	0
stvehx	31	S	A	B	167	0
stvewx	31	S	A	B	199	0
stvx	31	S	A	B	231	0
stvxl	31	S	A	B	487	0

Table A-38. Vector Pack Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
vpkpx	04	vD	vA	vB	782
vpkshss	04	vD	vA	vB	398
vpkshus	04	vD	vA	vB	270
vpkswss	04	vD	vA	vB	462
vpkswus	04	vD	vA	vB	334
vpkuhum	04	vD	vA	vB	14
vpkuhus	04	vD	vA	vB	142
vpkuwum	04	vD	vA	vB	78
vpkuwus	04	vD	vA	vB	206

Table A-39. Vector Unpack Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
vmrghb	04	vD	vA	vB	12
vmrghh	04	vD	vA	vB	76
vmrghw	04	vD	vA	vB	140
vmrglb	04	vD	vA	vB	268
vmrglh	04	vD	vA	vB	332
vupkhpx	04	D	0 0 0 0 0	B	846
vupkhsb	04	D	0 0 0 0 0	B	526
vupkhsh	04	D	0 0 0 0 0	B	590
vupklpx	04	D	0 0 0 0 0	B	974
vupklsb	04	D	0 0 0 0 0	B	654
vupklsh	04	D	0 0 0 0 0	B	718

Table A-40. Vector Splat Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
vspltb	04	vD	UIMM	vB	524
vsplth	04	vD	UIMM	vB	588
vspltisb	04	vD	SIMM	0 0 0 0 0	780
vspltish	04	vD	SIMM	0 0 0 0 0	844
vspltisw	04	vD	SIMM	0 0 0 0 0	908
vspltw	04	vD	UIMM	vB	652

Table A-41. Vector Permute Instruction

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30 31
vperm	04	vD	vA	vB	vC	43

Table A-42. Vector Select Instruction

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30 31
vsel	04	vD	vA	vB	vC	42

Table A-43. Vector Shift Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30 31
vsl	04	vD	vA	vB	452		
vsldoi	04	vD	vA	vB	0	SH	44
vslo	04	vD	vA	vB	1036		
vsro	04	vD	vA	vB	1100		

Table A-44. Move to/from Condition Register Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
mfvscr	04	vD	0 0 0 0 0	0 0 0 0 0	1540	0
mtvscr	04	0 0 0 0 0	0 0 0 0 0	vB	1604	

Table A-45. User-Level Cache Instructions

Name	0 5	6	7 8	9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
dss	31	A	0 0	STRM	0 0 0 0 0	0 0 0 0 0	822	0
dssall	31	A	0 0	STRM	0 0 0 0 0	0 0 0 0 0	822	0
dst	31	T	0 0	STRM	A	B	342	0
dstst	31	T	0 0	STRM	A	B	374	0
dststt	31	1	0 0	STRM	A	B	374	0
dstt	31	1	0 0	STRM	A	B	342	0

A.6 Instructions Sorted by Form

Table A-46 through Table A-61 list the MPC7410 instructions grouped by form.

Key:

▨ Reserved bits

Table A-46. I-Form

OPCD	LI	AA	LK

Table 1:

Specific Instruction

Name	0 5	6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31
b*x*	18	LI	AA	LK

Table 2:

Table A-47. B-Form

OPCD	BO	BI	BD	AA	LK

Table 3:

Specific Instruction

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31
bc*x*	16	BO	BI	BD	AA	LK

Table 4:

Table A-48. SC-Form

OPCD	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0

Table 5:

Specific Instruction

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31
sc	17	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0

Table 6:

Table A-49. D-Form

OPCD	D			A	d
OPCD	D			A	SIMM
OPCD	S			A	d
OPCD	S			A	UIMM
OPCD	crfD	0	L	A	SIMM
OPCD	crfD	0	L	A	UIMM
OPCD	TO			A	SIMM

Table 7:

Specific Instructions

Name	0 5	6 7 8	9	10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
addi	14	D			A	SIMM
addic	12	D			A	SIMM
addic.	13	D			A	SIMM
addis	15	D			A	SIMM
andi.	28	S			A	UIMM
andis.	29	S			A	UIMM
cmpi	11	crfD	0	L	A	SIMM
cmpli	10	crfD	0	L	A	UIMM
lbz	34	D			A	d
lbzu	35	D			A	d
lfd	50	D			A	d
lfdu	51	D			A	d
lfs	48	D			A	d
lfsu	49	D			A	d
lha	42	D			A	d
lhau	43	D			A	d
lhz	40	D			A	d
lhzu	41	D			A	d
lmw[1]	46	D			A	d
lwz	32	D			A	d
lwzu	33	D			A	d
mulli	7	D			A	SIMM

Table 8:

ori	24	S	A	UIMM
oris	25	S	A	UIMM
stb	38	S	A	d
stbu	39	S	A	d
stfd	54	S	A	d
stfdu	55	S	A	d
stfs	52	S	A	d
stfsu	53	S	A	d
sth	44	S	A	d
sthu	45	S	A	d
stmw[1]	47	S	A	d
stw	36	S	A	d
stwu	37	S	A	d
subfic	08	D	A	SIMM
twi	03	TO	A	SIMM
xori	26	S	A	UIMM
xoris	27	S	A	UIMM

Table 8:

[1] Load/store string/multiple instruction

Table A-50. X-Form

<table>
<tr><td>OPCD</td><td colspan="4">D</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">D</td><td colspan="3">A</td><td colspan="2">NB</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">D</td><td colspan="3">0 0 0 0 0</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">D</td><td colspan="3">0 0 0 0 0</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">D</td><td>0</td><td colspan="2">SR</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>Rc</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>1</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">A</td><td colspan="2">NB</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">A</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>Rc</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">0 0 0 0 0</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">0 0 0 0 0</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td>0</td><td colspan="2">SR</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">S</td><td colspan="3">A</td><td colspan="2">SH</td><td>XO</td><td>Rc</td></tr>
<tr><td>OPCD</td><td colspan="2">crfD</td><td>0</td><td>L</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="2">crfD</td><td colspan="2">0 0</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="2">crfD</td><td colspan="2">0 0</td><td colspan="2">crfS</td><td>0 0</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="2">crfD</td><td colspan="2">0 0</td><td colspan="3">0 0 0 0 0</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="2">crfD</td><td colspan="2">0 0</td><td colspan="3">0 0 0 0 0</td><td>IMM</td><td>0</td><td>XO</td><td>Rc</td></tr>
<tr><td>OPCD</td><td colspan="4">TO</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">D</td><td colspan="3">0 0 0 0 0</td><td colspan="2">B</td><td>XO</td><td>Rc</td></tr>
<tr><td>OPCD</td><td colspan="4">D</td><td colspan="3">0 0 0 0 0</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>Rc</td></tr>
<tr><td>OPCD</td><td colspan="4">crbD</td><td colspan="3">0 0 0 0 0</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>Rc</td></tr>
<tr><td>OPCD</td><td colspan="4">0 0 0 0 0</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">0 0 0 0 0</td><td colspan="3">0 0 0 0 0</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4">0 0 0 0 0</td><td colspan="3">0 0 0 0 0</td><td colspan="2">0 0 0 0 0</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4"><b>v</b>D</td><td colspan="3"><b>v</b>A</td><td colspan="2"><b>v</b>B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td colspan="4"><b>v</b>S</td><td colspan="3"><b>v</b>A</td><td colspan="2"><b>v</b>B</td><td>XO</td><td>0</td></tr>
<tr><td>OPCD</td><td>T</td><td>0 0</td><td colspan="2">STRM</td><td colspan="3">A</td><td colspan="2">B</td><td>XO</td><td>0</td></tr>
</table>

Table 9:

Specific Instructions

Name 0 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Table 10:

and*x*	31	S			A	B	28	Rc
andc*x*	31	S			A	B	60	Rc
cmp	31	crfD	0	L	A	B	0	0
cmpl	31	crfD	0	L	A	B	32	0
cntlzw*x*	31	S			A	0 0 0 0 0	26	Rc
dcba[1]	31	0 0 0 0 0			A	B	758	0
dcbf	31	0 0 0 0 0			A	B	86	0
dcbi[2]	31	0 0 0 0 0			A	B	470	0
dcbst	31	0 0 0 0 0			A	B	54	0
dcbt	31	0 0 0 0 0			A	B	278	0
dcbtst	31	0 0 0 0 0			A	B	246	0
dcbz	31	0 0 0 0 0			A	B	1014	0
dst	31	T	0 0	STRM	A	B	342	0
dstt[3]	31	1	0 0	STRM	A	B	342	0
dstst[3]	31	T	0 0	STRM	A	B	374	0
dststt[3]	31	1	0 0	STRM	A	B	374	0
dss[3]	31	A	0 0	STRM	0 0 0 0 0	0 0 0 0 0	822	0
dssall[3]	31	A	0 0	STRM	0 0 0 0 0	0 0 0 0 0	822	0
eciwx[1]	31	D			A	B	310	0
ecowx[1]	31	S			A	B	438	0
eieio	31	0 0 0 0 0			0 0 0 0 0	0 0 0 0 0	854	0
eqv*x*	31	S			A	B	284	Rc
extsb*x*	31	S			A	0 0 0 0 0	954	Rc
extsh*x*	31	S			A	0 0 0 0 0	922	Rc
fabs*x*	63	D			0 0 0 0 0	B	264	Rc
fcmpo	63	crfD	0 0		A	B	32	0
fcmpu	63	crfD	0 0		A	B	0	0
fctiw*x*	63	D			0 0 0 0 0	B	14	Rc
fctiwz*x*	63	D			0 0 0 0 0	B	15	Rc
fmr*x*	63	D			0 0 0 0 0	B	72	Rc
fnabs*x*	63	D			0 0 0 0 0	B	136	Rc
fneg*x*	63	D			0 0 0 0 0	B	40	Rc
frsp*x*	63	D			0 0 0 0 0	B	12	Rc
icbi	31	0 0 0 0 0			A	B	982	0

Table 10:

lbzux	31	D		A		B		119	0
lbzx	31	D		A		B		87	0
lfdux	31	D		A		B		631	0
lfdx	31	D		A		B		599	0
lfsux	31	D		A		B		567	0
lfsx	31	D		A		B		535	0
lhaux	31	D		A		B		375	0
lhax	31	D		A		B		343	0
lhbrx	31	D		A		B		790	0
lhzux	31	D		A		B		311	0
lhzx	31	D		A		B		279	0
lswi[4]	31	D		A		NB		597	0
lswx[4]	31	D		A		B		533	0
lvebx[3]	31	**v**D		**v**A		**v**B		7	0
lvehx[3]	31	**v**D		A		B		39	0
lvewx[3]	31	**v**D		A		B		71	0
lvsl[3]	31	**v**D		A		B		6	0
lvsr[3]	31	**v**D		A		B		38	0
lvx[3]	31	**v**D		A		B		103	0
lvxl[3]	31	**v**D		A		B		359	0
lwarx	31	D		A		B		20	0
lwbrx	31	D		A		B		534	0
lwzux	31	D		A		B		55	0
lwzx	31	D		A		B		23	0
mcrfs	63	crfD	0 0	crfS	0 0	0 0 0 0 0		64	0
mcrxr	31	crfD	0 0	0 0 0 0 0		0 0 0 0 0		512	0
mfcr	31	D		0 0 0 0 0		0 0 0 0 0		19	0
mffs*x*	63	D		0 0 0 0 0		0 0 0 0 0		583	Rc
mfmsr[2]	31	D		0 0 0 0 0		0 0 0 0 0		83	0
mfsr[2]	31	D		0	SR	0 0 0 0 0		595	0
mfsrin[2]	31	D		0 0 0 0 0		B		659	0
mtfsb0*x*	63	crbD		0 0 0 0 0		0 0 0 0 0		70	Rc
mtfsb1*x*	63	crfD		0 0 0 0 0		0 0 0 0 0		38	Rc
mtfsfi*x*	63	crbD	0 0	0 0 0 0 0		IMM	0	134	Rc

Table 10:

mtmsr[2]	31	S	0 0 0 0 0	0 0 0 0 0	146	0
mtsr[2]	31	S	0 SR	0 0 0 0 0	210	0
nand*x*	31	S	A	B	476	Rc
nor*x*	31	S	A	B	124	Rc
or*x*	31	S	A	B	444	Rc
orc*x*	31	S	A	B	412	Rc
slw*x*	31	S	A	B	24	Rc
sraw*x*	31	S	A	B	792	Rc
srawi*x*	31	S	A	SH	824	Rc
srw*x*	31	S	A	B	536	Rc
stbux	31	S	A	B	247	0
stbx	31	S	A	B	215	0
stfdux	31	S	A	B	759	0
stfdx	31	S	A	B	727	0
stfiwx[1]	31	S	A	B	983	0
stfsux	31	S	A	B	695	0
stfsx	31	S	A	B	663	0
sthbrx	31	S	A	B	918	0
sthux	31	S	A	B	439	0
sthx	31	S	A	B	407	0
stswi[4]	31	S	A	NB	725	0
stswx[4]	31	S	A	B	661	0
stvebx[3]	31	vS	A	B	135	0
stvehx[3]	31	vS	A	B	167	0
stvewx[3]	31	vS	A	B	199	0
stvx[3]	31	vS	A	B	231	0
stvxl[3]	31	vS	A	B	487	0
stwbrx[3]	31	S	A	B	662	0
stwcx.	31	S	A	B	150	1
stwux	31	S	A	B	183	0
stwx	31	S	A	B	151	0
sync	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	598	0
tlbia[5]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	370	0
tlbie[2]	31	0 0 0 0 0	0 0 0 0 0	B	306	0

Table 10:

tlbsync[2]	31	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	566	0
tw	31	TO	A	B	4	0
xor*x*	31	S	A	B	316	Rc

Table 10:

[1] Optional to the architecture but implemented by the MPC7410

[2] Supervisor-level instruction

[3] Altivec technology-specific instruction

[4] Load/store string/multiple instruction

[5] Optional instruction not implemented by the MPC7410

Table A-51. XL-Form

OPCD	BO		BI		0 0 0 0 0	XO	LK
OPCD	crbD		crbA		crbB	XO	0
OPCD	crfD	0 0	crfS	0 0	0 0 0 0 0	XO	0
OPCD	0 0 0 0 0		0 0 0 0 0		0 0 0 0 0	XO	0

Table 11:

Specific Instructions

Name	0 5	6 7 8	9 10	11 12 13	14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
bcctr*x*	19	BO		BI		0 0 0 0 0	528	LK
bclr*x*	19	BO		BI		0 0 0 0 0	16	LK
crand	19	crbD		crbA		crbB	257	0
crandc	19	crbD		crbA		crbB	129	0
creqv	19	crbD		crbA		crbB	289	0
crnand	19	crbD		crbA		crbB	225	0
crnor	19	crbD		crbA		crbB	33	0
cror	19	crbD		crbA		crbB	449	0
crorc	19	crbD		crbA		crbB	417	0
crxor	19	crbD		crbA		crbB	193	0
isync	19	0 0 0 0 0		0 0 0 0 0		0 0 0 0 0	150	0
mcrf	19	crfD	0 0	crfS	0 0	0 0 0 0 0	0	0
rfi[1]	19	0 0 0 0 0		0 0 0 0 0		0 0 0 0 0	50	0

Table 12:

[1] Supervisor-level instruction

Table A-52. XFX-Form

OPCD	D	spr			XO	0
OPCD	D	0	CRM	0	XO	0
OPCD	S	spr			XO	0
OPCD	D	tbr			XO	0

Table 13:

Specific Instructions

Name	0 5	6 7 8 9 10	11	12 13 14 15 16 17 18 19	20	21 22 23 24 25 26 27 28 29 30	31
mfspr[1]	31	D	spr			339	0
mftb	31	D	tbr			371	0
mtcrf	31	S	0	CRM	0	144	0
mtspr[1]	31	D	spr			467	0

Table 14:

[1] Supervisor- and user-level instruction

Table A-53. XFL-Form

OPCD	0	FM	0	B	XO	Rc

Table 15:

Specific Instructions

Name	0 5	6	7 8 9 10 11 12 13 14	15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30	31
mtfsf*x*	63	0	FM	0	B	711	Rc

Table 16:

Table A-54. XO-Form

OPCD	D	A	B	OE	XO	Rc
OPCD	D	A	B	0	XO	Rc
OPCD	D	A	0 0 0 0 0	OE	XO	Rc

Table 17:

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30	31
add*x*	31	D	A	B	OE	266	Rc

Table 18:

addc*x*	31	D	A	B	OE	10	Rc
adde*x*	31	D	A	B	OE	138	Rc
addme*x*	31	D	A	0 0 0 0 0	OE	234	Rc
addze*x*	31	D	A	0 0 0 0 0	OE	202	Rc
divw*x*	31	D	A	B	OE	491	Rc
divwu*x*	31	D	A	B	OE	459	Rc
mulhw*x*	31	D	A	B	0	75	Rc
mulhwu*x*	31	D	A	B	0	11	Rc
mullw*x*	31	D	A	B	OE	235	Rc
neg*x*	31	D	A	0 0 0 0 0	OE	104	Rc
subf*x*	31	D	A	B	OE	40	Rc
subfc*x*	31	D	A	B	OE	8	Rc
subfe*x*	31	D	A	B	OE	136	Rc
subfme*x*	31	D	A	0 0 0 0 0	OE	232	Rc
subfze*x*	31	D	A	0 0 0 0 0	OE	200	Rc

Table 18:

OPCD	D	A	B	0 0 0 0 0	XO	Rc
OPCD	D	A	B	C	XO	Rc
OPCD	D	A	0 0 0 0 0	C	XO	Rc
OPCD	D	0 0 0 0 0	B	0 0 0 0 0	XO	Rc

Table A-55.

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
fadd*x*	63	D	A	B	0 0 0 0 0	21	Rc
fadds*x*	59	D	A	B	0 0 0 0 0	21	Rc
fdiv*x*	63	D	A	B	0 0 0 0 0	18	Rc
fdivs*x*	59	D	A	B	0 0 0 0 0	18	Rc
fmadd*x*	63	D	A	B	C	29	Rc
fmadds*x*	59	D	A	B	C	29	Rc
fmsub*x*	63	D	A	B	C	28	Rc
fmsubs*x*	59	D	A	B	C	28	Rc
fmul*x*	63	D	A	0 0 0 0 0	C	25	Rc
fmuls*x*	59	D	A	0 0 0 0 0	C	25	Rc
fnmadd*x*	63	D	A	B	C	31	Rc
fnmadds*x*	59	D	A	B	C	31	Rc
fnmsub*x*	63	D	A	B	C	30	Rc
fnmsubs*x*	59	D	A	B	C	30	Rc
fres*x* [1]	59	D	0 0 0 0 0	B	0 0 0 0 0	24	Rc
frsqrte*x* [1]	63	D	0 0 0 0 0	B	0 0 0 0 0	26	Rc
fsel*x* [1]	63	D	A	B	C	23	Rc
fsqrt*x* [2]	63	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsqrts*x* [2]	59	D	0 0 0 0 0	B	0 0 0 0 0	22	Rc
fsub*x*	63	D	A	B	0 0 0 0 0	20	Rc
fsubs*x*	59	D	A	B	0 0 0 0 0	20	Rc

Table 19:

[1] Optional to the architecture but implemented by the MPC7410

[2] Optional instruction not implemented by the MPC7410

Table A-56. M-Form

OPCD	S	A	SH	MB	ME	Rc
OPCD	S	A	B	MB	ME	Rc

Table 20:

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25	26 27 28 29 30	31
rlwimi*x*	20	S	A	SH	MB	ME	Rc
rlwinm*x*	21	S	A	SH	MB	ME	Rc
rlwnm*x*	23	S	A	B	MB	ME	Rc

Table 21:

Table A-57. VA-Form

OPCD	**v**D	**v**A	**v**B	**v**C		XO
OPCD	**v**D	**v**A	**v**B	0	SH	XO

Table A-58.

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25	26 27 28 29 30 31
vmhaddshs[1]	04	**v**D	**v**A	**v**B	**v**C		32
vmhraddshs[1]	04	**v**D	**v**A	**v**B	**v**C		33
vmladduhm[1]	04	**v**D	**v**A	**v**B	**v**C		34
vmsumubm[1]	04	**v**D	**v**A	**v**B	**v**C		36
vmsummbm[1]	04	**v**D	**v**A	**v**B	**v**C		37
vmsumuhm[1]	04	**v**D	**v**A	**v**B	**v**C		38
vmsumuhs[1]	04	**v**D	**v**A	**v**B	**v**C		39
vmsumshm[1]	04	**v**D	**v**A	**v**B	**v**C		40
vmsumshs[1]	04	**v**D	**v**A	**v**B	**v**C		41
vsel[1]	04	**v**D	**v**A	**v**B	**v**C		42
vperm[1]	04	**v**D	**v**A	**v**B	**v**C		43
vsldoi[1]	04	**v**D	**v**A	**v**B	0	SH	44
vmaddfp[1]	04	**v**D	**v**A	**v**B	**v**C		46
vnmsubfp[1]	04	**v**D	**v**A	**v**B	**v**C		47

[1] AltiVec technology-specific instruction

Table A-59. VX-Form

OPCD	**v**D	**v**A	**v**B	XO	
OPCD	**v**D	0 0 0 0 0	0 0 0 0 0	XO	0
OPCD	0 0 0 0 0	0 0 0 0 0	**v**B	XO	0
OPCD	**v**D	0 0 0 0 0	**v**B	XO	
OPCD	**v**D	UIMM	**v**B	XO	

OPCD	vD	SIMM	0 0 0 0 0	XO

Specific Instructions

Name	0 5	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
vaddubm[1]	04	vD	vA	vB	0
vadduhm[1]	04	vD	vA	vB	64
vadduwm[1]	04	vD	vA	vB	128
vaddcuw[1]	04	vD	vA	vB	384
vaddubs[1]	04	vD	vA	vB	512
vadduhs[1]	04	vD	vA	vB	576
vadduws[1]	04	vD	vA	vB	640
vaddsbs[1]	04	vD	vA	vB	768
vaddshs[1]	04	vD	vA	vB	832
vaddsws[1]	04	vD	vA	vB	896
vsububm[1]	04	vD	vA	vB	1024
vsubuhm[1]	04	vD	vA	vB	1088
vsubuwm[1]	04	vD	vA	vB	1152
vsubcuw[1]	04	vD	vA	vB	1408
vsububs[1]	04	vD	vA	vB	1536
vsubuhs[1]	04	vD	vA	vB	1600
vsubuws[1]	04	vD	vA	vB	1664
vsubsbs[1]	04	vD	vA	vB	1792
vsubshs[1]	04	vD	vA	vB	1856
vsubsws[1]	04	vD	vA	vB	1920
vmaxub[1]	04	vD	vA	vB	2
vmaxuh[1]	04	vD	vA	vB	66
vmaxuw[1]	04	vD	vA	vB	130
vmaxsb[1]	04	vD	vA	vB	258
vmaxsh[1]	04	vD	vA	vB	322
vmaxsw[1]	04	vD	vA	vB	386
vminub[1]	04	vD	vA	vB	514
vminuh[1]	04	vD	vA	vB	578
vminuw[1]	04	vD	vA	vB	642
vminsb[1]	04	vD	vA	vB	770
vminsh[1]	04	vD	vA	vB	834

vminsw[1]	04	vD	vA	vB	898	
vavgub[1]	04	vD	vA	vB	1026	
vavguh[1]	04	vD	vA	vB	1090	
vavguw[1]	04	vD	vA	vB	1154	
vavgsb[1]	04	vD	vA	vB	1282	
vavgsh[1]	04	vD	vA	vB	1346	
vavgsw[1]	04	vD	vA	vB	1410	
vrlb[1]	04	vD	vA	vB	4	
vrlh[1]	04	vD	vA	vB	68	
vrlw[1]	04	vD	vA	vB	132	
vslb[1]	04	vD	vA	vB	260	
vslh[1]	04	vD	vA	vB	324	
vslw[1]	04	vD	vA	vB	388	
vsl[1]	04	vD	vA	vB	452	
vsrb[1]	04	vD	vA	vB	516	
vsrh[1]	04	vD	vA	vB	580	
vsrw[1]	04	vD	vA	vB	644	
vsr[1]	04	vD	vA	vB	708	
vsrab[1]	04	vD	vA	vB	772	
vsrah[1]	04	vD	vA	vB	836	
vsraw[1]	04	vD	vA	vB	900	
vand[1]	04	vD	vA	vB	1028	
vandc[1]	04	vD	vA	vB	1092	
vor[1]	04	vD	vA	vB	1156	
vnor[1]	04	vD	vA	vB	1284	
mfvscr[1]	04	vD	0 0 0 0 0	0 0 0 0 0	1540	0
mtvscr[1]	04	0 0 0 0 0	0 0 0 0 0	vB	1604	0
vmuloub[1]	04	vD	vA	vB	8	
vmulouh[1]	04	vD	vA	vB	72	
vmulosb[1]	04	vD	vA	vB	264	
vmulosh[1]	04	vD	vA	vB	328	
vmuleub[1]	04	vD	vA	vB	520	
vmuleuh[1]	04	vD	vA	vB	584	
vmulesb[1]	04	vD	vA	vB	776	
vmulesh[1]	04	vD	vA	vB	840	

vsum4ubs[1]	04	vD	vA	vB	1544
vsum4sbs[1]	04	vD	vA	vB	1800
vsum4shs[1]	04	vD	vA	vB	1608
vsum2sws[1]	04	vD	vA	vB	1672
vsumsws[1]	04	vD	vA	vB	1928
vaddfp[1]	04	vD	vA	vB	10
vsubfp[1]	04	vD	vA	vB	74
vrefp[1]	04	vD	0 0 0 0 0	vB	266
vrsqrtefp[1]	04	vD	0 0 0 0 0	vB	330
vexptefp[1]	04	vD	0 0 0 0 0	vB	394
vlogefp[1]	04	vD	0 0 0 0 0	vB	458
vrfin[1]	04	vD	0 0 0 0 0	vB	522
vrfiz[1]	04	vD	0 0 0 0 0	vB	586
vrfip[1]	04	vD	0 0 0 0 0	vB	650
vrfim[1]	04	vD	0 0 0 0 0	vB	714
vcfux[1]	04	vD	UIMM	vB	778
vcfsx[1]	04	vD	UIMM	vB	842
vctuxs[1]	04	vD	UIMM	vB	906
vctsxs[1]	04	vD	UIMM	vB	970
vmaxfp[1]	04	vD	vA	vB	1034
vminfp[1]	04	vD	vA	vB	1098
vmrghb[1]	04	vD	vA	vB	12
vmrghh[1]	04	vD	vA	vB	76
vmrghw[1]	04	vD	vA	vB	140
vmrglb[1]	04	vD	vA	vB	268
vmrglh[1]	04	vD	vA	vB	332
vmrglw[1]	04	vD	vA	vB	396
vspltb[1]	04	vD	UIMM	vB	524
vsplth[1]	04	vD	UIMM	vB	588
vspltw[1]	04	vD	UIMM	vB	652
vspltisb[1]	04	vD	SIMM	0 0 0 0 0	780
vspltish[1]	04	vD	SIMM	0 0 0 0 0	844
vspltisw[1]	04	vD	SIMM	0 0 0 0 0	908
vslo[1]	04	vD	vA	vB	1036
vsro[1]	04	vD	vA	vB	1100

vpkuhum[1]	04	vD	vA	vB	14	
vpkuwum[1]	04	vD	vA	vB	78	
vpkuhus[1]	04	vD	vA	vB	142	
vpkuwus[1]	04	vD	vA	vB	206	
vpkshus[1]	04	vD	vA	vB	270	
vpkswus[1]	04	vD	vA	vB	334	
vpkshss[1]	04	vD	vA	vB	398	
vpkswss[1]	04	vD	vA	vB	462	
vpkswus[1]	04	vD	vA	vB	334	
vupkhsb[1]	04	vD	0 0 0 0 0	vB	526	
vupkhsh[1]	04	vD	0 0 0 0 0	vB	590	
vupklsb[1]	04	vD	0 0 0 0 0	vB	654	
vupklsh[1]	04	vD	0 0 0 0 0	vB	718	
vpkpx[1]	04	vD	vA	vB	12	782
vupkhpx[1]	04	vD	0 0 0 0 0	vB	846	
vupklpx[1]	04	vD	0 0 0 0 0	vB	974	
vxor[1]	04	vD	vA	vB	1220	

[1] AltiVec technology-specific instruction

OPCD	vD	vA	vB	Rc	XO

Table A-60.

Table A-61. VXR-Form

Specific Instructions

Name	05	6 7 8 9 10	11 12 13 14 15	16 17 18 19 20	21	22 23 24 25 26 27 28 29 30 31
vcmpbfp_x_[1]	04	**v**D	**v**A	**v**B	Rc	966
vcmpeqfp_x_[1]	04	**v**D	**v**A	**v**B	Rc	198
vcmpequb_x_[1]	04	**v**D	**v**A	**v**B	Rc	6
vcmpequh_x_[1]	04	**v**D	**v**A	**v**B	Rc	70
vcmpequw_x_[1]	04	**v**D	**v**A	**v**B	Rc	134
vcmpgefp_x_[1]	04	**v**D	**v**A	**v**B	Rc	454
vcmpgtfp_x_[1]	04	**v**D	**v**A	**v**B	Rc	710
vcmpgtsb_x_[1]	04	**v**D	**v**A	**v**B	Rc	774
vcmpgtsh_x_[1]	04	**v**D	**v**A	**v**B	Rc	838
vcmpgtsw_x_[1]	04	**v**D	**v**A	**v**B	Rc	902
vcmpgtub_x_[1]	04	**v**D	**v**A	**v**B	Rc	518
vcmpgtuh_x_[1]	04	**v**D	**v**A	**v**B	Rc	582
vcmpgtuw_x_[1]	04	**v**D	**v**A	**v**B	Rc	646

Table 22:

[1] AltiVec techlogy-specific instruction

A.7 Instruction Set Legend

Table A-62 provides general information on the MPC7410 instruction set (such as the architectural level, privilege level, and form).

Table A-62. MPC7410 General Instruction Set Legend

	UISA	VEA	OEA	Supervisor Level	Optional	Form
add*x*	√					XO
addc*x*	√					XO
adde*x*	√					XO
addi	√					D
addic	√					D
addic.	√					D
addis	√					D
addme*x*	√					XO
addze*x*	√					XO
and*x*	√					X
andc*x*	√					X
andi.	√					D
andis.	√					D
b*x*	√					I
bc*x*	√					B
bcctr*x*	√					XL
bclr*x*	√					XL
cmp	√					X
cmpi	√					D
cmpl	√					X
cmpli	√					D
cntlzw*x*	√					X
crand	√					XL
crandc	√					XL
creqv	√					XL
crnand	√					XL
crnor	√					XL

Table A-62. MPC7410 General Instruction Set Legend (Continued)

	UISA	VEA	OEA	Supervisor Level	Optional	Form
cror	√					XL
crorc	√					XL
crxor	√					XL
dcba		√			√	X
dcbf		√				X
dcbi			√	√		X
dcbst		√				X
dcbt		√				X
dcbtst		√				X
dcbz		√				X
divw*x*	√					XO
divwu*x*	√					XO
eciwx		√			√	X
ecowx		√			√	X
eieio		√				X
eqv*x*	√					X
extsb*x*	√					X
extsh*x*	√					X
fabs*x*	√					X
fadd*x*	√					A
fadds*x*	√					A
fcmpo	√					X
fcmpu	√					X
fctiw*x*	√					X
fctiwz*x*	√					X
fdiv*x*	√					A
fdivs*x*	√					A
fmadd*x*	√					A
fmadds*x*	√					A
fmr*x*	√					X
fmsub*x*	√					A

Table A-62. MPC7410 General Instruction Set Legend (Continued)

	UISA	VEA	OEA	Supervisor Level	Optional	Form
fmsubs*x*	√					A
fmul*x*	√					A
fmuls*x*	√					A
fnabs*x*	√					X
fneg*x*	√					X
fnmadd*x*	√					A
fnmadds*x*	√					A
fnmsub*x*	√					A
fnmsubs*x*	√					A
fres*x*	√				√	A
frsp*x*	√					X
frsqrte*x*	√				√	A
fsel*x*	√				√	A
fsqrt*x*	√				√	A
fsqrts*x*	√				√	A
fsub*x*	√					A
fsubs*x*	√					A
icbi		√				X
isync		√				XL
lbz	√					D
lbzu	√					D
lbzux	√					X
lbzx	√					X
lfd	√					D
lfdu	√					D
lfdux	√					X
lfdx	√					X
lfs	√					D
lfsu	√					D
lfsux	√					X
lfsx	√					X

Table A-62. MPC7410 General Instruction Set Legend (Continued)

	UISA	VEA	OEA	Supervisor Level	Optional	Form
lha	√					D
lhau	√					D
lhaux	√					X
lhax	√					X
lhbrx	√					X
lhz	√					D
lhzu	√					D
lhzux	√					X
lhzx	√					X
lmw [2]	√					D
lswi [2]	√					X
lswx [2]	√					X
lwarx	√					X
lwbrx	√					X
lwz	√					D
lwzu	√					D
lwzux	√					X
lwzx	√					X
mcrf	√					XL
mcrfs	√					X
mcrxr	√					X
mfcr	√					X
mffs	√					X
mfmsr			√	√		X
mfspr[1]	√		√	√		XFX
mfsr			√	√		X
mfsrin			√	√		X
mftb		√				XFX
mtcrf	√					XFX
mtfsb0*x*	√					X
mtfsb1*x*	√					X

Table A-62. MPC7410 General Instruction Set Legend (Continued)

	UISA	VEA	OEA	Supervisor Level	Optional	Form
mtfsf*x*	√					XFL
mtfsfi*x*	√					X
mtmsr			√	√		X
mtspr[1]	√		√	√		XFX
mtsr			√	√		X
mtsrin			√	√		X
mulhw*x*	√					XO
mulhwu*x*	√					XO
mulli	√					D
mullw*x*	√					XO
nand*x*	√					X
neg*x*	√					XO
nor*x*	√					X
or*x*	√					X
orc*x*	√					X
ori	√					D
oris	√					D
rfi			√	√		XL
rlwimi*x*	√					M
rlwinm*x*	√					M
rlwnm*x*	√					M
sc	√		√			SC
slw*x*	√					X
sraw*x*	√					X
srawi*x*	√					X
srw*x*	√					X
stb	√					D
stbu	√					D
stbux	√					X
stbx	√					X
stfd	√					D

Table A-62. MPC7410 General Instruction Set Legend (Continued)

	UISA	VEA	OEA	Supervisor Level	Optional	Form
stfdu	√					D
stfdux	√					X
stfdx	√					X
stfiwx	√					X
stfs	√					D
stfsu	√					D
stfsux	√					X
stfsx	√					X
sth	√					D
sthbrx	√					X
sthu	√					D
sthux	√					X
sthx	√					X
stmw [2]	√					D
stswi [2]	√					X
stswx [2]	√					X
stw	√					D
stwbrx	√					X
stwcx.	√					X
stwu	√					D
stwux	√					X
stwx	√					X
subf*x*	√					XO
subfc*x*	√					XO
subfe*x*	√					XO
subfic	√					D
subfme*x*	√					XO
subfze*x*	√					XO
sync	√					X
tlbia*x*			√	√	√	X
tlbie*x*			√	√	√	X

Table A-62. MPC7410 General Instruction Set Legend (Continued)

	UISA	VEA	OEA	Supervisor Level	Optional	Form
tlbsync			√	√		X
tw	√					X
twi	√					D
xor*x*	√					X

Table A-62. MPC7410 General Instruction Set Legend (Continued)

	UISA	VEA	OEA	Supervisor Level	Optional	Form
xori	√					D
xoris	√					D

Notes:

[2] Load/store string or multiple instruction

[3] Optional instruction provided to support temporary 64-bit bridge

[4] Defined for the 32-bit architecture and by the temporary 64-bit bridge

Table A-63 provides information specific to the MPC7410 instruction set.

Table A-63. MPC7410-Specific Instruction Set Legend

dss[1]		√						√	VX
dssall[1]		√						√	VX
dst[1]	√							√	VX
dstst[1]		√						√	VX
dststt[1]		√						√	VX
dstt[1]		√						√	VX
lvebx[1]	√							√	X
lvehx[1]	√							√	X
lvewx[1]	√							√	X
lvsl[1]	√							√	X
lvsr[1]	√							√	X
lvx[1]	√							√	X
lvxl[1]	√							√	X
mfvscr[1]	√							√	VX
mtvscr[1]	√							√	VX
stvebx[1]	√							√	X
stvehx[1]	√							√	X
stvewx[1]	√							√	X
stvx[1]	√							√	X
stvxl[1]	√							√	X
vaddcuw[1]	√							√	VX
vaddfp[1]	√							√	VX
vaddsbs[1]	√							√	VX
vaddshs[1]	√							√	VX

Table A-63. MPC7410-Specific Instruction Set Legend (Continued)

vaddsws[1]	√							√	VX
vaddubm[1]	√							√	VX
vaddubs[1]	√							√	VX
vadduhm[1]	√							√	VX
vadduhs[1]	√							√	VX
vadduwm[1]	√							√	VX
vadduws[1]	√							√	VX
vand[1]	√							√	VX
vandc[1]	√							√	VX
vavgsb[1]	√							√	VX
vavgsh[1]	√							√	VX
vavgsw[1]	√							√	VX
vavgub[1]	√							√	VX
vavguh[1]	√							√	VX
vavguw[1]	√							√	VX
vcfsx[1]	√							√	VX
vcfux[1]	√							√	VX
vcmpbfp_x_[1]	√							√	VXR
vcmpeqfp_x_[1]	√							√	VXR
vcmpequb_x_[1]	√							√	VXR
vcmpequh_x_[1]	√							√	VXR
vcmpequw_x_[1]	√							√	VXR
vcmpgefp_x_[1]	√							√	VXR
vcmpgtfp_x_[1]	√							√	VXR
vcmpgtsb_x_[1]	√							√	VXR
vcmpgtsh_x_[1]	√							√	VXR
vcmpgtsw_x_[1]	√							√	VXR
vcmpgtub_x_[1]	√							√	VXR
vcmpgtuh_x_[1]	√							√	VXR
vcmpgtuw_x_[1]	√							√	VXR
vctsxs[1]	√							√	VX
vctuxs[1]	√							√	VX
vexptefp[1]	√							√	VX
vlogefp[1]	√							√	VX

Table A-63. MPC7410-Specific Instruction Set Legend (Continued)

vmaddfp[1]	√							√	VA
vmaxfp[1]	√							√	VX
vmaxsb[1]	√							√	VX
vmaxsh[1]	√							√	VX
vmaxsw[1]	√							√	VX
vmaxub[1]	√							√	VX
vmaxuh[1]	√							√	VX
vmaxuw[1]	√							√	VX
vmhaddshs[1]	√							√	VA
vmhraddshs[1]	√							√	VA
vminfp[1]	√							√	VX
vminsb[1]	√							√	VX
vminsh[1]	√							√	VX
vminsw[1]	√							√	VX
vminub[1]	√							√	VX
vminuh[1]	√							√	VX
vminuw[1]	√							√	VX
vmladduhm[1]	√							√	VA
vmrghb[1]	√							√	VX
vmrghh[1]	√							√	VX
vmrghw[1]	√							√	VX
vmrglb[1]	√							√	VX
vmrglh[1]	√							√	VX
vmrglw[1]	√							√	VX
vmsummbm[1]	√							√	VA
vmsumshm[1]	√							√	VA
vmsumshs[1]	√							√	VA
vmsumubm[1]	√							√	VA
vmsumuhm[1]	√							√	VA
vmsumuhs[1]	√							√	VA
vmulesb[1]	√							√	VX
vmulesh[1]	√							√	VX
vmuleub[1]	√							√	VX
vmuleuh[1]	√							√	VX

Table A-63. MPC7410-Specific Instruction Set Legend (Continued)

vmulosb[1]	√							√	VX
vmulosh[1]	√							√	VX
vmuloub[1]	√							√	VX
vmulouh[1]	√							√	VX
vnmsubfp[1]	√							√	VA
vnor[1]	√							√	VX
vor[1]	√							√	VX
vperm[1]	√							√	VA
vpkpx[1]	√							√	VX
vpkshss[1]	√							√	VX
vpkshus[1]	√							√	VX
vpkswss[1]	√							√	VX
vpkswus[1]	√							√	VX
vpkuhum[1]	√							√	VX
vpkuhus[1]	√							√	VX
vpkuwum[1]	√							√	VX
vpkuwus[1]	√							√	VX
vrefp[1]	√							√	VX
vrfim[1]	√							√	VX
vrfin[1]	√							√	VX
vrfip[1]	√							√	VX
vrfiz[1]	√							√	VX
vrlb[1]	√							√	VX
vrlh[1]	√							√	VX
vrlw[1]	√							√	VX
vrsqrtefp[1]	√							√	VX
vsel[1]	√							√	VA
vsl[1]	√							√	VX
vslb[1]	√							√	VX
vsldoi[1]	√							√	VA
vslh[1]	√							√	VX
vslo[1]	√							√	VX
vslw[1]	√							√	VX
vspltb[1]	√							√	VX

Table A-63. MPC7410-Specific Instruction Set Legend (Continued)

vsplth[1]	√							√	VX
vspltisb[1]	√							√	VX
vspltish[1]	√							√	VX
vspltisw[1]	√							√	VX
vspltw[1]	√							√	VX
vsr[1]	√							√	VX
vsrab[1]	√							√	VX
vsrah[1]	√							√	VX
vsraw[1]	√							√	VX
vsrb[1]	√							√	VX
vsrh[1]	√							√	VX
vsro[1]	√							√	VX
vsrw[1]	√							√	VX
vsubcuw[1]	√							√	VX
vsubfp[1]	√							√	VX
vsubsbs[1]	√							√	VX
vsubshs[1]	√							√	VX
vsubsws[1]	√							√	VX
vsububm[1]	√							√	VX
vsubuhm[1]	√							√	VX
vsububs[1]	√							√	VX
vsubuhs[1]	√							√	VX
vsubuwm[1]	√							√	VX
vsubuws[1]	√							√	VX
vsumsws[1]	√							√	VX
vsum2sws[1]	√							√	VX
vsum4sbs[1]	√							√	VX
vsum4shs[1]	√							√	VX
vsum4ubs[1]	√							√	VX
vupkhpx[1]	√							√	VX
vupkhsb[1]	√							√	VX
vupkhsh[1]								√	VX
vupklpx[1]								√	VX
vupklsb[1]								√	VX

Table A-63. MPC7410-Specific Instruction Set Legend (Continued)

vupklsh[1]	√							√	VX
vxor[1]	√							√	VX

[1] AltiVec technology-specific instruction

.

Appendix B
Instructions Not Implemented

This appendix provides a list of the 32-bit and 64-bit PowerPC instructions that are not implemented in the MPC7410 microprocessor. Note that any attempt to execute instructions that are not implemented on the MPC7410 will generate an illegal instruction exception. Note that exceptions are referred to as interrupts in the architecture specification.

Table B-1 provides the 32-bit PowerPC instructions that are optional to the architecture and not implemented by the MPC7410.

Table B-1. 32-Bit Instructions Not Implemented by the MPC7410 Processor

Mnemonic	Instruction
dcba	Data Cache Block Allocate
fsqrt	Floating Square Root (Double-Precision)
fsqrts	Floating Square Root Single
tlbia	TLB Invalidate All

Table B-2 provides a list of 64-bit instructions that are not implemented by the MPC7450.

Table B-2. 64-Bit Instructions Not Implemented by the MPC7410 Processor

Mnemonic	Instruction
cntlzd	Count Leading Zeros Double Word
divd	Divide Double Word
divdu	Divide Double Word Unsigned
extsw	Extend Sign Word
fcfid	Floating Convert From Integer Double Word
fctid	Floating Convert to Integer Double Word
fctidz	Floating Convert to Integer Double Word with Round toward Zero
ld	Load Double Word
ldarx	Load Double Word and Reserve Indexed
ldu	Load Double Word with Update
ldux	Load Double Word with Update Indexed
ldx	Load Double Word Indexed
lwa	Load Word Algebraic
lwaux	Load Word Algebraic with Update Indexed

Table B-2. 64-Bit Instructions Not Implemented by the MPC7410 Processor (Continued)

Mnemonic	Instruction
lwax	Load Word Algebraic Indexed
mtmsrd	Move to Machine State Register Double Word
mtsrd	Move to Segment Register Double Word
mtsrdin	Move to Segment Register Double Word Indirect
mulld	Multiply Low Double Word
mulhd	Multiply High Double Word
mulhdu	Multiply High Double Word Unsigned
rldcl	Rotate Left Double Word then Clear Left
rldcr	Rotate Left Double Word then Clear Right
rldic	Rotate Left Double Word Immediate then Clear
rldicl	Rotate Left Double Word Immediate then Clear Left
rldicr	Rotate Left Double Word Immediate then Clear Right
rldimi	Rotate Left Double Word Immediate then Mask Insert
slbia	SLB Invalidate All
slbie	SLB Invalidate Entry
sld	Shift Left Double Word
srad	Shift Right Algebraic Double Word
sradi	Shift Right Algebraic Double Word Immediate
srd	Shift Right Double Word
std	Store Double Word
stdcx.	Store Double Word Conditional Indexed
stdu	Store Double Word with Update
stdux	Store Double Word Indexed with Update
stdx	Store Double Word Indexed
td	Trap Double Word
tdi	Trap Double Word Immediate

Appendix C
Revision History

C.1 History of User’s Manual Revisions

This appendix provides a list of the major differences between the *MPC7410/MPC7400 RISC Microprocessor Reference Manual*, Revision 0 through Revision 2.

C.2 Changes From Revision 1 to Revision 2

Major changes to the *MPC7410/MPC7400 RISC Microprocessor Reference Manual*, from Revision 1 to Revision 2 are as follows:

Section, Page No. **Changes**

Throughout Update language to reflect new definitions of PowerPC and Power Architecture technology.

2.1.1/2-1 Remove L2 Private Memory Control Register and second EAR/SPR 282 box from Figure 2-1, “Register Set Overview.”

2.1.5.5/2-25 In Table 2-11,“Instruction Address Breakpoint Register Field Descriptions,” remove the second row for bit 31.

2.3.2.4.1/2-45 In Table 2-22, “Control Registers Synchronization Requirements,” remove HID1, ICTRL, and LDSTCR rows. In row HID0, remove BHTCLR, FOLD, LRSTK, TBEN, STEN, and XAEN. To same row, add EMCP, EBA, EBD, BCLK, ECLK, and PAR with the synchronization requirement, “A sync and context synchronizing instruction must follow a **mtspr**.” Change “L3PM” register to “L2CR, L2PMCR”.

4.6.2/4-16 Change the second sentence of the first paragraph in “Machine Check Exception (0x00200)” section to read: “The MPC7410 conditionally initiates a machine check exception if MSR[ME] = 1 and any of the following occur:

- A system bus error (TEA assertion on data bus)
- Assertion of the machine check (MCP) signal
- Address bus parity error on system bus
- Data bus parity error onsystem bus
- L2 data bus parity error”

5.3/5-19 Update final paragraph to read “Thus, multiple BAT hits (with valid bits set) that map a given effective address to different physical addresses are considered a programming error whether translation is enabled or disabled. This can lead to unpredictable results if translation is enabled, or if translation is disabled when translation is eventually enabled. For the case of unused BATs, if translation is to be enabled, it is a sufficient precaution to simply clear the valid bits of the unused BAT entries."

C.3 Changes From Revision 0 to Revision 1

Section, Page No. **Changes**

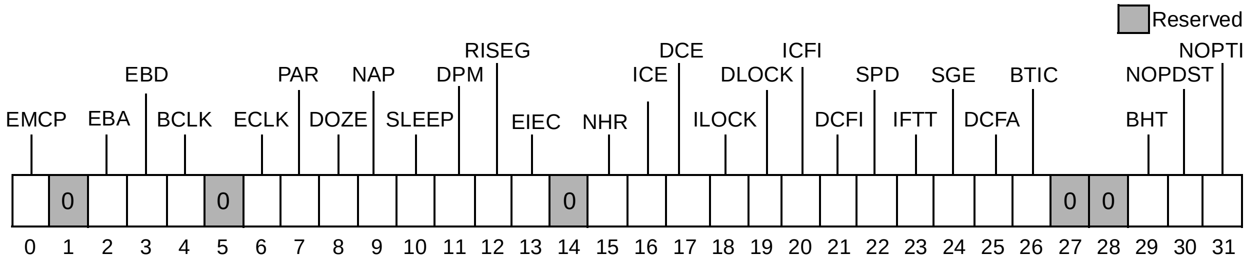
1.2.2.4.2, 1-13 Remove 'division' from the second bullet. The bullet should read as follows:

- Vector complex integer unit (VCIU)—executes longer-latency vector integer instructions, such as multiplication, multiplication/addition, and sum-across with saturation.

1.14, 1-45 In Table 1-8, add the following 'Load/Store Ordering' row after the'Sequencing' row:

Load/Store Ordering	On the MPC750, load and store operations are assumed to be weakly ordered. That is, the load/store unit (LSU) can perform load operations that occur later in the program ahead of store operation. However, strongly ordered load and store operations can be enforced through setting the caching-inhibited (I) memory/cache access attribute. On the MPC7410, load and store operations are also assumed to be weakly ordered, and load operations can bypass store operations. However, unlike the MPC750 and other PowerPC microprocessors, the MPC7410 does not enforce load/store ordering when the access is caching-inhibited or write-through guarded. See Section 3.4.4.2, "Sequential Consistency of Memory Accesses," for more information.

2.1.2.2, 2-11 In Figure 2-3, bit 12 of HID0 should be 'RISEG' instead of reserved. Replace the existing diagram with the following:



2.1.2.2, 2-13 In Table 2-4, bit 17 (DCE), the last sentence in the Function description for the 0 state should say 'DCE' instead of 'ICE.' The description should read as follows:

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 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.

2.1.2.2, 2-15 Replace Table 2-5 with the following:

Table 0-1. Table 2-5. HID0[BCLK] and HID0[ECLK] CLK_OUT Configuration

$\overline{\text{HRESET}}$	HID0[ECLK]	HID0[BCLK]	CLK_OUT
Asserted	x	x	External bus clock (SYSCLK)
Negated	0	0	Reserved for factory
Negated	0	1	Reserved for factory
Negated	1	0	Core
Negated	1	1	External bus clock (SYSCLK)

2.1.6, 2-25 Replace the second paragraph with the following:

Because the MSSCR0 parameters SHDEN, SHDPEN3, L1_INTVEN, and L2_INTVEN alter how the MPC7410 responds to snoop requests, it is important that changes to these parameters are handled correctly.

Replace the second sentence of the second paragraph, ‘The correct sequence ... value of MSSCR0 is as follows:’ with the following:

The correct sequence necessary to change the values for HDEN, SHDPEN3, L1_INTVEN, and L2_INTVEN is as follows:

2.1.7, 2-29 In Table 2-17, bit 14–15 (L2OH), the last bit setting should be ‘11’ instead of ‘10.’ The description should read as follows:

14–15	L2OH	L2 output hold Configure output hold time for address, data, and control signals driven by the MPC7410 to the L2 data RAMs. They should generally be set according to the SRAMs input hold time requirements, for which late-write SRAMs usually differ from flow-through or burst SRAMs. 00 Shortest output hold 01 Short output hold 10 Long output hold 11 Longest output hold See the MPC7410 hardware specifications for specific output hold times.

2.1.7, 2-29 In Table 2-17, bit 16 (L2SL), replace the last sentence with the following (note that 100 MHz is replaced with 150 MHz):

Generally, L2SL should be set if the L2 RAM interface is operated below 150 MHz.

2.3.4.4.1, 2-57 In the third paragraph of the section (first paragraph on the page), remove the following part of the first sentence:

and condition register logical instructions (**crand**, **cror**, **crxor**, **crnand**, **crnor**, **crandc**, **creqv**, **crorc**, and **mcrf**).

The sentence should read as follows:

Note that in the MPC7410, all branch instructions (**b**, **ba**, **bl**, **bla**, **bc**, **bca**, **bcl**, **bcla**, **bclr**, **bclrl**, **bcctr**,and **bcctrl**) are executed by the BPU.

3.4.4, 3-30 Replace the first paragraph with the following:

Load and store operations are assumed to be weakly ordered on the MPC7410. In general, the load/store unit (LSU) can perform load operations that occur later in the program ahead of store operations, even when the access is caching-inhibited or when data cache is disabled. Any load followed by any store is performed in order. See Section 3.4.4.2, “Sequential Consistency of Memory Accesses,” for more information.

3.4.4.2, 3-31 Split the second paragraph into two and add Table 3-8. The second paragraph, Table 3-8, and the third paragraph should read as follows:

The MPC7410 achieves sequential consistency by operating a single pipeline to the cache/MMU. All memory accesses are presented to the MMU in exact program order and, therefore, exceptions are determined in order. Table 3-8 describes load/store ordering.

Table 3-8. MPC7410 Load/Store Ordering

Cache/Memory Access Attributes	WIMG[1]	Store-Store Ordered	Load-Load Ordered	Store-Load Ordered	Load-Store Ordered
Caching-inhibited, guarded	01x1	Yes	Yes	Requires **eieio**	Yes
Caching-inhibited, non-guarded	01x0	Yes	Yes	Requires **sync**	Yes
Write-through, guarded	10x1	Yes	Yes	Requires **sync**	Yes
Write-through, non-guarded	10x0	Yes	Requires **eieio**	Requires **sync**	Yes
Write-back, coherency-required	001x	Requires **eieio**	Requires **eieio**	Requires **sync**	Yes
Write-back, coherency-not-required	000x	Requires **eieio**	Requires **eieio**	Requires **sync**	Yes

[1] The PowerPC architecture states that combinations where WIMG = 11xx are not supported.

Loads are allowed to bypass stores once exception checking has been performed for the store, but data dependency checking is handled in the load/store unit so that a load will not bypass a store with an address match. Newer caching-allowed loads can bypass older caching-allowed loads only if the two loads are to different 32-byte address granules. Newer caching-allowed write-back stores can bypass older caching-allowed write-back stores if they do not store to overlapping bytes of data.

3.4.4.3, 3-31 Replace the first paragraph with the following:

Unlike previous PowerPC microprocessor implementations, the MPC7410 does reorder cache-inhibited memory accesses and write-through, guarded memory accesses. As shown in Table 3-8, certain memory accesses require an **eieio** or a **sync** instruction to ensure ordering. These instructions are used to enforce storage ordering.”

3.7.2, 3-50 After the first paragraph, add Figure 3-35 and the following paragraph:

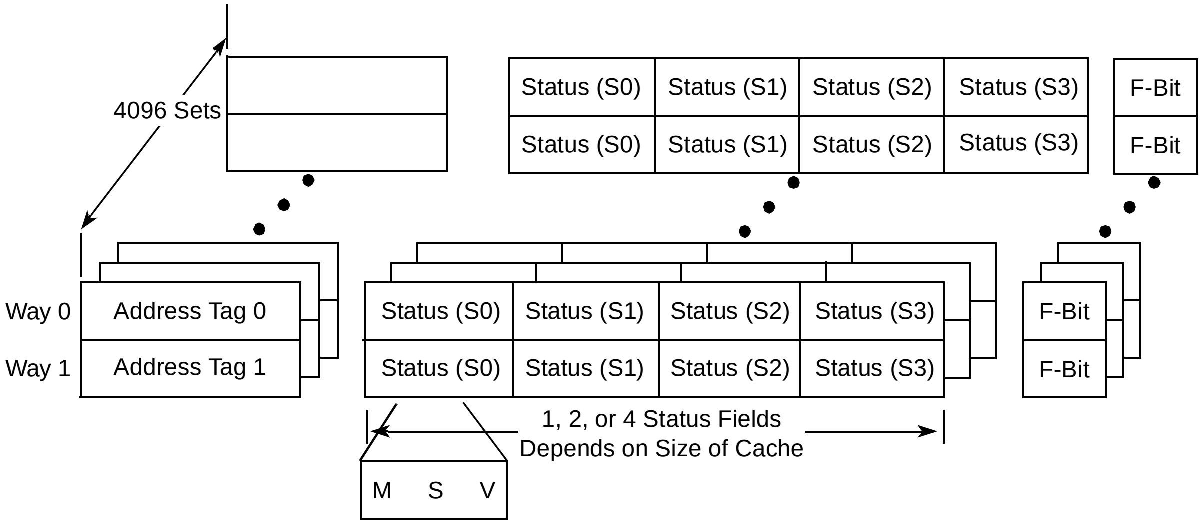


Figure 0-1. Figure 3-35. L2 Cache Controller Tag Organization

Physical address bits PA[13:24] provide the index to select a cache set. The tags consist of physical address bits PA[0:12]. Physical address bits A[25:31] locate a byte within the selected block.

3.7.3.7, 3-54 Add the following paragraph as the first step for performing a global invalidation of the L2 cache:

1. Prefetch the code that monitors L2CR[L2IP] (step 5) into the L1 instruction cache. The L2IP monitor code must be resident in the L1 instruction cache before the L2CR[L2I] bit is set (step 4). Otherwise the global invalidate operation will prevent the fetching of the L2IP monitor code from memory until after the invalidate has completed and the L2IP monitor code will never see the L2IP bit set.

3.7.3.7, 3-54 In the first sentence of step 4, replace ‘L2CR[L2P]’ with ‘L2CR[L2IP].’ The sentence should read as follows:

2. Monitor the L2CR[L2IP] bit to determine when the global invalidation operation is completed (indicated by the clearing of L2CR[L2IP]).

3.7.6.2, 3-62 Add the following sentence at the beginning of the only paragraph:

The L2 cache uses a least-recently used (LRU) replacement algorithm.

3.7.9, 3-65 Add the following sentence at the end of the last paragraph (first paragraph on the page):

Note that due to the influence of L2TS on the replacement algorithm, it is necessary to initialize an address range that is twice (2X) the physical L2 cache size and perform testing on the second half of that address range.

3.7.9.2, 3-65 Add the following sentence after the first sentence of step 4:

The range of addresses must be twice the physical L2 cache size.

Replace the second sentence of step 6 as follows:

These loads and stores should be in the second half of the range of addresses used to initialize the caches in step 4 so that each access hits in the L2 cache.

3.7.9.3, 3-66 Add the following sentence after the first sentence of step 4:

The range of addresses must be twice the physical L2 cache size.

Replace the first sentence of step 8 as follows:

Perform a series of reads from the second half of the original range of addresses located in the cache and verify that the data read was not affected by the stores performed in step 6.

Replace the second sentence of step 9 as follows:

All accesses to the second half of the original region should hit.

3.9.3, 3-77 Add the following sentence at the end of the last paragraph:

Note that the MPC7410 snoops its own transactions and may assert $\overline{\text{ARTRY}}$ for **tlbie**, **tlbsync**, **icbi**, and **sync** broadcasts that result in pipeline collisions.

4.2, 4-6 In Table 4-3, remove the ‘L1 address or data parity error’ from the description of the machine check exception. The description should read as follows:

1	Machine check	Any enabled machine check condition (assertion of $\overline{\text{TEA}}$ or $\overline{\text{MCP}}$, or memory subsystem error as defined in MSSSR0 (see Section 2.1.5.4, “Memory Subsystem Status Register (MSSSR0) for further details), address or data parity error, data cache error, instruction cache error, L2 data parity error, L2 tag error)

4.2, 4-7 In Table 4-3, AltiVec unavailable exception has the priority of 4 instead of 3.

4.6, 4-13 In Table 4-6, add the following row at the end of the table:

AltiVec assist	0	0	—	0	0	0	—	0	0	0	0	—	0	0	0	0	ILE

4.6.2, 4-16 In Table 4-8, HID0[DBP] is a reserved bit, so remove the description on the DBP bit from the table.

5.2, 5-18 Replace the fourth line of the second paragraph with the following sentence:

Note also, that the G bit must be set to ensure that store operations are strongly ordered with other store operations and load operations are strongly ordered with other load operations.

6.3.2.3, 6-15 In the second sentence of the first paragraph, the processor/bus clock ratio should be ‘2:1’ instead of ‘1:2.’ The sentence should read as follows:

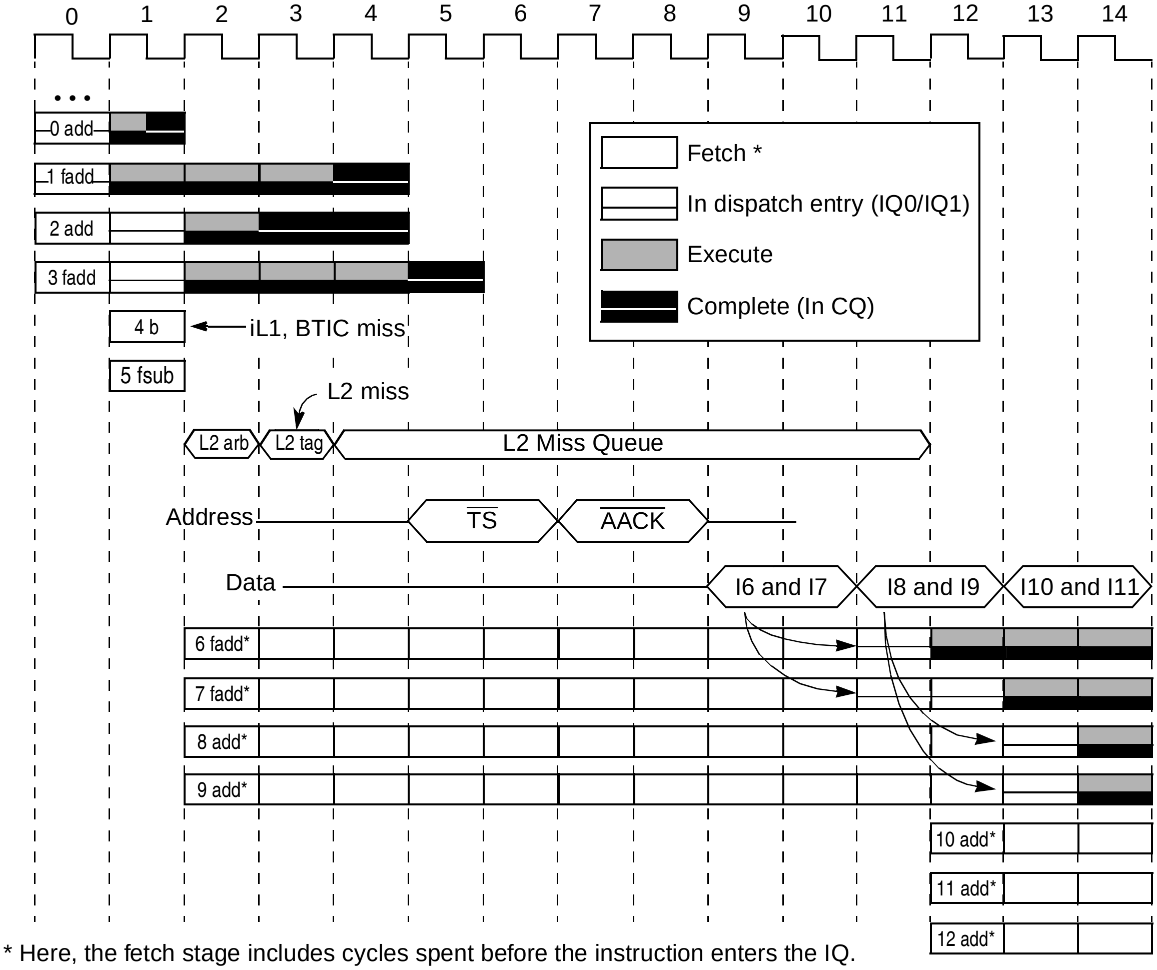
A processor/bus clock ratio of 2:1 is used.

Delete the last sentence of the first paragraph. The paragraph should read as follows:

Figure 6-6 shows an instruction fetch that misses both the on-chip cache and L2 cache. A processor/bus clock ratio of 2:1 is used. The same instruction sequence is used as in Section 6.3.2.2, “Cache Hit,” however in this example, the branch target instruction is not in either the L1 or L2 cache.

Delete second and third paragraphs from this section.

6.3.2.3, 6-16 Replace Figure 6-6 with the following cache miss timing diagram:



* Here, the fetch stage includes cycles spent before the instruction enters the IQ.

Instruction Queue

IQ5															
IQ4															
IQ3	3	5													
IQ2	2	4													
IQ1	1	3										7		9	
IQ0	0	2										6	7	8	

Completion Queue

CQ7															
CQ6															
CQ5															
CQ4															
CQ3															9
CQ2			3	3	3										8
CQ1		1	2	2	2									7	7
CQ0		0	1	1	1	3							6	6	6

6.3.2.3, 6-16 Add the following two paragraphs after Figure 6-6:

A cache miss extends the latency of the fetch stage, so in this example, the fetch stage shown represents not only the time the instruction spends in the IQ, but the time required for the instruction to be loaded from system memory, beginning in clock cycle 3.

By clock cycle 3, a memory access must occur because the target instruction for the **b** instruction is not in the BTIC (the target instruction is not in the L1 cache, so it cannot be in the BTIC), the instruction cache, or the L2 cache. During clock cycle 5, the address of the block of instructions is sent to the system bus. During clock cycle 9, two instructions (64 bits) are returned from memory on the first beat and are forwarded both to the cache and the instruction fetcher.

6.3.3.1, 6-19 Replace the last sentence of the first paragraph and the formula for the L2 cache latency with the following:

The formula for the L2 cache latency for instruction and data accesses is as follows:

2 processor clock + 3 L2 clocks + 1 processor clock

6.4.1.1, 6-23 Replace the third paragraph with the following:

Figure 6-11 shows branch folding. Here a **b** instruction is encountered in a series of **add** instructions. The branch is resolved as taken. What happens on the next clock cycle depends on whether the target instruction stream (**add**) is in the BTIC, the instruction cache, or if it must be fetched from the L2 cache or from system memory.

8.1.2, 8-5 In Table 8-2, replace the row ‘Clock control,’ with the following row:

Clock control	CLK_OUT	Bus clock (SYSCLK)

8.2.2.3, 8-8 The section should be changed as follows:

In the State Meaning, Negated, delete two sentences beginning with ‘If $\overline{\text{ABB}}$ is negated ...’ and ending with ‘... transaction begins.’

In the Timing Comments, Assertion, replace with ‘Asserted with $\overline{\text{TS}}$.’

In the Timing Comments, Negation, delete the sentence beginning with ‘If ABB is negated ... was asserted.’

8.2.5.2.1, 8-14 Add the following two sentences at the end of ‘Asserted’ state of the $\overline{\text{ARTRY}}$ signal:

Note that the MPC7410 is self-snooping and may assert $\overline{\text{ARTRY}}$ for its own transaction. See Section 3.9.3, “Snooping,” for more information.

8.5.1.1, 8-38 Replace the state meaning of ‘Asserted/Negated’ with the following description and add Table 8-7 after this paragraph.

Asserted/Negated—Represents the address of the data to be transferred to the L2 cache. The L2 address bus is configured with bit 0 as the least-significant bit. The L2 address signals reflect the real address for various L2 cache sizes and data bus widths as shown in Table 8-7. Note that the L2 address does not correspond bit-for-bit with the real address.

Table 0-2. Table 8-7. L2 Cache Address Signal Mappings

L2ADDR	Real Address Bit							
	64-Bit Data Bus				32-Bit Data Bus			
	2-Mbyte	1-Mbyte	512-Kbyte	256-Kbyte	2-Mbyte	1-Mbyte	512-Kbyte	256-Kbyte
18	Low (0b0)				12	PM[1]	Low (0b0)	
17	12	PM[1]	Low (0b0)		13		PM [1]	Low (0b0)
16	13		PM [1]	Low (0b0)	14			PM [1]
15	14			PM [1]	15			
14	15				16			
13	Way 11[2]	Way 12 [2]	Way 13 [2]	Way 14 [2]	Way 11 [2]	Way 12 [2]	Way 13 [2]	Way 14 [2]
12	17				17			
11	18				18			
10	19				19			
9	20				20			
8	21				21			
7	22				22			
6	23				23			
5	24				24			
4	25				25			
3	26				26			
2	16				27			
1	27				28			
0	28				29			

[1] PM is high (0b1) if the transaction is to private memory space or, otherwise, low (0b0).

[2] Way *nn* is the way associated with the L2 cache tag if the transaction hits in the L2 cache or bit *nn* of the real address if the transaction is to private memory space.

8.5.5.3, 8-47 In State Meaning, replace the second to the last sentence with:

The CLK_OUT signal defaults to a clock with the same frequency as the bus clock (SYSCLK) following the assertion of $\overline{\text{HRESET}}$.

9.3.1, 9-12 Add the following sentence after the first sentence of the second paragraph of the section (first paragraph on the page):

The internally generated $\overline{\textit{abb}}$ signal is asserted in the cycle of any $\overline{\text{TS}}$ assertion on the external bus. The $\overline{\textit{abb}}$ signal remains asserted until the corresponding assertion of $\overline{\text{AACK}}$.

9.3.1.2, 9-13 In the third paragraph of the section (first paragraph on the page), replace 'internally generated $\overline{\text{ABB}}$ ...' with 'internally generated address bus busy ($\overline{\textit{abb}}$) ...'

In the fifth paragraph, add the following sentence to the end of the paragraph:

abb is asserted internally starting with TS and continuing through AACK.

In the sixth paragraph, delete two sentences beginning ‘Upon recognizing ...’ and ending with ‘... new transaction.’

9.3.1.3, 9-14 Remove the third sentence from the first paragraph:

It does, however, put more responsibility on the system arbiter; the arbiter must synthesize its own version of ABB and must not issue an address bus grant (BG) to a processor when it would cause a collision on the address bus with an address tenure from another processor.

9.3.2.2, 9-17 In Table 9-1, replace HID0[IFFT] with HID0[IFTT] in note 2.

9.3.2.4.1, 9-20 In Table 9-5, replace the last row with the following:

Misaligned—	First access	0 0 1	1 1 1	—	—	—	—	—	—	—	A
	Second access	0 1 1	0 0 0	A	A	A	—	—	—	—	—

9.3.3.1, 9-21 Add the following two sentences after the first sentence of the first paragraph:

Note that the MPC7410 is self-snooping and may assert ARTRY for its own transaction. See Section 3.9.3, “Snooping,” for more information.

9.4.1.1, 9-24 Replace the sentence ‘Note that ... TS is asserted.’ after the bulleted list with the following paragraph:

A data bus grant may be qualified as early as the clock cycle when TS is asserted if DBG is asserted during that cycle (for example, when the data bus is parked). However, the requirement remains that the first (or only) assertion of TA for the transaction must occur no sooner than two clock cycles after TS is asserted. This requirement is necessary to guarantee that the first (or only) assertion of TA will not occur before a possible ARTRY response. See Section 9.3.3.1, “Address Retry Window and Qualified ARTRY,” for more information about ARTRY signal requirements.

9.6.1.1.1, 9-38 Replace the last sentence of the last paragraph with the following:

However, as explained above, if AACK is used to delay an address tenure beyond the first cycle after TS, the bus arbiter must negate BG in every cycle that the arbiter delays AACK.

9.6.1.4, 9-43 Add the following two sentences after the second sentence of the third paragraph:

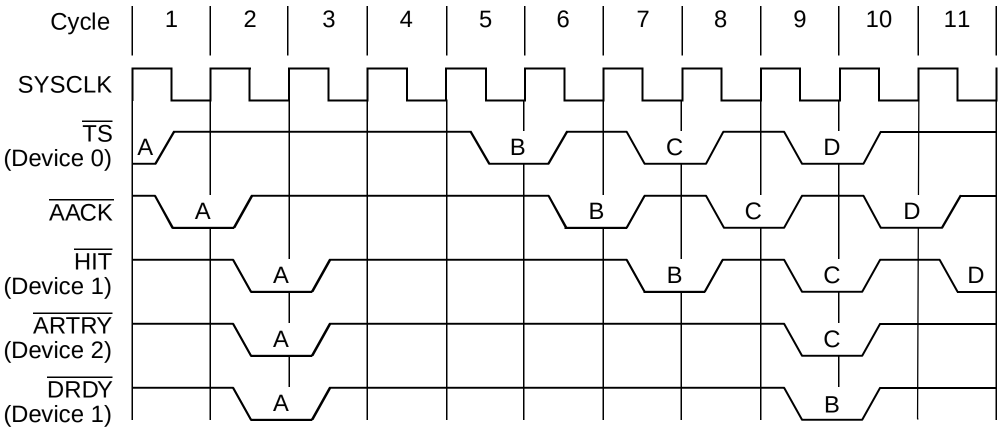
As in the 60x bus mode, MPC7410 is self-snooping and may assert ARTRY for its own transaction. See Section 3.9.3, “Snooping,” for more information.

9.6.2.1.1, 9-49 Replace the three items under the third bullet with the following two items:

— The processor is not already using the data bus, or

— The processor has a burst access in progress, the processor has already received the next-to-last TA for the current burst, and the source of the next access is the same as the source for the current access.

9.6.2.2.6, 9-55 Replace Figure 9-27 with the following ($\overline{\text{DRDY}}$ signal goes high at the middle of cycle 10 rather than cycle 11 and delete the description of cycle 10):



Cycle 2:Transaction A receives an $\overline{\text{ARTRY}}$ so the $\overline{\text{HIT}}$ and $\overline{\text{DRDY}}$ signals are ignored by the system.

Cycle 5:Transaction B starts. This is the earliest possible $\overline{\text{TS}}$ after an $\overline{\text{ARTRY}}$.

Cycle 7:Transaction B receives a $\overline{\text{HIT}}$ response. $\overline{\text{DRDY}}$ is delayed.

Cycle 9:Transaction C receives a $\overline{\text{HIT}}$ and an $\overline{\text{ARTRY}}$. The system understands that the $\overline{\text{DRDY}}$ is for transaction B and considers it valid.

Cycle 11:$\overline{\text{HIT}}$ for transaction D is asserted. The bus master must not assert $\overline{\text{DRDY}}$ for transaction C at this point since the system would interpret it as the $\overline{\text{DRDY}}$ for transaction D.

9.7.3, 9-57 Replace the first sentence of the first paragraph, ‘When either reset input negates, ... reset exception vector.’ with the following sentence:

When $\overline{\text{HRESET}}$ is negated or $\overline{\text{SRESET}}$ is asserted, the processor attempts to fetch code from the system reset exception vector.

11.4, 11-12 Replace ‘MSR[PMM]’ with ‘MSR[PM]’ in both bullets after the second paragraph as follows:

- Counting is unconditionally enabled regardless of the states of MSR[PM] and MSR[PR]. This can be accomplished by clearing MMCR0[0–4].
- Counting is unconditionally disabled regardless of the states of MSR[PM] and MSR[PR]. This is done by setting MMCR0[0] = 1. Note that SIAR is not updated if MMCR0[0] = 1.

11.5.1, 11-14 In Table 11-8, event 18 of PMC1 is missing from the table. Add the following row after event 17:

18 (001_0010)	Clean L1 cast outs to L2	Counts the number of times the L1 casts out clean data to L2.

11.5.1, 11-14 In Table 11-8, event 21 of PMC1 is missing from the table. Add the following row after event 20:

21 (001_0101)	Instruction TLB search latency over threshold	Counts when an instruction TLB search requires more than the threshold cycles to complete. Includes search operations that do not find a matching PTE and search operations caused by **dst***x* instructions. Note that MMCR2[0] determines whether the value in the MMCR0 threshold field is multiplied by 2 or 32.

11.5.1, 11-14 In Table 11-8, event 28 of PMC1, replace ‘transactions’ with ‘cycles’ in the event description as follows:

28 (001_1100)	BIU single-beat read cycles	Counts when $\overline{\text{TA}}$ generates single-beat reads

11.5.1, 11-15 In Table 11-8, event 37 of PMC1, replace ‘Count cycles ...’ with ‘Count the number of bus cycles...’ in the event description as follows:

37 (010_0101)	Speculative stalled BIU cycles	Counts the number of bus cycles in the BIU that cannot request the address bus because it is waiting for a guarded load to become nonspeculative

11.5.2, 11-16 In Table 11-9, event 5 of PMC2 is missing from the table. Add the following row after event 4:

5 (00_0101)	Fall-through branches	Counts the branches that were resolved as a not-taken branch.

11.5.2, 11-16 In Table 11-9, events 7 and 8 of PMC2 are missing from the table. Add the following two rows after event 6:

7 (00_0111)	Instructions completed in VIU1	Counts the instructions completed by the ALU’s simple integer subunit. The counter can increment by 0, 1, 2, or 3 depending on the number of completed instructions per cycle.
8 (00_1000)	Instructions completed in AltiVec ALU vector complex integer subunit	Counts the instructions completed by the AltiVec ALU complex integer subunit. The counter can increment by 0, 1, 2, or 3 depending on the number of completed instructions per cycle.

11.5.2, 11-16 In Table 11-9, event 15 of PMC2 is missing from the table. Add the following row after event 14:

15 (00_1111)	L1 data load miss	Counts speculative load attempts that missed in the L1 data cache and were queued either in the dRLT or LFQ. Does not include MMU.

11.5.2, 11-16 In Table 11-9, event 19 of PMC2, in the second sentence of the event description, replace ‘L2CR[L2DSIZ]’ with ‘L2PMCR[DBSIZ].’ Also add the following sentence to the end of the description:

19 (01_0011)	L2SRAM read cycles	Incremented for each beat of a read from the L2 SRAMs. A beat consists of 8 bytes for 64-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b00). For 32-bit L2 data bus mode (L2PMCR[DBSIZ] = 0b10), each beat consists of 4 bytes. Note that for designs that use pipeline burst SRAMs (PB2) event 19 does not count the Rdrv cycle as shown in Figure 3-36.

11.5.2, 11-16 In Table 11-9, event 21 of PMC2, replace ‘transactions’ with ‘cycles’ in the event description as follows:

21 (01_0101)	BIU multi-beat read cycles	Counts when $\overline{\text{TA}}$ is generated for multiple beat reads

11.5.2, 11-17 In Table 11-9, event 42 of PMC2, replace ‘assertions’ with ‘cycles’ in the event description as follows:

42 (10_1010)	Generate BIU $\overline{\text{TA}}$ cycles	Counts when $\overline{\text{TA}}$ generates (single-beat or multi-beat, read or write). Indicates system interface bandwidth consumed when compared to the number of elapsed bus cycles.

11.5.3, 11-19 In Table 11-10, event 29 of PMC3, replace ‘transactions’ with ‘cycles’ in the event description as follows:

29 (1_1101)	BIU single-beat write cycles	Counts when $\overline{\text{TA}}$ is generated for single-beat writes

11.5.3, 11-19 In Table 11-11, event 5 of PMC4 is missing from the table. Add the following row after event 4:

5 (0_0101)	Mispredicted branches	Counts the number of mispredicted branches

11.5.3, 11-19 In Table 11-11, events 7 and 8 of PMC4 are missing from the table. Add the following two rows after event 6:

7 (0_0111)	VFPU instructions completed	Counts instructions completed by the VFPU. 0, 1, or 2 instructions per cycle.
8 (0_1000)	VPU wait	Counts cycles that the VPU had a valid dispatch but invalid operands.

11.5.4, 11-20 In Table 11-11, event 18 of PMC4, add the followng sentence to the end of the description:

18 (1_0010)	L2SRAM cycles used	Counts L2SRAM read cycles, L2SRAM write cycles, and turnaround dead cycles required between reads and writes. Indicates L2SRAM bandwidth consumed when compared to the number of L2 clock cycles elapsed. Note that for designs that use pipeline burst SRAMs (PB2) event 18 also counts the Rdrv cycle as shown in Figure 3-36.

11.5.4, 11-20 In Table 11-11, event 25 of PMC4, replace ‘transactions’ with ‘cycles’ in the event description as follows:

25 (1_1001)	BIU multi-beat write cycles	Counts when $\overline{\text{TA}}$ is asserted for multiple beat writes

Glossary

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book. Some of the terms and definitions included in the glossary are reprinted from IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, copyright ©1985 by the Institute of Electrical and Electronics Engineers, Inc. with the permission of the IEEE.

A

Architecture. A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

Asynchronous exception. *Exceptions* that are caused by events external to the processor's execution. In this document, the term 'asynchronous exception' is used interchangeably with the word *interrupt*.

Atomic access. A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The architecture implements atomic accesses through the **lwarx**/**stwcx.** instruction pair.

B

BAT (block address translation) mechanism. A software-controlled array that stores the available block address translations on-chip.

Biased exponent. An *exponent* whose range of values is shifted by a constant (bias). Typically a bias is provided to allow a range of positive values to express a range that includes both positive and negative values.

Big-endian. A byte-ordering method in memory where the address n of a word corresponds to the *most-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the most-significant byte. See Little-endian.

Block. An area of memory that ranges from 128 Kbyte to 256 Mbyte whose size, translation, and protection attributes are controlled by the BAT mechanism.

Boundedly undefined. A characteristic of certain operation results that are not rigidly prescribed by the PowerPC ISA. Boundedly-undefined results for a given operation may vary among implementations and between execution attempts in the same implementation.

Although the architecture does not prescribe the exact behavior for when results are allowed to be boundedly undefined, the results of executing instructions in contexts where results are allowed to be boundedly undefined are constrained to ones that could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction.

Branch folding. The replacement with target instructions of a branch instruction and any instructions along the not-taken path when a branch is either taken or predicted as taken.

Branch prediction. The process of guessing whether a branch will be taken. Such predictions can be correct or incorrect; the term ‘predicted’ as it is used here does not imply that the prediction is correct (successful). The architecture defines a means for static branch prediction as part of the instruction encoding.

Branch resolution. The determination of whether a branch is taken or not taken. A branch is said to be resolved when the processor can determine which instruction path to take. If the branch is resolved as predicted, the instructions following the predicted branch that may have been speculatively executed can complete (see completion). If the branch is not resolved as predicted, instructions on the mispredicted path, and any results of speculative execution, are purged from the pipeline and fetching continues from the nonpredicted path.

Burst. A multiple-beat data transfer whose total size is typically equal to a cache block.

C

Cache. High-speed memory containing recently accessed data and/or instructions (subset of main memory).

Cache block. A small region of contiguous memory that is copied from memory into a *cache*. The size of a cache block may vary among processors; the maximum block size is one *page*. In processors that are built on Power Architecture technology, *cache coherency* is maintained on a cache-block basis. Note that the term ‘cache block’ is often used interchangeably with ‘cache line’.

Cache coherency. An attribute wherein an accurate and common view of memory is provided to all devices that share the same memory system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor’s cache.

Cache flush. An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory. This operation is generated typically by a Data Cache Block Flush (**dcbf**) instruction.

Caching-inhibited. A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

Cast-outs. *Cache blocks* that must be written to memory when a cache miss causes a cache block to be replaced.

Changed bit. One of two *page history bits* found in each *page table entry* (PTE). The processor sets the changed bit if any store is performed into the *page*. See also Page access history bits and Referenced bit.

Clean. An operation that causes a cache block to be written to memory, if modified, and then left in a valid, unmodified state in the cache.

Clear. To cause a bit or bit field to register a value of zero. See also Set.

Completion. Completion occurs when an instruction has finished executing, written back any results, and is removed from the completion queue. When an instruction completes, it is guaranteed that this instruction and all previous instructions can cause no exceptions.

Context synchronization. An operation that ensures that all instructions in execution complete past the point where they can produce an *exception*, that all instructions in execution complete in the context in which they began execution, and that all subsequent instructions are *fetched* and executed in the new context. Context synchronization may result from executing specific instructions (such as **isync** or **rfi**) or when certain events occur (such as an exception).

Copy-back. An operation in which modified data in a *cache block* is copied back to memory.

D

Data intervention. An approach used in MPX bus mode to allow data to be forwarded directly to the requesting master from the processor that has it cached. A cache-to-cache data transfer.

Denormalized number. A nonzero floating-point number whose exponent has a reserved value, usually the format's minimum, and whose explicit or implicit leading significand bit is zero.

Direct-mapped cache. A cache in which each main memory address can appear in only one location within the cache, operates more quickly when the memory request is a cache hit.

E

Effective address (EA). The 32- or 64-bit address specified for a load, store, or an instruction fetch. This address is then submitted to the MMU for translation to either a *physical memory* address or an I/O address.

Exception. A condition encountered by the processor that requires special, supervisor-level processing.

Exception handler. A software routine that executes when an exception is taken. Normally, the exception handler corrects the condition that caused the exception, or performs some other meaningful task (that may include aborting the program that caused the exception). The address for each exception handler is identified by an exception vector offset defined by the architecture and a prefix selected via the MSR.

Execution synchronization. A mechanism by which all instructions in execution are architecturally complete before beginning execution (appearing to begin execution) of the next instruction. Similar to context synchronization but doesn't force the contents of the instruction buffers to be deleted and refetched.

Exponent. In the binary representation of a floating-point number, the exponent is the component that normally signifies the integer power to which the value two is raised in determining the value of the represented number. *See also* Biased exponent.

F

Fall-through (branch fall-through). Removal of a not-taken branch.

Fetch. Retrieving instructions from either the cache or main memory and placing them into the instruction queue.

Finish. Finishing occurs in the last cycle of execution. In this cycle, the CQ entry is updated to indicate that the instruction has finished executing.

Floating-point register (FPR). Any of the 32 registers in the floating-point register file. These registers provide the source operands and destination results for floating-point instructions. Load instructions move data from memory to FPRs and store instructions move data from FPRs to memory. The FPRs are 64 bits wide and store floating-point values in double-precision format

Flush. An operation that causes a cache block to be invalidated and the data, if modified, to be written to memory.

Fraction. In the binary representation of a floating-point number, the field of the *significand* that lies to the right of its implied binary point.

G

General-purpose register (GPR). Any of the 32 registers in the general-purpose register file. These registers provide the source operands and destination results for all integer data manipulation instructions. Integer load instructions move data from memory to GPRs and store instructions move data from GPRs to memory.

Guarded. The guarded attribute pertains to out-of-order execution. When a page is designated as guarded, instructions and data cannot be accessed out-of-order.

H

Harvard architecture. An architectural model featuring separate caches for instructions and data.

Hashing. An algorithm used in the *page table* search process.

I

IEEE 754. A standard written by the Institute of Electrical and Electronics Engineers that defines operations and representations of binary floating-point numbers.

Illegal instructions. A class of instructions that are not implemented for a particular processor. These include instructions not defined by the architecture. In addition, for 32-bit implementations, instructions that are defined only for 64-bit implementations are considered to be illegal instructions. For 64-bit implementations instructions that are defined only for 32-bit implementations are considered to be illegal instructions.

Implementation. A particular processor that conforms to the architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features. The PowerPC ISA has many different implementations.

Imprecise exception. A type of *synchronous exception* that is allowed not to adhere to the precise exception model (see Precise exception). The architecture allows only floating-point exceptions to be handled imprecisely.

Instruction queue. A holding place for instructions fetched from the current instruction stream.

Integer unit. A functional unit in the MPC750 responsible for executing integer instructions.

In-order. An aspect of an operation that adheres to a sequential model. An operation is said to be performed in-order if, at the time that it is performed, it is known to be required by the sequential execution model. *See* Out-of-order.

Instruction latency. The total number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

Interrupt. An *asynchronous exception*. On processors, interrupts are a special case of exceptions. See also asynchronous exception.

K

Key bits. A set of key bits referred to as Ks and Kp in each segment register and each BAT register. The key bits determine whether supervisor or user programs can access a *page* within that *segment* or *block.*

Kill. An operation that causes a *cache block* to be invalidated without writing any modified data to memory.

L

Latency. The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.

L2 cache. *See* Secondary cache.

Least-significant bit (lsb). The bit of least value in an address, register, data element, or instruction encoding.

Least-significant byte (LSB). The byte of least value in an address, register, data element, or instruction encoding.

Little-endian. A byte-ordering method in memory where the address n of a word corresponds to the *least-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the *most-significant byte*. See Big-endian.

M

Memory access ordering. The specific order in which the processor performs load and store memory accesses and the order in which those accesses complete.

Memory-mapped accesses. Accesses whose addresses use the page or block address translation mechanisms provided by the MMU and that occur externally with the bus protocol defined for memory.

Memory coherency. An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.

Memory consistency. Refers to agreement of levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

Memory management unit (MMU). The functional unit that is capable of translating an *effective* (logical) *address* to a physical address, providing protection mechanisms, and defining caching methods.

MESI (modified/exclusive/shared/invalid). *Cache coherency* protocol used to manage caches on different devices that share a memory system. Note that the PowerPC ISA does not specify the implementation of a MESI protocol to ensure cache coherency.

Most-significant bit (msb). The highest-order bit in an address, registers, data element, or instruction encoding.

Most-significant byte (MSB). The highest-order byte in an address, registers, data element, or instruction encoding.

N

NaN. An abbreviation for not a number; a symbolic entity encoded in floating-point format. There are two types of NaNs—signaling NaNs and quiet NaNs.

No-op. No-operation. A single-cycle operation that does not affect registers or generate bus activity.

Normalization. A process by which a floating-point value is manipulated such that it can be represented in the format for the appropriate precision (single- or double-precision). For a floating-point value to be representable in the single- or double-precision format, the leading implied bit must be a 1.

O

OEA (operating environment architecture). The level of the architecture that describes PowerPC memory management model, supervisor-level registers, synchronization requirements, and the exception model. It also defines the time-base feature from a supervisor-level perspective. Implementations that conform to the OEA also conform to the UISA and VEA.

Optional. A feature, such as an instruction, a register, or an exception, that is defined by the architecture but not required to be implemented.

Out-of-order. An aspect of an operation that allows it to be performed ahead of one that may have preceded it in the sequential model, for example, speculative operations. An operation is said to be performed out-of-order if, at the time that it is performed, it is not known to be required by the sequential execution model. *See* In-order.

Out-of-order execution. A technique that allows instructions to be issued and completed in an order that differs from their sequence in the instruction stream.

Overflow. An condition that occurs during arithmetic operations when the result cannot be stored accurately in the destination register(s). For example, if two 32-bit numbers are multiplied, the result may not be representable in 32 bits.

P

Page. A region in memory. The OEA defines a page as a 4-Kbyte area of memory, aligned on a 4-Kbyte boundary.

Page access history bits. The *changed* and *referenced* bits in the PTE keep track of the access history within the page. The referenced bit is set by the MMU whenever the page is accessed for a read or write operation. The changed bit is set when the page is stored into. See Changed bit and Referenced bit.

Page fault. A page fault is a condition that occurs when the processor attempts to access a memory location that does not reside within a *page* not currently resident in *physical memory.* On processors, a page fault exception condition occurs when a matching, valid *page table entry* (PTE[V] = 1) cannot be located.

Page table. A table in memory is comprised of *page table entries*, or PTEs. It is further organized into eight PTEs per PTEG (page table entry group). The number of PTEGs in the page table depends on the size of the page table (as specified in the SDR1 register).

Page table entry (PTE). Data structures containing information used to translate *effective address* to physical address on a 4-Kbyte page basis. A PTE consists of 8 bytes of information in a 32-bit processor and 16 bytes of information in a 64-bit processor.

Physical memory. The actual memory that can be accessed through the system's memory bus.

Pipelining. A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

Precise exceptions. A category of exception for which the pipeline can be stopped so instructions that preceded the faulting instruction can complete, and subsequent instructions can be flushed and redispatched after exception handling has completed. *See* Imprecise exceptions.

Primary opcode. The most-significant 6 bits (bits 0–5) of the instruction encoding that identifies the type of instruction.

Program order. The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the cache

Protection boundary. A boundary between *protection domains*.

Protection domain. A protection domain is a segment, a virtual page, a BAT area, or a range of unmapped effective addresses. It is defined only when the appropriate relocate bit in the MSR (IR or DR) is 1.

Q

Quiesce. To come to rest. The processor is said to quiesce when an exception is taken or a **sync** instruction is executed. The instruction stream is stopped at the decode stage and executing instructions are allowed to complete to create a controlled context for instructions that may be affected by out-of-order, parallel execution. *See* Context synchronization.

Quiet NaN. A type of *NaN* that can propagate through most arithmetic operations without signaling exceptions. A quiet NaN is used to represent the results of certain invalid operations, such as invalid arithmetic operations on infinities or on NaNs, when invalid. *See* Signaling NaN.

R

rA. The **r**A instruction field is used to specify a GPR to be used as a source or destination.

rB. The **r**B instruction field is used to specify a GPR to be used as a source.

rD. The **r**D instruction field is used to specify a GPR to be used as a destination.

rS. The **r**S instruction field is used to specify a GPR to be used as a source.

Real address mode. An MMU mode when no address translation is performed and the *effective address* specified is the same as the physical address. The processor's MMU is operating in real address mode if its ability to perform address translation has been disabled through the MSR registers IR and/or DR bits.

Record bit. Bit 31 (or the Rc bit) in the instruction encoding. When it is set, updates the condition register (CR) to reflect the result of the operation.

Referenced bit. One of two *page history bits* found in each *page table entry* (PTE). The processor sets the *referenced bit* whenever the page is accessed for a read or write. See also Page access history bits.

Register indirect addressing. A form of addressing that specifies one GPR that contains the address for the load or store.

Register indirect with immediate index addressing. A form of addressing that specifies an immediate value to be added to the contents of a specified GPR to form the target address for the load or store.

Register indirect with index addressing. A form of addressing that specifies that the contents of two GPRs be added together to yield the target address for the load or store.

Rename register. Temporary buffers used by instructions that have finished execution but have not completed.

Reservation. The processor establishes a reservation on a *cache block* of memory space when it executes an **lwarx** instruction to read a memory semaphore into a GPR.

Reservation station. A buffer between the dispatch and execute stages that allows instructions to be dispatched even though the results of instructions on which the dispatched instruction may depend are not available.

Retirement. Removal of the completed instruction from the CQ

RISC (reduced instruction set computing). An *architecture* characterized by fixed-length instructions with nonoverlapping functionality and by a separate set of load and store instructions that perform memory accesses.

S

Secondary cache. A cache memory that is typically larger and has a longer access time than the primary cache. A secondary cache may be shared by multiple devices. Also referred to as L2, or level-2, cache.

Sector. A 32-byte cache block.

Set (*v*). To write a nonzero value to a bit or bit field; the opposite of *clear*. The term 'set' may also be used to generally describe the updating of a bit or bit field.

Set (*n*). A subdivision of a *cache*. Cacheable data can be stored in a given location in any one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache block* corresponding to that address was used least recently. *See* Set-associative.

Set-associative. Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

Signaling NaN. A type of *NaN* that generates an invalid operation program exception when it is specified as arithmetic operands. *See* Quiet NaN.

Significand. The component of a binary floating-point number that consists of an explicit or implicit leading bit to the left of its implied binary point and a fraction field to the right.

Simplified mnemonics. Assembler mnemonics that represent a more complex form of a common operation.

Slave. The device addressed by a master device. The slave is identified in the address tenure and is responsible for supplying or latching the requested data for the master during the data tenure.

Snarfing. When one device provides data specifically for another device and a third device samples the data for its own purposes.

Snooping. Monitoring addresses driven by a bus master to detect the need for coherency actions.

Snoop push. Response to a snooped transaction that hits a modified cache block. The cache block is written to memory and made available to the snooping device.

Split-transaction. A transaction with independent request and response tenures.

Split-transaction bus. A bus that allows address and data transactions from different processors to occur independently.

Stage. The term ‘stage’ is used in two different senses, depending on whether the pipeline is being discussed as a physical entity or a sequence of events. In the latter case, a stage is an element in the pipeline during which certain actions are performed, such as decoding the instruction, performing an arithmetic operation, or writing back the results. Typically, the latency of a stage is one processor clock cycle. Some events, such as dispatch, write-back, and completion, happen instantaneously and may be thought to occur at the end of a stage. An instruction can spend multiple cycles in one stage. An integer multiply, for example, takes multiple cycles in the execute stage. When this occurs, subsequent instructions may stall. An instruction may also occupy more than one stage simultaneously, especially in the sense that a stage can be seen as a physical resource—for example, when instructions are dispatched they are assigned a place in the CQ at the same time they are passed to the execute stage. They can be said to occupy both the complete and execute stages in the same clock cycle.

Stall. An occurrence when an instruction cannot proceed to the next stage.

Static branch prediction. Mechanism by which software (for example, compilers) can hint to the machine hardware about the direction a branch is likely to take.

Static memory. Memory that assumes a reasonable degree of locality and that the data is needed several times over a relatively long period.

Superscalar machine. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

Supervisor mode. The privileged operation state of a processor. In supervisor mode, software, typically the operating system, can access all control registers and can access the supervisor memory space, among other privileged operations.

Synchronization. A process to ensure that operations occur strictly *in order*. *See* Context synchronization and Execution synchronization.

Synchronous exception. An *exception* that is generated by the execution of a particular instruction or instruction sequence. There are two types of synchronous exceptions, *precise* and *imprecise*.

System memory. The physical memory available to a processor.

T

Tenure. A tenure consists of three phases: arbitration, transfer, termination. There can be separate address bus tenures and data bus tenures.

TLB (translation lookaside buffer). A cache that holds recently-used *page table entries*.

Throughput. The measure of the number of instructions that are processed per clock cycle.

Transaction. A complete exchange between two bus devices. A transaction is typically comprised of an address tenure and one or more data tenures, which may overlap or occur separately from the address tenure. A transaction may be minimally comprised of an address tenure only.

Transfer termination. Signal that refers to both signals that acknowledge the transfer of individual beats (of both single-beat transfer and individual beats of a burst transfer) and to signals that mark the end of the tenure.

Transient memory. Memory that has poor locality and is likely to be referenced very few times or over a very short period of time.

U

UISA (user instruction set architecture). The level of the architecture to which user-level software should conform. The UISA defines the base user-level instruction set, user-level registers, data types, floating-point memory conventions and exception model as seen by user programs, and the memory and programming models.

Underflow. A condition that occurs during arithmetic operations when the result cannot be represented accurately in the destination register. For example, underflow can happen if two floating-point fractions are multiplied and the result requires a smaller *exponent* and/or mantissa than the single-precision format can provide. In other words, the result is too small to be represented accurately.

User mode. The operating state of a processor used typically by application software. In user mode, software can access only certain control registers and can access only user memory space. No privileged operations can be performed. Also referred to as problem state.

V

VEA (virtual environment architecture). The level of the *architecture* that describes the memory model for an environment in which multiple devices can access memory, defines aspects of the cache model, defines cache control instructions, and defines the time-base facility from a user-level perspective. *Implementations* that conform to the VEA also adhere to the UISA, but may not necessarily adhere to the OEA.

Virtual address. An intermediate address used in the translation of an *effective address* to a physical address.

Vector. The spatial parallel processing of short, fixed-length, one-dimensional matrices performed by an execution unit.

Virtual memory. The address space created using the memory management facilities of the processor. Program access to virtual memory is possible only when it coincides with *physical memory.*

W

Way. A location in the cache that holds a cache block, its tags and status bits.

Word. A 32-bit data element.

Write-back. A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache block is *cast out* to make room for newer data.

Write-through. A cache memory update policy in which all processor write cycles are written to both the cache and memory.

Index