IOP 480 Data Book

PLX TECHNOLOGY®
THE I/O INTERCONNECT SOLUTION

IOP 480 Data Book

Revision 2.0

July 2000

Website: http://www.plxtech.com
Email: apps@plxtech.com
Phone: 408 774-9060
800 759-3735
Fax: 408 774-2169

Order Number: IOP 480-SIL-DB-P1-2.0

Printed in the USA, July 2000

Contents

24. IOP 480 CPU Programming Model 24-1

Figures

Tables

Registers

Timing Diagrams

PREFACE

The information contained in this document is subject to change without notice. Although an effort has been made to keep the information accurate, there may be misleading or even incorrect statements made herein.

SUPPLEMENTAL DOCUMENTATION

The following is a list of additional documentation to provide the reader with further information:

- *PCI Local Bus Specification, Revision 2.2*, December 18, 1998
  PCI Special Interest Group (PCI SIG)
  5440 SW Westgate Drive #217, Portland, OR 97221 USA
  Tel: 800 433-5177 (domestic only) or 503 693-6232, Fax: 503 693-8344, http://www.pcisig.com
- *PCI Hot-Plug Specification, Revision 1.0*
  PCI Special Interest Group (PCI SIG)
  5440 SW Westgate Drive #217, Portland, OR 97221 USA
  Tel: 800 433-5177 (domestic only) or 503 693-6232, Fax: 503 693-8344, http://www.pcisig.com
- *PCI Bus Power Management Interface Specification, Revision 1.1*, December 18, 1998
  PCI Special Interest Group (PCI SIG)
  5440 SW Westgate Drive #217, Portland, OR 97221 USA
  Tel: 800 433-5177 (domestic only) or 503 693-6232, Fax: 503 693-8344, http://www.pcisig.com
- *PICMG 2.1, CompactPCI Hot Swap Specification, Revision 1.0*, August 3, 1998
  PCI Industrial Computer Manufacturers Group (PICMG)
  c/o Virtual Inc., 401 Edgewater Place, Suite 500, Wakefield, MA 01880, USA
  Tel: 781 224-1100, Fax: 781 224-1239, http://www.picmg.org
- *Intelligent I/O (I_2O) Architecture Specification, Revision 2.0*, May 1999
  I_2O Special Interest Group (I_2O SIG)
  404 Balboa Street, San Francisco, CA 94118, USA
  Tel: 415 750-8352, Fax: 415 751-4829, http://www.i2osig.org

TERMS AND DEFINITIONS

- Direct Master

  External Local Bus Master initiates Data write/read to/from the PCI Bus

- Direct Slave

  External PCI Bus Master initiates Data write/read to/from the Local Bus

Data Assignment Conventions

Data Width	IOP 480 Convention
1 byte (8 bits)	Byte
2 bytes (16 bits)	Word
4 bytes (32 bits)	Lword
8 bytes (64 bits)	Qword

REVISION HISTORY

Date	Revision	Comment
7/30/1997	0.1	Red Book initial release.
3/6/1998	0.2	Revised Red Book to include detailed core and peripheral specifications.
4/15/1998	0.3	Revised Red Book to separate IBM and PLX material.
7/8/1998	0.31	Revised Red Book to combine IBM and PLX materials and incorporate engineering changes.
12/7/1998	0.90	Blue Book initial release. Incorporate Red Book engineering changes.
1/27/1999 – 2/12/1999	0.92	Blue Book update.
7/1999	0.96	Blue Book Update.
10/1999	1.0	Initial Release.
7/12/2000	2.0	Initial release Revision 2.0.

IOP 480

Integrated PowerPC I/O Processor

July 2000
Revision 2.0

Incorporates Advanced PCI Functionality, Memory Controller and PowerPC Processor for Adapters and Embedded Systems

1 INTRODUCTION

1.1 HIGHLIGHTS

- PLX industry-leading advanced Data Pipe Architecture™ technology PCI controller with dual DMA channels, programmable Direct Master and Direct Slave data transfer modes, and PCI messaging functions
- Local Bus DMA channel with Flyby DMA support
- 66 MHz 32-Bit PowerPC RISC CPU Core
- 32-bit PCI interface operating up to 33 MHz
- Flexible Memory controller offering support of up to 256 MB of SDRAM or EDO DRAM of up to 66 MHz
- PCI Bus Arbiter supports up to three external PCI Masters and high priority mode Local Bus Arbiter supports two external Local Bus Masters and high priority mode
- Programmable interrupt controller with multiple timers and counters
- I_2O-*Ready* Messaging Unit with r2.0 performance extensions
- *PCI Specification r2.2* Power Management features
- PCI Hot Plug compatible
- CompactPCI® Hot Swap *Friendly*
- PCI Dual Address Cycle (DAC) support
- 32-bit Multiplexed Local Bus up to 66 MHz supports 8-, 16-, or 32-bit peripheral and memory devices with unlimited bursting up to 264 MB/s
- Debug Serial Port (Tx/Rx)
- IEEE 1149.1 JTAG port for test and debug
- 3.3V, 5V tolerant PCI and Local signaling supports universal PCI adapter designs
- 3.3V Core CMOS in 208-pin PQFP and 225-pin PBGA
- Industrial Temp Range operation

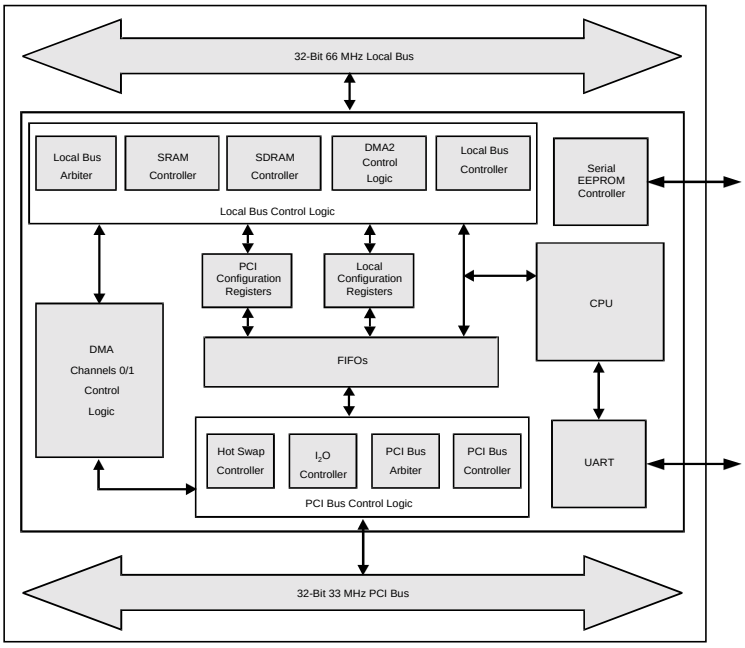


Figure 1-1. IOP 480 Block Diagram

1.2 FEATURES

1.2.1 PowerPC RISC Processor Core

- 66 MHz PowerPC RISC 32-Bit CPU core
- Thirty-two, 32-bit general purpose registers
- Code compatible with PowerPC User Instruction Set Architecture and Development Tools
- Separate 4 KB Instruction cache and Write-Back/ Write-Through 2 KB Data cache
- Hardware multiply and divide
- On-Chip clock generation and power management up to 66 MHz CPU core and 66 MHz Local Bus
- Four timers (64-bit time base, programmable interval timer, fixed interval timer, and watchdog timer)
- Memory Management Unit

1.2.2 On-Chip Peripheral Logic

1.2.2.1 Advanced Data Pipe Architecture

- *PCI Specification r2.2*-compliant 32-bit, 33 MHz Bus Master Interface Controller with PCI Power Management features for adapters and embedded systems
- Data Pipe Architecture technology includes two DMA engines, programmable Target and Initiator Data Transfer modes and PCI messaging functions
- Dual Independent DMA channels with flexible prioritization scheme
  - Direct hardware control of DMA
    - Demand mode DMA operation
    - Burst length control–BTERM
    - End of transfer (EOT)
  - Programmable burst length using thresholds including unlimited burst
  - Shuttle mode DMA channel support with automatic invalidation of used DMA descriptors
  - Unaligned transfer support
  - Supports PCI Bus Mastering from Local Slave-Only devices (and vice-versa)
  - Scatter/Gather List/Ring management
    - Descriptors can be found in PCI Bus memory or in Local Bus memory
    - Automatic polling for valid descriptors (PCI or Local Bus)
    - Automatically returns transfer count upon hardware (EOT) DMA termination
- Local-to-Local DMA
  - Flyby I/O-to-Memory transfers between Local devices
  - DMA initiated Burst reads and writes on the Local Bus (memory-to-memory)
- Direct Master Mode [1]
  - Type 0 and Type 1 configuration cycles
  - Supports all PCI cycle types, including full support for Memory Write and Invalidate (MWI) cycles
  - Initiator READ prefetching
  - Burst length control
  - Unaligned transfer control
  - Endian swapping
- Direct Slave Mode
  - Multiple independent address spaces
  - Dynamic Local Bus width control
  - Target READ prefetching
  - Endian swapping
  - Local Bus priority control
  - PCI Latency Timer
  - Eight Local Bus chip selects
  - Supports both Memory-Mapped and I/O-Mapped Burst accesses from PCI-to-Local
- PCI Messaging
  - Incorporates an I_2O-*Ready* Messaging Unit, which is fully compatible with the PCI extensions of *I_2O Architecture Specification r2.0* (Pull and Outbound option)
  - Complete messaging unit with two Doorbell registers, and eight Mailbox registers with interrupt capability
  - Automatically updating Queue Management pointers, which can be used for message passing under the I_2O protocol or a custom protocol

1. Although *PCI Local Bus Specification*, *Revision 2.2* utilizes "PCI Initiator" for "Direct Master" and "PCI Target" for "Direct Slave," PLX Technology, Inc., continues to utilize "Direct Master" and "Direct Slave."

- Vital Product Data Support—Fully supports the Vital Product Data (VPD) PCI extension, which provides an alternate access method other than Expansion ROM for VPD. It is updated for distinguishing identical system boards.
- PCI Dual Address Cycle (DAC) support (64-Bit Address Space)—Supports PCI DAC beyond the 4-GB Address space, which can be used during IOP 480 Bus Master operations (DMA, Direct Master).
- PCI ↔ Locally-sustained data transfers up to 132 MB/s.
- Six FIFOs for zero wait-state burst in Master, Target, and DMA modes (32 words deep for writes, and 16 words deep for reads).
- PCI Configuration Cycles—In Direct Master mode, the PCI controller can generate Type 0 and Type 1 PCI Configuration cycles to enable configuration of other PCI devices or cards in the system.
- Interrupt Generator—The PCI controller can generate PCI and Local interrupts to the internal interrupt controller from several sources, including PCI Interrupt (INTA#), System Error (SERR#), and Parity Error (PERR#).
- Asynchronous Bus Clocks—The Local Bus interface runs from a Local system clock and generates the necessary internal clocks. This clock is capable of running asynchronously to the PCI clock.
- The IOP 480 requires 3.3V Vcc and provides 3.3V signaling with 5V I/O tolerance on both the PCI and Local Buses. It can support universal PCI adapter designs.
- Serial EEPROM Interface—The IOP 480 contains a serial EEPROM interface, which can optionally be used to load configuration information. This is useful for loading information unique to a particular adapter (such as, Device ID or Vendor ID).

1.2.2.2 Memory Controller Interface

- 66 MHz Memory-Bus capability
- Programmable timing, supporting either SDRAM, EDO DRAM, SRAM, parallel EEPROM, FIFO, Flash memory, and/or I/O peripheral chips
- High-Bandwidth bus (32-bit data bus)

1.2.2.3 Arbiters

- PCI Bus arbiter supports up to three external Masters
- Local Bus arbiter supports up to two external Masters
- High priority modes supported by both arbiters enables deterministic transfer

1.2.3 JTAG Interface

- IEEE 1149.1 JTAG Boundary Scan interface
- Used for Testing and PowerPC debug

1.2.4 Programmable Interrupt Controller

- Critical Interrupt input (from external devices)
- External Interrupt input (from external devices)
- Timer and debug event interrupts
- Interrupts programmable as either active-high or active-low, and maskable

1.2.5 Local Bus Interface

- Multiplexed 32-bit external bus operates up to 66 MHz
- Compatible with industry-leading DSPs, RISC processors, and a wide variety of I/O and memory devices
- Supports 8-, 16-, or 32-bit peripherals
- Big-Endian or Little-Endian device attachment
- Programmable wait states
- Up to four external programmable peripherals/ memory regions

1.2.6 Programmable Chip Selects

- Provides chip select pins for up to four external I/O or memory-mapped devices (non-SDRAM) connected to the Local Bus
- Each chip select is programmable for bus width and wait states

1.2.7 Serial Port

- Debug Serial UART port for communications with serial devices
- TTL-level RX and TX signals
- Used for controller debug or attaching external serial devices

1.2.8 Data Transfer Mechanisms

The IOP 480 supports nine data transfer mechanisms:

- **Direct Master**—Local External Master initiates cycle to Direct Slave
- **Direct Slave**—PCI Master initiates cycle to Local Slave, perhaps memory
- **Local Bus Master**—Access to internal registers
- **PCI Bus**—Access to:
  - Local Configuration/Runtime registers
  - PCI Configuration registers
  - I_2O Message queues
- **DMA**—Local-to-PCI**,** PCI-to-Local
- **CPU**—Access to:
  - SPU
  - Direct Slave
  - Local Slave
  - All internal registers
- **DMA2**—Local Slave-to-Local Slave (Flyby or Non-Flyby)
- **VPD**—Access to serial EEPROM
- **External Local Master**—Access to External Local Slave (as we now own the Local Bus)

1.3 COMPANY AND PRODUCT BACKGROUND

PLX Technology, Inc., the world leader in PCI-to-Local Bus I/O accelerator chips, supports more than 500 OEM customers in a wide variety of PCI applications. Customer applications include PC workstations and servers, PCI add-in boards, embedded PCI communication systems (such as routers and switches), and industrial PCI implementations (such as CompactPCI, PMC, and Passive Backplane PCI).

PLX Technology, Inc., is an active participant in industry standard committees, including the PCISIG®, I_2O SIG®, and PICMG®, and maintains active developer technology and cross-marketing partnerships with industry leaders, such as Intel, IBM, Hewlett-Packard, Motorola, Integrated Systems, WindRiver, and others.

Focused on providing complete solutions for PCI implementations, PLX provides design assistance to customers in the form of Reference Design kits and Software Development kits. Depending upon the application, these kits may include reference boards, API libraries, software debug tools, and sample device drivers with source, enabling customers to quickly bring new designs to production. New tools, application notes, FAQs, and information updates are constantly added to our website (www.plxtech.com) for the convenience of PLX customers. Our expertise and total solutions for the PCI interface allow customers to focus on adding value in their designs without worrying about the complexities of implementing PCI, I_2O, and CompactPCI.

1.3.1 IOP 480 I/O Processor General Description

The IOP 480 is designed to support a new class of products that are used in the embedded space—the I/O processor (IOP). Design requirements driving the development of the IOP include the following:

- Distributed processing architecture (off-load Host processor)
- High-Speed PCI system bus
- High-Performance, low-cost processor
- High-Performance burst mode data transfers
- Intelligent message passing support
- Efficient I/O transaction management
- Performance scalability

The IOP 480 Integrated PowerPC I/O Processor is also designed for applications where cost, space, power consumption and performance are all equally critical. The IOP 480 provides a high level of integration, reducing chip count from five chips to one, thereby significantly reducing system component cost. The high integration of the IOP 480 results in a simplified board design, less power consumption and faster time-to-market solution. This cost-effective, general-purpose integrated processor targets system PCI interfaces in networking, telecommunications and other embedded markets. The IOP 480 can be used for control purposes in applications such as routers, switches, network storage applications, and image display systems.

The IOP 480 is a high-performance, low-cost, low-power, integrated I/O processor (IOP). It combines a PowerPC RISC CPU, SDRAM/SRAM/EDO Memory Controller, and a *PCI r2.2*-compliant Bus Master Controller to enable the development of low-cost

intelligent PCI adapters and embedded Host PCI systems.

The IOP 480 CPU is a PowerPC core with integrated 4 KB instruction cache and 2 KB data cache. It is code-compatible with other members or the PowerPC 401xx, 403xx, and 60x processor families.

The IOP 480 PCI Bus Master design is based on the industry-standard PLX PCI 9054 I/O Accelerator device. Additional enhancements to the PCI 9054 design include the following:

- DMA Ring management
- Local-to-Local DMA mode
- A Memory controller
- PCI and Local Bus arbiters
- Additional enhancements to the I_2O-*Ready* messaging unit

At the heart of the IOP 480 Bus Master is the PLX proprietary Data Pipe Architecture technology.

Many high-performance designs are adopting the PCI standard for use in their I/O subsystems. However, implementations vary, which can lead to inefficiencies. The PLX Data Pipe Architecture technology is tuned for PCI efficiency. Its capabilities include significantly reduced management and housekeeping overhead and intelligent DMA logic for scatter/gather protocols.

1.3.2 Development Tool Support

The IOP 480 is supported by a wide variety of development tools, including the IBM High C/C++ Compiler, IBM RISCWatch emulator, third party compilers, debuggers, Real Time Operating Systems, and other tools available through the PLX Partners Program.

The IOP 480 is fully compatible with PLX PCI SDK and I_2O SDK software development kits, which allow quick and easy development of high performance Local and host PCI software through standard APIs, I_2O messaging protocols, PCI debug tools, and example device drivers.

IOP 480 design support is provided through Reference Design Kits (RDK) which provide a flexible PCI development board, complete with Orcad schematics, documentation, and software. Simulation models for the IOP 480 are also available.

1.4 APPLICATIONS

1.4.1 PCI Adapter Cards

Major PCI adapter card applications for the IOP 480 include high performance communications, networking, disk control, multimedia and video adapters. The IOP 480 moves data between the host PCI Bus and the adapter Local Bus in several ways. First, the host processor may program the IOP 480 DMA controller to move data between the adapter memory and the host PCI Bus. Second, the IOP 480 can perform “Direct Master transfers,” whereby a Local mastering device accesses the PCI Bus directly through a PCI Master transfer. The IOP 480 supports slave (target) transfers in which another PCI device is the master. The IOP 480 also has a complete messaging unit with mailbox registers, doorbell registers, and queue management pointers, which can be used for message passing under the I_2O protocol or a custom protocol.

Adapter cards are the primary vehicle for I/O Processing and Host processor off-loading. (Refer to Figure 1-2.)

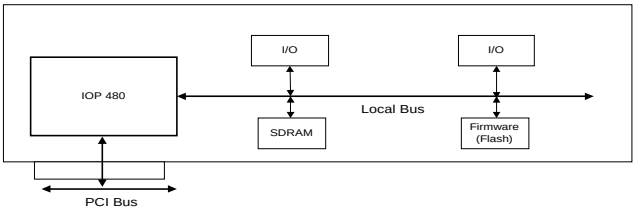


Figure 1-2. Sample PCI I/O Processor Adapter Design

1.4.2 PCI Host Embedded Systems

Another application for the IOP 480 lies in PCI host embedded systems, such as network hubs and routers, printer engines, set top boxes, and industrial equipment. In this configuration, all four of the above-mentioned data transfer modes are used. In addition, the IOP 480 supports Type 0 and Type 1 PCI configuration cycles. This allows the IOP 480 to configure the other PCI devices or cards in the system. The IOP 480 provides a PCI Bus arbiter with support for up to three PCI Bus master devices or PCI slots. (Refer to Figure 1-3.)

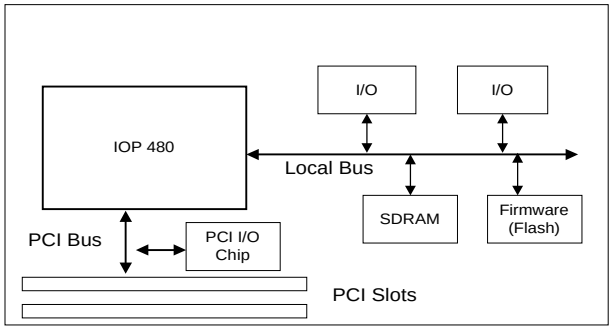


Figure 1-3. Sample PCI Host Embedded System Design

1.4.3 High-Performance PCI I_2O Design

As a member of the I_2O SIG, PLX helped define the I_2O specification, and was the first to offer a processor-independent I_2O implementation in the PCI 9080. The IOP 480 is the first second-generation IOP, incorporating the latest I_2O and PCI performance enhancements. Applications include high-performance storage controllers (RAID), and network interface cards (10/100BaseT, ATM).

I_2O also provides an efficient solution for embedded designs. I_2O takes advantage of the PCI performance features, while providing a level of abstraction from both the host operating system and I/O subsystem. The I_2O design simplifies the upgrade path of the user's product to take advantage of future hardware and software performance enhancements. (Refer to Figure 1-4 and Figure 1-5.)

The IOP 480 incorporates the Pull and Outbound option, as specified in *I_2O Specification r2.0*. These options aid the development of Write-Only architecture.

Advantages:

- Split driver model
- Model coprocessor implementation
- Facilitates peer-to-peer transfers
- Manages batching interrupts
- Aides write-only architectures
- Aides Local Bus balancing

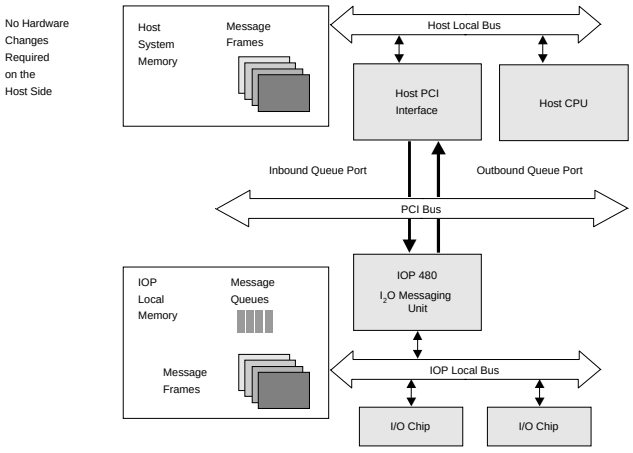


Figure 1-4. Typical I_2O Server/Adapter Card Design

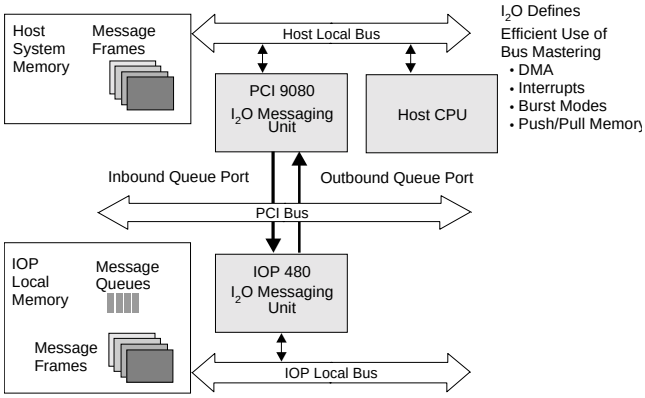


Figure 1-5. Typical I_2O Embedded System Design

1.4.4 High-Performance CompactPCI Adapter Design

Another key application for the IOP 480 is CompactPCI adapters for telecom and networking applications. These applications include high performance communications, such as WAN/LAN controller cards, high-speed modem cards, Frame Relay cards, and telephony cards for telecom switches and remote-access systems.

The IOP 480 has integrated key features to enable live-insertion of Hot Swap CompactPCI adapters. The IOP 480 *PICMG 2.1 r1.0*-compatible Hot Swap *Friendly* PCI interface includes both Hot Swap *Capable* and Hot Swap *Friendly* features. (Refer to Figure 1-6.)

1.4.4.1 Hot Swap Capable

- *PCI Specification r2.1* or better
- Tolerant of Vcc from early power
- Tolerant of asynchronous reset
- Tolerant of precharge voltage
- Limited I/O pin leakage at precharge voltage

1.4.4.2 Hot Swap Friendly

- Incorporates the Hot Swap Control/Status register (HSCSR)
- Incorporates an Extended Capability Pointer (ECP) mechanism
- Incorporates added resources for software control of ENUM#, the ejector switch, and the status LED, which indicates insertion and removal to the user

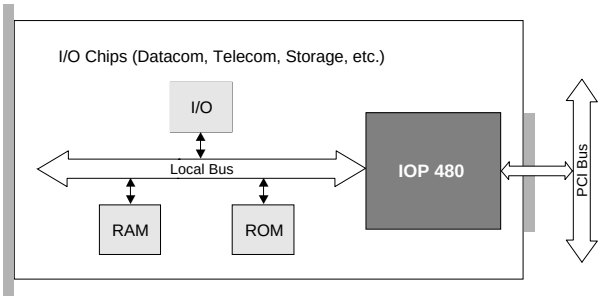


Figure 1-6. High-Performance CompactPCI Adapter

1.4.5 Real Time Application Design

To face the problems of real-time applications on the PCI Bus, The IOP 480 provides a high-priority mode. The PCI Bus provides a 132 MB/s of bandwidth. However, there is no built-in mechanism for a particular application to have a consistent slice of that bandwidth. If a plug-in card is gathering real-time data, even at relatively slow rates, large buffers are required on board to guarantee that the none of the data is dropped. Dropped data may appear as missed frames in a movie or clicks in an audio stream. (Refer to Figure 1-7.)

To address this problem, the IOP 480's arbiters can be put into a High-Priority mode. Instead of using the standard fairness algorithm, the IOP 480 can be set up to be deliberately biased towards a particular transfer. If the internal DMA Channels (Channels 0 and 1) are used to transfer data and the built-in local and PCI arbiters are set for high priority (and used), the IOP 480 delivers a guaranteed minimum bandwidth. This amount of guaranteed bandwidth depends on several factors. The burst capability of the PCI and local targets, the relative speeds of the PCI and Local Buses, and the Bus Latency Timer settings all contribute to this guaranteed bandwidth. The number and traffic patterns of other masters on either bus do not effect this bandwidth. A 33 MHz system capable of bursts up to 16 words can easily achieve a 100 MB/s guaranteed minimum bandwidth with the IOP 480. (Refer to Figure 1-7.)

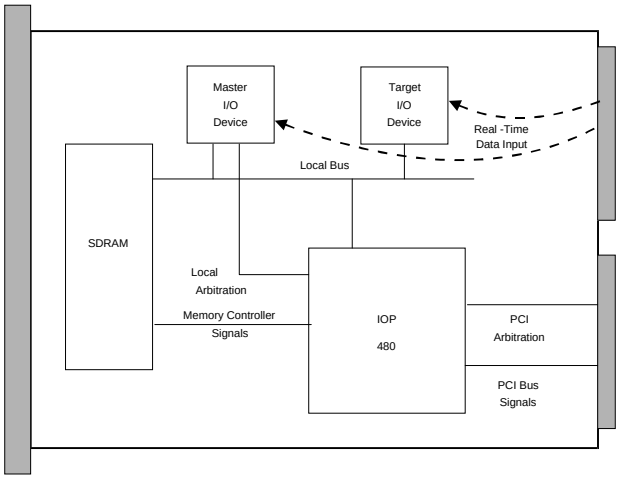


Figure 1-7. Sample Real-Time Application

1.4.6 Data Communications Design

In many datacom and telecom applications, the IOP 480 is tasked to work in conjunction with a processor (either in the IOP itself or externally) to transfer data to and from I/O chips. This usually means that the processor would set up DMA descriptor chains for both Transmit and Receive operations. To occupy a fixed block of memory, the tails of these chains are made to point back to the head, resulting in descriptor rings. The IOP 480 incorporates ring management specifically for this type of application. (Refer to Figure 1-8.)

The rings and the I/O chips may reside either on the PCI Bus or on the Local Bus. One of the IOP's DMA channels would be set for Local-to-PCI transfers and the other for PCI to local transfers (for separate transmit and receive rings). The IOP utilizes a valid bit in each DMA descriptor link to keep track of its location in the ring sequence. This Valid bit is automatically invalidated at the completion of each descriptor link. As the IOP circles through the rings, the managing processor can update the links and then again validate

them. If the IOP 480 reaches an invalidate link, it waits for that link to become valid before processing.

In this manner, both the IOP and the managing processor can run at their fastest possible speed without interrupts. The end of a packet would be signaled to the IOP via the EOT# (end of transfer) pins. When the IOP encounters an EOT, instead of simply invalidating the current descriptor link, it writes the number of words remaining to be transferred (in the current link) to the current descriptor link location, giving an idea of packet size.

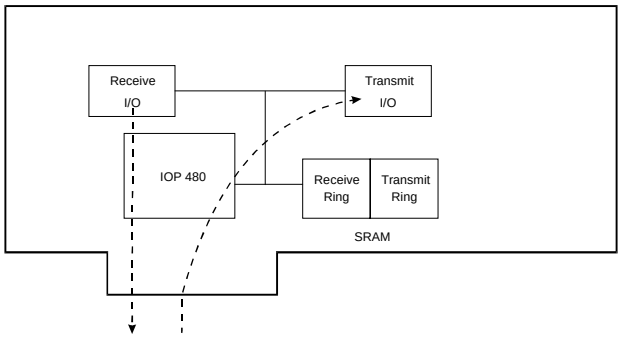


Figure 1-8. Data Communication Design

2 LOCAL BUS INTERFACE

2.1 INTRODUCTION

The Local Bus provides a data path between the PCI Bus and non-PCI devices, such as the IOP 480 CPU, memory devices, and peripherals. The Local Bus is 32-bit multiplexed bus, with bus memory regions that can be programmed for 8-, 16-, or 32-bit widths. The IOP 480 Local Bus is signal-compatible with popular RISC and Bridge architecture, including the i960Jx, PPC401 GF, and J Mode of the PCI 9080 or PCI 9054.

The IOP 480 acts as a master and slave on the Local Bus. When the IOP 480 is acting as a Local Bus slave, an external master on the Local Bus can access all internal configuration registers, as well as perform Direct Master accesses to the PCI Bus. The IOP 480 accepts burst transfers at a maximum rate of one word per clock cycle. External Local Bus masters that access the IOP 480 must have a 32-bit data bus, and treat the IOP 480 as a 32-bit slave device.

When the IOP 480 is acting as a Local Bus master, the Direct Slave controller, internal DMA controllers, or internal IOP 480 CPU can transfer data between the Local Bus, internal registers and FIFOs. Burst lengths are limited to four Lwords when the IOP 480 CPU controls the bus, and to memory page boundaries when DMA channels or a Direct Slave controller are bursting data. The width of the bus depends upon the Local Address Space accessed. There are four SRAM spaces, one DRAM space, and one default space. Each space contains a set of configuration registers that determine all Local Bus characteristics when that space is accessed.

2.1.1 Transactions

Four types of transactions can occur on a Local Bus:

- Read
- Write
- Read Burst
- Write Burst

A Bus access is a transaction which is bounded by the assertion of ADS# or ALE at the beginning and de-assertion of BLAST# at the end. A Bus access consists of an address cycle followed by one or more data transfers. During each clock cycle of an access, the Local Bus is in one of four basic states defined in Section 2.1.2, “Basic Bus States.” A clock cycle consists of one period of the Local Bus clock.

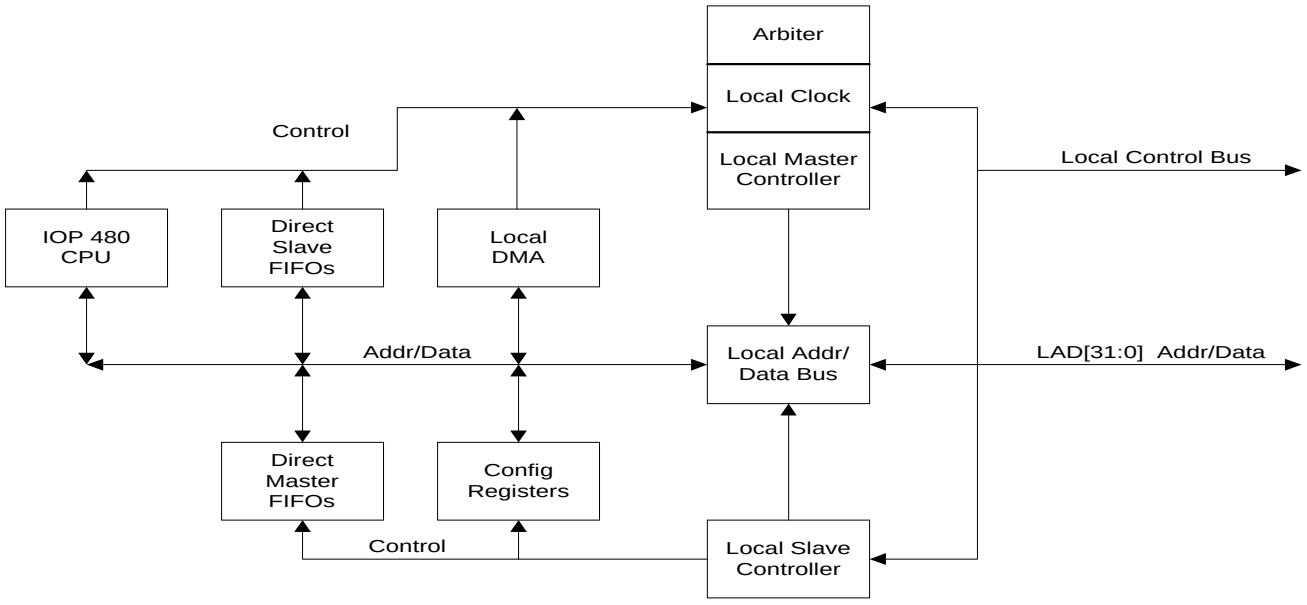


Figure 2-1. Local Bus Block Diagram

2.1.2 Basic Bus States

The four basic bus states are idle, address, data/wait, and recovery. Once the Local Bus master owns the Bus and needs to start a bus access, the address state is entered, ADS# or ALE is asserted, and a valid address is presented on the address/data bus. Data is then transferred while in a data/wait state. READY# or WAIT# is used to insert wait states. BLAST# is asserted during the last data/wait state to signify the last transfer of the access. After all data has been transferred, the bus enters the recovery state to allow the bus devices to recover. After recovery state, the bus enters the idle state and waits for another access.

2.2 LOCAL SIGNALS

The key Local Bus control signals shown in most timing diagram examples are as follows:

- ADS# or ALE indicates the start of an access
- READY#, WAIT#, and BTERM# indicate data transfer with wait state or terminating Burst cycle
- LWR#, direction of data transfer
- BLAST#, BTERM# indicate the end of an access

The key data signals are:

- LAD address, data bus
- LBE# local byte enables, indicating valid byte lane

2.3 LOCAL BUS SIGNALS

There are four groups of Local Bus signals: Clock, Address/Data/Parity, Control/Status, and Arbitration. Signal usage varies upon application.

2.3.1 Clock

LCLK, the Local Bus clock, operates at frequencies up to 66 MHz, and is asynchronous to the PCI Bus clock. Most Local Bus signals are driven and sampled on the rising edge of LCLK. The ALE address latch enable is asserted on the rising edge and de-asserted on the falling edge of LCLK. Setup and hold times, with respect to LCLK, must be observed.

2.4 BUS REGIONS

There are six bus regions and each region can have its own characteristics, such as Bus width (8, 16, or 32 bit), Burst enable, Bterm enable, Parity checking, Big/Little Endian selection, Read prefetch count, and Timeout enable. If the IOP 480 is a Local Bus slave, the bus width is fixed 32-bit, and the regions are Direct Master memory, Direct Master I/O, and Local access registers.

The bus regions are:

- Chip Selects (LCS0#, LCS1#, LCS2#, and LCS3#)
- DRAM
- Default

The three Master Bus regions are:

- Direct Master Memory
- Direct Master I/O
- Configuration Register

The Direct Master and Configuration regions are fixed at 32-bit wide regions.

2.4.1 Address/Data/Parity

2.4.1.1 LAD[31:0]

The LAD[31:0] bus is a 32-bit multiplexed address/data bus. During address state, LAD[31:2] contains the word address of the transfer. Memory address bits MA[17:0] provide an incrementing word address during burst access (refer to Section 12, “Memory Controller”). If the Bus width is 8 or 16 bits, additional de-multiplexed address signals are available on the unused byte enables: LBE1# acts as A1 and LBE0# acts as A0. (Refer to Table 2-14 on page 2-16.)

Note: *Dedicated address pins are available.*

During Data states, LAD[31:0], LAD[15:0], or LAD[7:0] contain transfer data for a 32-, 16-, or 8-bit bus, respectively. If the bus is 8 or 16 bits wide, the data supplied by the IOP is replicated across the entire 32-bit wide bus.

2.4.1.2 DP[3:0]

Byte parity is generated when data is driven out of the IOP 480. When data is passed into the IOP 480, parity is checked on each active byte lane. Even or odd parity can be selected. An interrupt can be generated if a parity error is detected. Each address space on the Local Bus has configuration register bits for enabling parity checking and selecting even or odd parity.

Table 2-1. DP[3:0]

DP Bit Number	LAD Parity Byte Lane Relationship
3	[31:24]
2	[23:16]
1	[15:8]
0	[7:0]

2.4.2 Control/Status

The control/status signals the control address latches and flow of data across the Local Bus.

2.4.2.1 ADS# and ALE

A Local Bus access starts when ADS# (address/data status) is asserted during an address state by the Local Bus Master. ALE is used to strobe the LAD Bus into an external address latch.

2.4.2.2 LBE[3:0]#

During an address state, the LBE[3:0]# byte enables denote which byte lanes are being used during access of a 32-bit bus. They remain asserted until the end of the data transfer. Refer to Table 2-14 on page 2-16 for a functionality description of LBE[3:0]# for different bus widths.

2.4.2.3 LWR#

During an address state, LWR# is driven to a valid state, signifying the data transfer direction. When the IOP 480 is the Local Bus master, LWR# is driven high when the IOP 480 is writing data to the Local Bus, and low when it is reading the bus. As a slave, the IOP 480 monitors LWR# to determine direction of the data transfer. LWR# is driven high by another Local Bus master that is writing data to the IOP 480 Local Bus, and low when data is being read from the IOP 480 Local Bus.

2.4.2.4 BLAST#

BLAST# is asserted by the current Local Bus master to indicate the last transfer of an access, both for Single and Burst accesses. When the IOP 480 is the Local Bus Master, the BLAST# Timing bit (LOCCTL[19]) determines the behavior of BLAST#. If that bit is set to 0, BLAST# is asserted by the master during the entire last transfer of an access. If that bit is set to 1, then BLAST# is not asserted by the master until the internal wait state counters have finished counting.

Note: *DMA can be programmed to end an access without BLAST# being asserted (C0MODE[15], C1MODE[15], and/or C2MODE[15]).*

2.4.2.5 READY#

READY# indicates that Write data is being accepted or Read data is being provided by the bus slave. If a Bus Slave needs to insert wait states, it can de-assert READY# until it is ready to accept or provide data. The READY# input pin has a corresponding Enable bit in the Configuration registers for each Local address space. If READY# is disabled, then the length of the Local Bus transfer is determined by internal wait state generators. (Refer to Table 2-2.)

Table 2-2. READY Data Transfers

Master Device	Slave Device	READY# Input Enable	READY#	Description
IOP 480	SRAM space	0	Driven	Data transfers determined by wait state generator. READY# is asserted when data transfer takes place.
IOP 480	SRAM space	1	Input	Data transfers determined by an external device which asserts READY# to indicate data transfer is taking place.
IOP 480	DRAM space	—	Driven	Data transfers determined by wait state generator. READY# is asserted when data transfer takes place.
IOP 480	Default space	—	Input	Data transfers are determined by an external device, which asserts READY# to indicate data transfer is taking place.
External	SRAM space	0	Driven	Data transfers determined by wait state generator. READY# is asserted when data transfer takes place.
External	SRAM space	1	Input	Data transfers determined by an external device, which asserts READY# to indicate data transfer is taking place.
External	DRAM space	—	Driven	Data transfers determined by wait state generator. READY# is asserted when data transfer takes place.
External	Default space	—	Input Driven if timeout	Data transfers are determined by an external device, which asserts READY# to indicate data transfer is taking place If a timeout occurs, and the Timeout Ready Out Enable bit is set, READY# is driven.
External	IOP 480 Configuration Registers	—	Driven	Data transfers determined by internal logic. READY# is asserted when data transfer takes place.
External	IOP 480 Direct Master Controller	—	Driven	Data transfers determined by internal logic. READY# is asserted when data transfer takes place.

2.4.2.6 BTERM#

The burst terminate signal, BTERM#, is used by a bus slave to stop a burst access. When the IOP 480 is the Local Bus Master and detects BTERM# asserted, it terminates the burst. If there is more data to transfer, a new burst access is initiated with the assertion of ADS#. If the internal wait state generators are active, then BTERM# is not sampled until the wait state counters decrement to zero. BTERM# also signals to the IOP 480 that data has been accepted or provided, so READY# need not be asserted when BTERM# is asserted.

When the IOP 480 is the Local Bus Slave, it asserts BTERM# and READY# when a PCI abort or Retry timeout is encountered during a Direct Master access. If the READY# input is enabled, then BTERM# is asserted when the Local address matches the page boundary. If the READY# input is disabled, BTERM# is asserted when READY# is driven by the IOP 480 if the Local address matches the page boundary.

The BTERM# input pin has a corresponding Enable bit in the configuration registers for each Local Address Space. If BTERM# is disabled, then the length of a Local burst when the IOP 480 is the Local Bus Master is a maximum of four Lwords.

Table 2-3. Burst and Bterm on Local Bus

Mode	Burst	Bterm	Result
Single Cycle	0	0	One ADS# per data (default).
Single Cycle	0	1	One ADS# per data.
Burst-4	1	0	One ADS# per four data (recommended for i960 and PPC401 family).
Burst Forever	1	1	One ADS# per BTERM#.

Note: In Table 2-3, 0 = disable and 1 = enable.

2.4.2.7 WAIT#

When an external agent is the Local Bus Master, WAIT# can be used to signal the IOP 480 that the Local Bus Master cannot accept or provide data, and needs wait states to be inserted. The IOP 480 signals the Local Bus master with READY# when it has provided/accepted the Direct Master data. Using both WAIT# and READY# allows the Master or Slave to insert wait states during Direct Master accesses.

When the IOP 480 is the Local Bus Master, WAIT# is an output that provides status of the internal wait state generators. It is asserted while internal wait states are being inserted—READY# as an input is not sampled until WAIT# is de-asserted.

2.4.2.8 LLOCK#

When the IOP 480 is the Local Bus Master, LLOCK# is asserted to indicate that an atomic operation for a Direct Slave PCI access may require multiple transactions to complete. LLOCK# is asserted during the address state of the first transaction of the atomic operation, and de-asserted one clock after the last transaction of the atomic operation is complete. If enabled, the Local Bus arbiter does not grant the bus to another Master until the atomic operation has completed.

2.4.3 Arbitration

Arbitration signals control which device is to be Local Bus Master. A round-robin arbiter selects between five internal and two external Local Bus Masters. (Refer to Section 8, “Local Bus Internal Arbiter.”)

2.4.3.1 BOFF#

BOFF# is an output indicating the IOP 480 requires the Local Bus for a Direct Slave access while a Direct Master access is pending. It is associated with the Deadlock situation (refer to Section 5.9, “Deadlock Conditions”). LARBR[16] is used to enable the Local Bus BOFF# signal (refer to Section 5.9.1, “Backoff”).

2.4.3.2 LHOLDREQ0/LHOLDACK and LHOLDREQ1

The IOP 480 Local Bus arbiter supports two external bus masters and five internal bus masters (a Direct Slave controller, three DMA controllers, and the IOP 480 CPU). When an external Local Bus Master wants control of the Local Bus, it asserts its corresponding Local Bus Hold Request signal through the IOP 480 (either the multiplexed LHOLDREQ0/ LHOLDACK pin or the non-multiplexed LHOLDREQ1 pin). After the Local Bus is granted, the external Local Bus Master should keep its corresponding Local Bus Hold Request signal (either the multiplexed LHOLDREQ0/LHOLDACK pin or the non-multiplexed LHOLDREQ1 pin) asserted until it is finished with the bus. (Refer to Table 2-4.)

The Local Arbiter Enable bit (LARBR[0]), is used to enable the IOP 480 Local Bus arbiter (default is enabled).

Table 2-4. Local Arbitration Signal Directions

External Arbitration Signal	Local Arbiter Enabled	Local Arbiter Disabled
LHOLDREQ0/LHOLDACK	Input	Input
LHOLDREQ1	Input	Not used
LHOLDACK0/LDREQ	Output	Output
LHOLDACK1/BREQ	Output	Not used

2.4.3.3 LHOLDACK0/LDREQ and LHOLDACK1/BREQ

The IOP 480 Local Bus arbiter asserts its corresponding Local Bus Hold Acknowledge signal (either the multiplexed LHOLDACK0/LDREQ or LHOLDACK1/BREQ pin) to grant the Local Bus to an external Local Bus Master. (Refer to Table 2-4.)

2.4.4 Local Chip Selects

During a Local Bus cycle, the accessed range can be one of the following:

- Configuration register
- Direct Master Memory
- Direct Master I/O
- One of four SRAM spaces (LCS[3:0]#)
- DRAM space
- Default

Local Chip Select (LCS[3:0]#) is used to select one of four SRAM spaces (devices). Each range is defined by its base and range, and access should be enabled for it to be selected.

For example, LCS3# is asserted if the Local address of an access is within the range defined by the Local Chip Select 3 Base Address (LCS3BASE) and Local Chip Select 3 Range (LCS3RANGE), and Local Chip Select 3 enabled (LCS3BASE[0]=1). Because LCS3# is the output on a multiplexed pin (LCS3#/MA17), the LCS3# function **must** be selected in the LOCCTL Configuration register (LOCCTL[7]=0) before using this Bus region.

The default LCS0# Base Address is 0xFFF0_0000, which allows the internal IOP 480 CPU to boot from a serial EEPROM at location 0xFFFF_FFFC.

The local characteristics of all accesses in the LCS range are defined by three registers, as described in the following table.

Table 2-5. Registers Defining Local Characteristics of all LCS Range Accesses

Register	Defined By
Local Chip Select Bus Region Descriptor (LCSxBRD)	Characteristics of: • Recovery state • Prefetch • Timeout • Parity • Memory write protect • BTERM# input • READY# input • Burst • IOP 480 byte ordering • Big Endian byte lane mode • Local byte ordering • Data Bus width
Local Chip Select Write Timing (LCSxWT)	Number of: • Write Recovery states • Write Hold states • Write Delay states • Write Data-to-Data Wait states • Write Address-to-Data Wait states
Local Chip Select Read Timing (LCSxRT)	Number of: • Read Recovery states • Read Delay states • Read Data-to-Data Wait states • Read Address-to-Data Wait states

2.5 LOCAL BUS PROTOCOL

2.5.1 Basic Bus Accesses

A basic Bus access consists of one data transfer. It can be one byte, word, or Lword of data being transferred between a master and a slave. After a Master gains control of the Local Bus, it pulses ALE to latch the address or asserts ADS#. The IOP 480 does both with the following valid address and cycle status information:

- LAD[31:2] Word address
- LBE[3:0]# Byte enables
- LWR# 1=Write status, 0=Read status

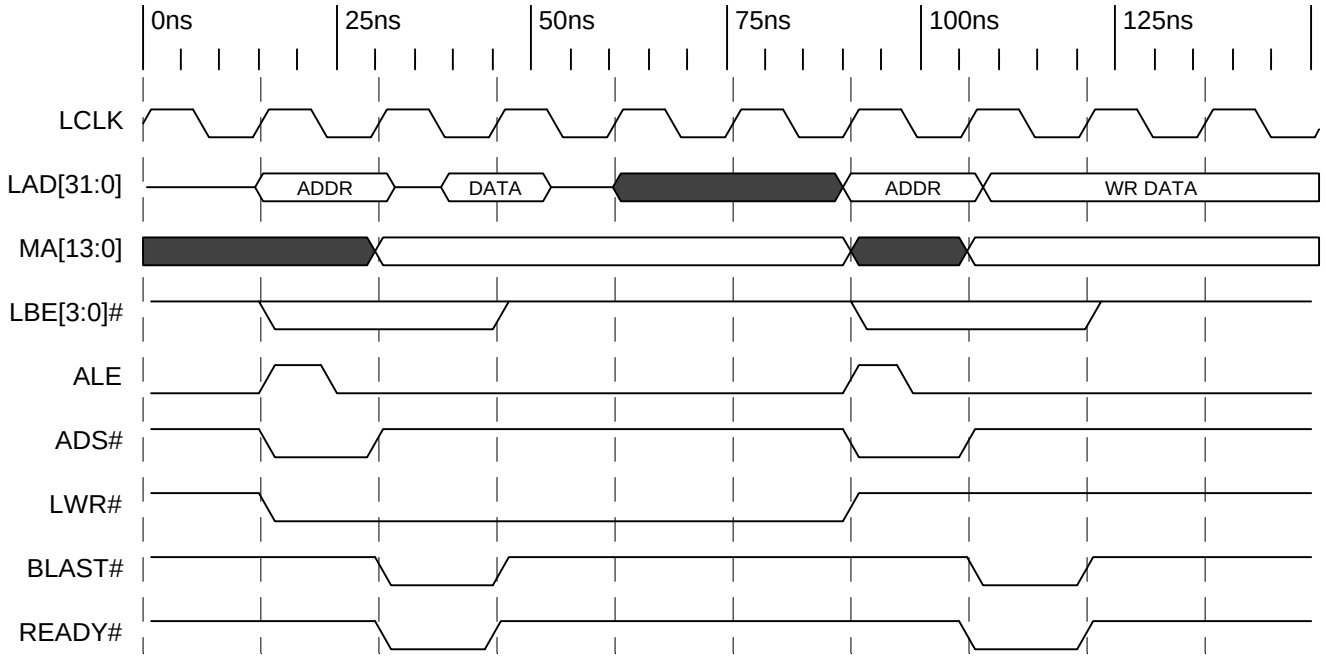
For additional details of other signals during a Read or Write access, refer to Section 12, “Memory Controller.”

During the next clock cycle, ADS# is de-asserted, and BLAST# asserted. The Bus Master then waits for a data transfer upon which BLAST# is de-asserted. (Refer to Figure 2-2.)

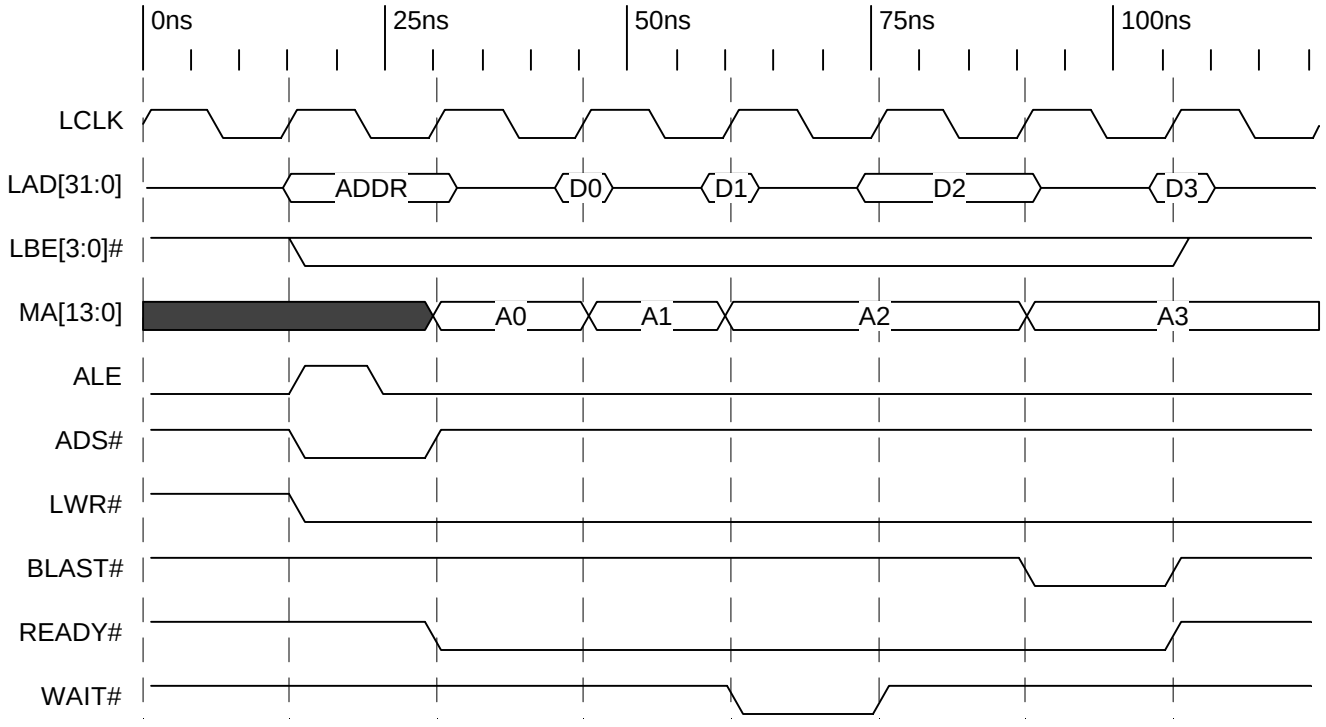
2.5.2 Wait States

READY# is used by a slave to insert wait states in a data transfer. During data/wait states, data transfer takes place if READY# was sampled asserted by the master on the rising edge of the clock. If READY# input is disabled for a Local Address space, then the number of wait states is determined by the internal wait state generator. For DRAM space, READY# input is ignored and the number of wait states is always determined by the internal wait state generator.

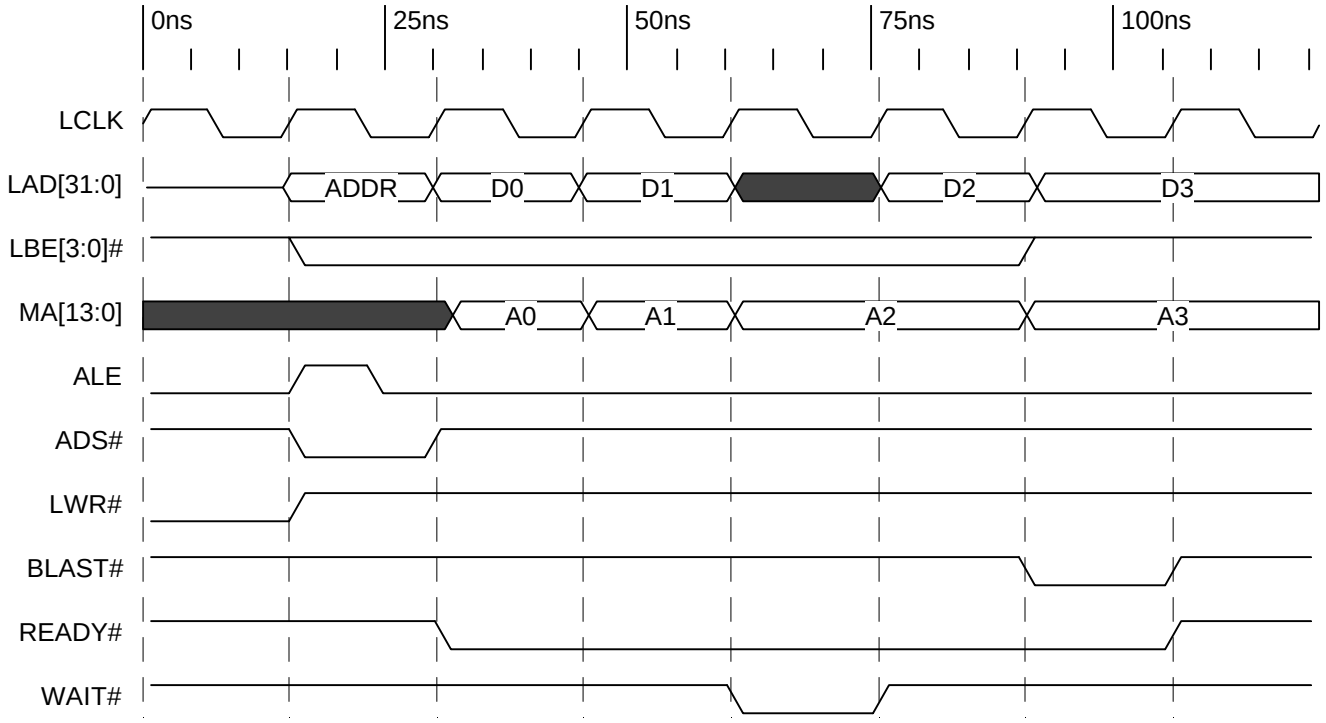
The WAIT# pin is sampled by the IOP 480 during Direct Master accesses, and used by the Local Bus Master to insert wait states. The WAIT# pin is driven by the IOP 480 when it is the Local Bus Master, and asserted when the internal wait state generator is inserting wait states.



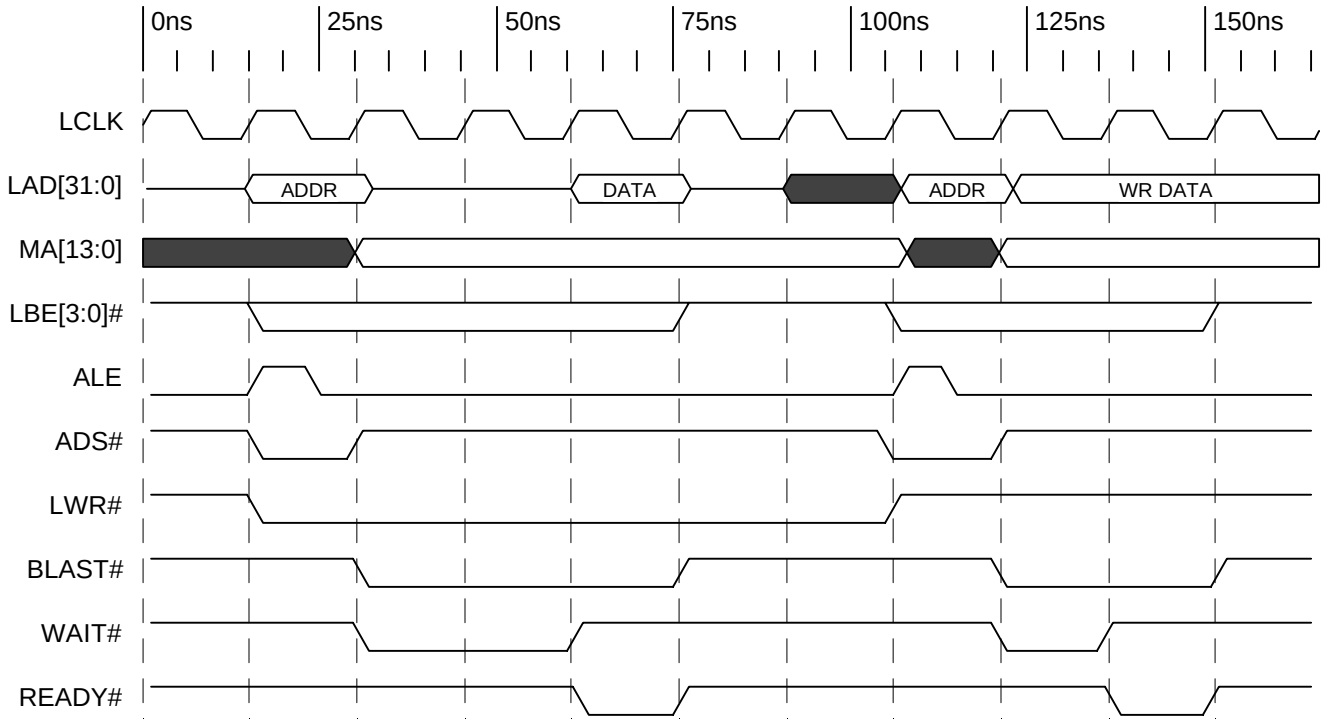
Timing Diagram 2-1. Single Read/Write, 32-Bit Bus, Master=IOP 480



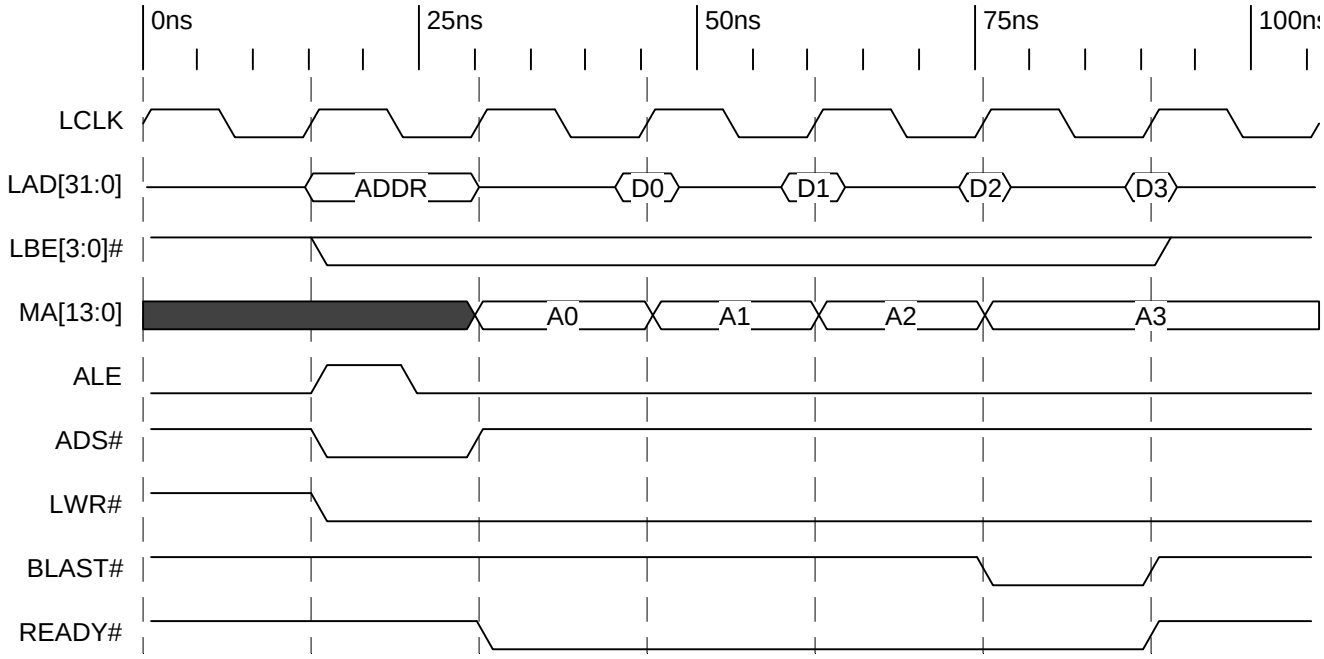
Timing Diagram 2-2. Local Bus Burst Read, Delayed with WAIT#, Master=External Local Bus Master



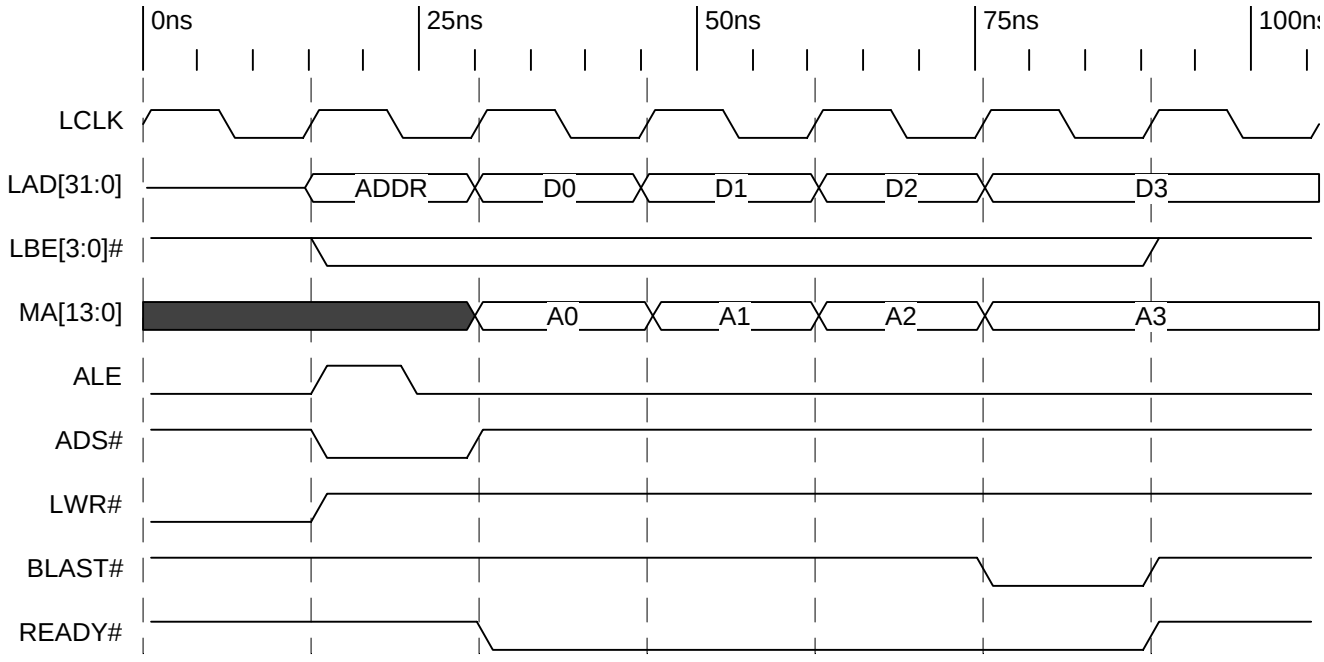
Timing Diagram 2-3. Local Bus Burst Write, Delayed with WAIT#, Master=External Local Bus Master



Timing Diagram 2-4. Single Read/Write with Internal Wait States, 32-Bit Bus, Master = IOP 480



Timing Diagram 2-5. Local Bus Burst Read, 32-Bit Bus, No Wait States, Master=IOP 480, Slave=Default Space



Timing Diagram 2-6. Local Bus Burst Write, 32-Bit Bus, No Wait States, Master=IOP 480, Slave=Default Space

2.5.3 Bus and Control Signals during Recovery and Idle States

Bus signals are driven to the following state during recovery:

- LAD[31:0] floats following a Read access
- LAD[31:0] freezes
- ALE, ADS#, and BLAST# are de-asserted
- LBE[3:0]# are driven to ones
- MA[17:0] and LWR# freeze

If the internal arbiter is enabled (LARBR[6]=1) and no external requester controls the bus, LAD and LBE[3:0]# are driven by the IOP 480 during idle states.

2.5.4 Burst Transactions

A Burst access consists of at least two data transfers. Maximum length of a burst depends upon the Local Bus Master, Local Slave, and Bus memory region descriptor registers.

As with a basic Bus access, a Master gains control of the Local Bus and asserts ADS# with the valid address and cycle status information. The Local Bus Master then continues with the data transfer. BLAST# is asserted during the last data transfer, indicating the end of the bus access. However, an exception exists. DMA can be programmed to use EOT# or DREQ# in place of BLAST# to end the Bus access. (Refer to Section 7, “DMA Operation.”) If the number of Address-to-Data wait states is set to zero, care must be taken to prevent bus contention between the address and first data word. In most cases, at least one Address-to-Data wait state is required.

During a write access, the Master continues driving the write data until the Slave asserts READY#. The next word of data is driven and burst address incremented. BLAST# is asserted during the last data transfer, indicating the end of access.

2.6 BUS REGION DESCRIPTORS

2.6.1 Direct Slave or DMA Burst

When the Direct Slave Interface, or one of the internal DMA controllers in the IOP 480 is the Local Bus Master, there are several bits in the Memory Controller Configuration registers that determine bursting characteristics. There are six sets of bus region descriptor registers, one of which is selected during a Local Bus access. The value on LAD[31:2] Bus during an address state determines which set is used. Within each set of registers there are two bits that control bursting, Burst Enable and Bterm Enable. If Burst is disabled, then the IOP 480 only runs single transfers on the bus. If Burst is enabled, the IOP 480 runs burst access if more than one piece of data needs to be transferred. Bterm Enable controls the length of bursts. If Bterm is disabled, then bursting can start on any address boundary and continue up to an address boundary specified in Table 2-6 for a maximum of four data transactions.

Table 2-6. Burst Boundaries with Bterm Enable=0

Bus Width	Boundary
32 bits	Four Lwords or up to a quad-Lword boundary (LA3,LA2 = 11)
16 bits	Four words or up to a quad-word boundary (LA2,LA1 = 11)
8 bits	Four bytes or up to a quad-byte boundary (LA1,LA0 = 11)

If Bterm Enable is high, then bursting continues until one of the following occurs:

- FIFOs become full or empty
- Target device asserts BTERM#
- Page Boundary Crossing detection asserts BTERM#, according to Table 2-7

If there is more data to transfer when the burst terminates, a new bus access is initiated.

Table 2-7. Burst Boundaries with Bterm Enable=1

Local Address Space	Number of Columns	Boundary
SRAM	—	32 KB (7FFFh)
EDO DRAM	8	1 KB (3FFh)
EDO DRAM	9	2 KB (7FFh)
EDO DRAM	10	4 KB (FFFh)
EDO DRAM	11	8 KB (1FFFh)
EDO DRAM	12	16 KB (3FFFh)
SDRAM (x16)	8	1 KB (3FFh)
SDRAM (x8)	9	2 KB (7FFh)
SDRAM (x4)	10	4 KB (FFFh)
Default	—	2 KB (7FFh)

2.6.2 Wait State Control

If READY# mode is disabled, the external READY# input signal has no effect on wait states for a Local access. Wait states between Data cycles are asserted internally by a wait state counter. The wait state counter is initialized with its Configuration register value at the start of each data access.

If READY# mode is enabled, READY# then controls the number of additional wait states.

BTERM# overrides READY# when BTERM# is enabled and asserted.

2.7 ENDIAN SWAPPING

The IOP 480 supports both Big and Little Endian byte ordering on the Local Bus. Big Endian implies that the most significant byte is located at the lowest address, while Little Endian implies that the least significant byte is located at the lowest address. There are two approaches to mapping between Big and Little Endian, address invariance and data invariance. The IOP 480 uses the address invariance approach in which the address of each byte in memory is preserved. This technique employs the use of byte lane swapping.

Table 2-8. Endian Swapping

Big Endian Word				Little Endian Word			
b0……b31				b31……b0			
B0	B1	B2	B3	B3	B2	B1	B0
P	L	X	T	T	X	L	P
MSB			LSB	MSB			LSB
LAD31……LAD0				LAD31……LAD0			

2.7.1 Direct Master or Configuration Register Access

As a Local Bus slave, the IOP 480 supports Big and Little Endian byte ordering. The Big/Little Endian configuration register (BIGEND) contains bits to select Big or Little Endian mode for Direct Master or configuration register accesses. One set of bits (BIGEND[1:0]) are used when an external master controls the Local Bus, and another set (BIGEND[3:2]) when the IOP 480 CPU controls the Local Bus.

2.7.2 Direct Slave or DMA Access Endian Swapping

As a Local Bus Master, and during Direct Slave or DMA accesses, the IOP 480 byte ordering and Big Endian Byte Lane Mode bits in each of the memory controller bus region descriptor registers determine byte ordering.

2.7.3 Endian Swapping Example

Suppose a value of 0xAABBCCDD is stored in one of the internal registers or the IOP 480 FIFOs. The value appears on various byte lanes depending upon bus width, address, and Endian selection. When Big Endian mode is selected, upper or lower bytes lanes can be used for 8- or 16-bit bus widths, depending upon the configuration bit. (Refer to Table 2-9 and Table 2-10.)

2.7.4 Internal IOP 480 CPU

The natural byte ordering of the IOP 480 CPU is Big Endian. Although the IOP 480 CPU configuration bits allow it to switch to Little Endian byte ordering, it is recommended that Endian swapping configuration issues be handled through the IOP 480 configuration registers (LCS0BRD[4], LCS1BRD[4], LCS2BRD[4], LCS3BRD[4], DFLTBRD[4], and DRAMBRD[4]). These IOP 480 Configuration registers allow selection

of Big or Little Endian ordering for each of six Local Address Spaces (four SRAM, one DRAM, and one default).

The IOP 480 utilizes address invariant Endian swapping. This preserves the relative byte addresses of all data flowing through, thus preserving character strings. Accesses to an 8-bit Local Bus device do not swap byte order, regardless of whether the Local Bus is set for Big or Little Endian ordering. (Refer to Figure 2-2.)

Table 2-9. Endian Swapping, Lower Byte Lanes

Word Data Type			Byte Lanes							
Bus Width	Addr Bits A1, A0	Transfer	Little Endian				Big Endian, Lower Byte Lanes			
			AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]	AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]
32 bits	00	1st	AA	BB	CC	DD	DD	CC	BB	AA
16 bits	00	1st	—	—	CC	DD	—	—	DD	CC
	10	2nd	—	—	AA	BB	—	—	BB	AA
8 bits	00	1st	—	—	—	DD	—	—	—	DD
	01	2nd	—	—	—	CC	—	—	—	CC
	10	3rd	—	—	—	BB	—	—	—	BB
	11	4th	—	—	—	AA	—	—	—	AA

Table 2-10. Endian Swapping, Upper Byte Lanes

Word Data Type			Byte Lanes							
Bus Width	Addr Bits A1, A0	Transfer	Little Endian				Big Endian, Upper Byte Lanes			
			AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]	AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]
32 bits	00	1st	AA	BB	CC	DD	DD	CC	BB	AA
16 bits	00	1st	—	—	CC	DD	DD	CC	—	—
	10	2nd	—	—	AA	BB	BB	AA	—	—
8 bits	00	1st	—	—	—	DD	DD	—	—	—
	01	2nd	—	—	—	CC	CC	—	—	—
	10	3rd	—	—	—	BB	BB	—	—	—
	11	4th	—	—	—	AA	AA	—	—	—

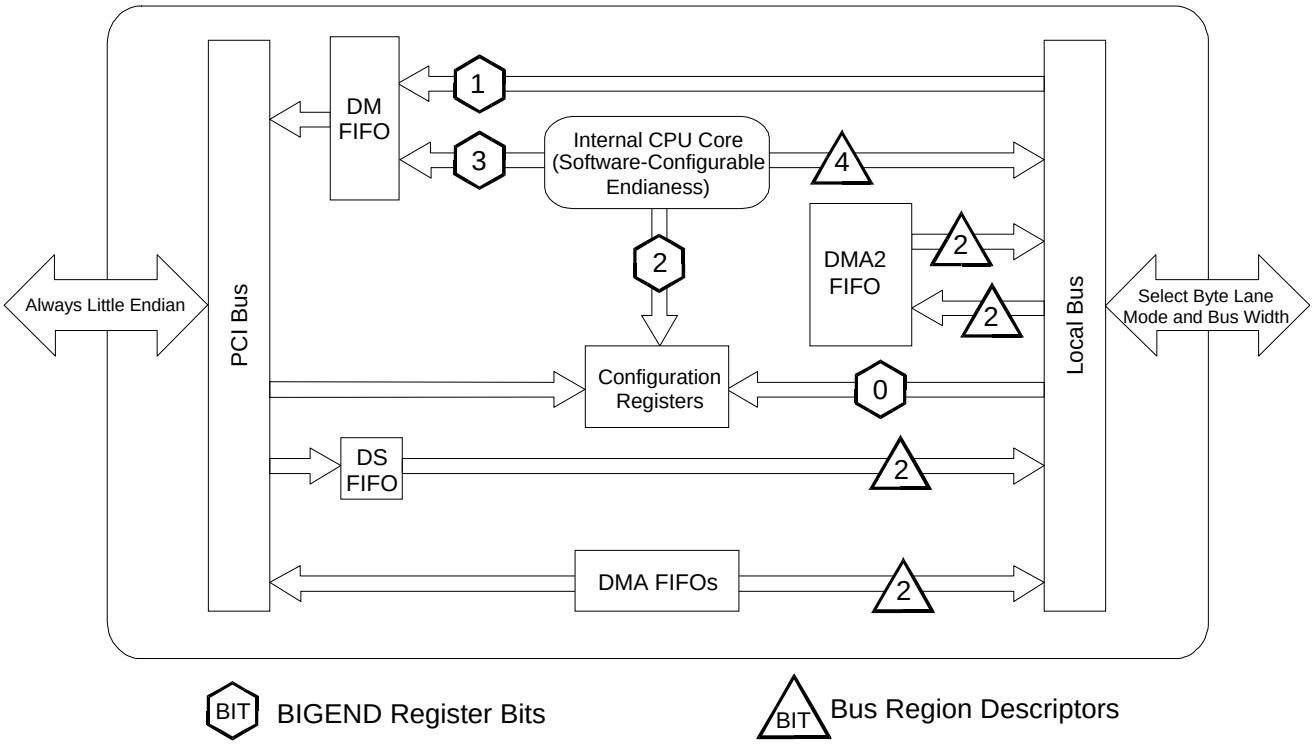


Figure 2-2. Endian Swapping

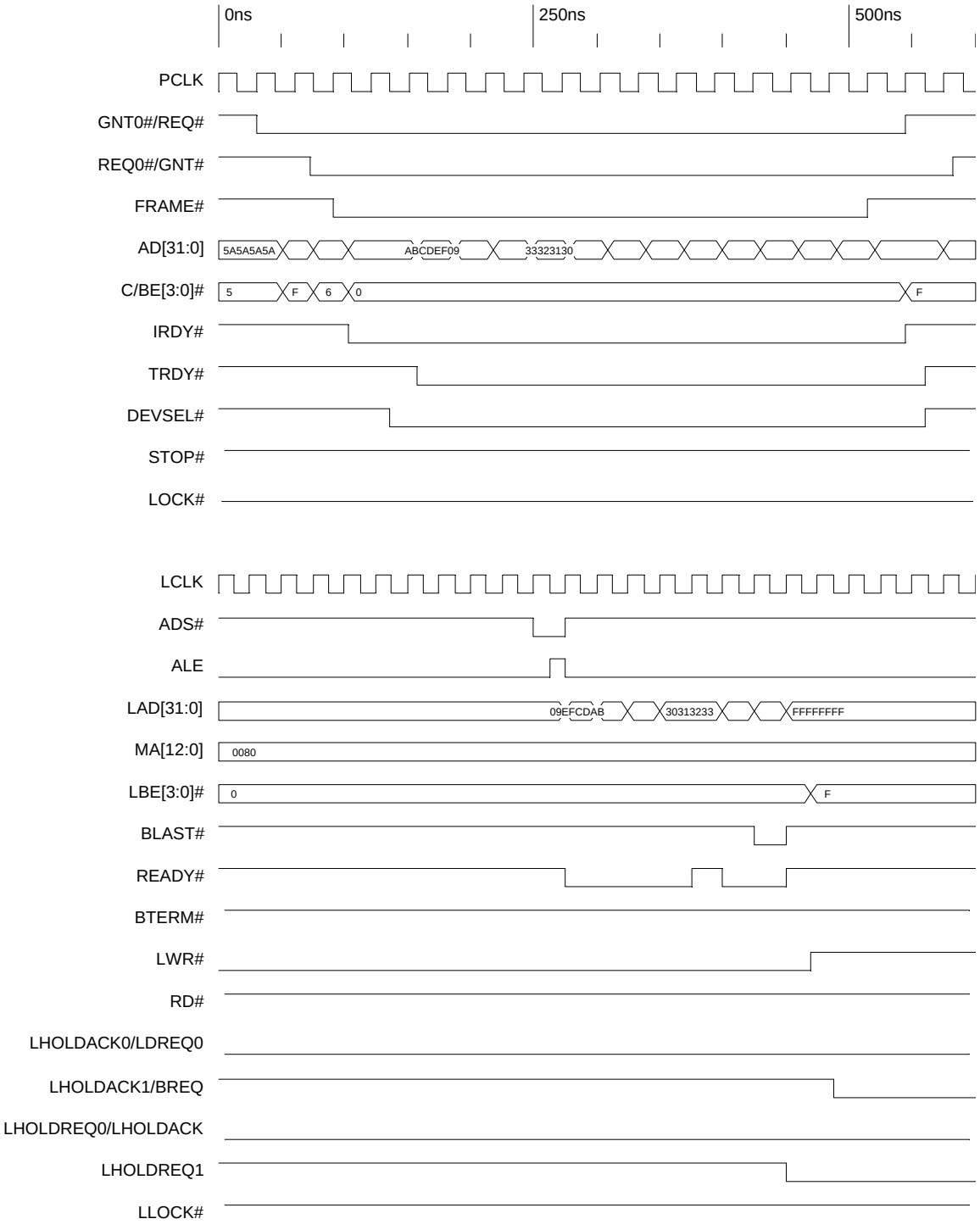
Note: *Arrows in the figure denote the initiator of a cycle (read or write).*

2.7.4.1 Big Endian (Internal CPU Setting)

Table 2-11. IOP 480 CPU Big Endian Byte Ordering

Word Data Type			Byte Lanes							
Bus Width	Addr Bits A1, A0	Transfer	Little Endian				Internal CPU Bus			
			AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]	AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]
32 bits	00	1st	AA	BB	CC	DD	DD	CC	BB	AA
16 bits	00	1st	—	—	CC	DD	DD	CC	—	—
	10	2nd	—	—	AA	BB	—	—	BB	AA
8 bits	00	1st	—	—	—	DD	DD	—	—	—
	01	2nd	—	—	—	CC	—	CC	—	—
	10	3rd	—	—	—	BB	—	—	BB	—
	11	4th	—	—	—	AA	—	—	—	AA

2.7.4.1.1 Big Endian Cycle Timing Diagram



Timing Diagram 2-7. Big Endian Cycle

2.7.4.2 Little Endian (Internal CPU Setting)

Table 2-12. IOP 480 CPU Little Endian Byte Ordering

Word Data Type			Byte Lanes							
Bus Width	Addr Bits A1,A0	Transfer	Little Endian				Internal CPU Bus			
			AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]	AD[31:24]	AD[23:16]	AD[15:8]	AD[7:0]
32 bits	00	1st	AA	BB	CC	DD	AA	BB	CC	DD
16 bits	00	1st	—	—	CC	DD	—	—	CC	DD
	10	2nd	—	—	AA	BB	AA	BB	—	—
8 bits	00	1st	—	—	—	DD	—	—	—	DD
	01	2nd	—	—	—	CC	—	—	CC	—
	10	3rd	—	—	—	BB	—	BB	—	—
	11	4th	—	—	—	AA	AA	—	—	—

2.8 BUS WIDTH

Memory controller configuration bus region descriptor registers determine the width of the Local Bus. As a slave, the IOP 480 expects all four byte lanes to be used (32-bit bus). The following byte lanes and byte enables are used for various bus widths. (Refer to Table 2-14).

During Write accesses, data is replicated on unused byte lanes. (Refer to Table 2-15).

2.9 DATA ALIGNMENT

2.9.1 IOP 480 as a Local Bus Slave

As a Local Bus Slave, the IOP 480 allows unaligned accesses (any combination of byte enables asserted) to the Configuration registers or the Direct Master FIFOs, and either Single or Burst transfers are supported. The IOP 480 must always be accessed as a 32-bit slave on the Local Bus.

2.9.2 IOP 480 as a Local Bus Master

As a Local Bus Master, the IOP 480 supports unaligned transfers.

For Direct Slave writes, the IOP 480 breaks any partial write into a single access. It only bursts on the Local Bus if all byte enables are asserted. If no PCI byte enables are active, there is no transfer to the Local Bus, and the IOP 480 issues a Target abort.

For Direct Slave reads, regardless of the PCI byte enable, the IOP 480 reads all bytes on the Local Bus during a burst prefetch transaction. For single reads, the IOP 480 passes the byte enables.

Table 2-13. Direct Slave Local Byte Enable Methods

Direct Slave Methods	Byte Enables
DSR Burst	Don't pass
DSR Single / I/O	Pass
DSW Burst	Breakup burst for alignment
DSW Single / I/O	Pass
DSW to 8-bit bus	Skip if no byte enables allowed

For a 32-bit bus, if a DMA transfer does not start at an address with LAD[1:0] = 00, then a partial word transfer is first performed, followed by word bursts. For a 16-bit bus, if the DMA transfer does not start at an address with LAD[0] = 0, then a byte transfer is first performed, followed by word bursts. For an 8-bit bus, a DMA transfer may start bursting at any address.

2.10 BUS ACCESSES

Any bus access can have different characteristics based upon the bus region descriptor. These characteristics include bus width and burst behavior.

Table 2-14. Byte Enable and Coding

Bus Width	LBE3#	LBE2#	LBE1#	LBE0#	Comments
32 bits	LAD[31:24]	LAD[23:16]	LAD[15:8]	LAD[7:0]	(all byte lanes used)
16 bits	LAD[15:8]	not used	A1	LAD[7:0]	(LBE3# = BHE#, LBE0# = BLE#)
8 bits	not used	not used	A1	A0	(all data transfers use LAD[7:0])

Note: *Unused signals are driven to 1 as output, and ignored as input.*

Table 2-15. Byte Lane Contents

Bus Width	LAD[31:24]	LAD[23:16]	LAD[15:8]	LAD[7:0]	Comments
8 bits	byte n	byte n	byte n	byte n	8-bit access
16 bits	byte n	byte n	byte n	byte n	8-bit access
16 bits	byte n+1	byte n	byte n+1	byte n	16-bit access
32 bits	byte n	byte n	byte n	byte n	8-bit access
32 bits	byte n+1	byte n	byte n+1	byte n	16-bit access

2.10.1 Data Transfer

Data transfer occurs on the Local Bus when the master and slave devices are ready to receive data. The master device uses the WAIT# pin to insert wait states, and the slave device uses the READY# pin. (Refer to Table 2-16.)

Table 2-16. Data Transfer Control

READY# Enabled	WAIT#	READY#	Transfer
0	0	x	No
0	1	x	Yes
1	0	x	No
1	1	0	Yes
1	1	1	No

3 PCI BUS INTERFACE

3.1 OVERVIEW

The IOP 480 is compliant with *PCI Specification r2.2*. Refer to *PCI Specification r2.2* for specific PCI Bus functions.

3.2 DIRECT SLAVE COMMAND CODES

As a Target, the IOP 480 allows access to the IOP 480 internal registers and the Local Bus, using the commands listed in Table 3-1.

All Read or Write accesses to the IOP 480 can be Byte, Word, or Longword (Lword) accesses. All memory commands are aliased to basic memory commands. All I/O accesses to the IOP 480 are decoded to an Lword boundary. Byte enables are used to determine which bytes are read or written. An I/O access with illegal byte enable combinations is terminated with a Target Abort.

Table 3-1. Direct Slave Command Codes

Command Type	Code (C/BE[3:0]#)
I/O Read	0010 (2h)
I/O Write	0011 (3h)
Memory Read	0110 (6h)
Memory Write	0111 (7h)
Configuration Read	1010 (Ah)
Configuration Write	1011 (Bh)
Memory Read Multiple	1100 (Ch)
Memory Read Line	1110 (Eh)
Memory Write and Invalidate	1111 (Fh)

3.3 PCI MASTER COMMAND CODES

The IOP 480 can access the PCI Bus to perform DMA or Direct Master Local-to-PCI Bus transfers. During a Direct Master or DMA transfer, the command code assigned to the IOP 480 internal register location (PCICTL[15:0]) is used as the PCI command code.

Notes: *Programmable internal registers determine PCI command codes when the IOP 480 is the Master.*

DMA cannot perform I/O or configuration accesses.

3.3.1 DMA Master Command Codes

The IOP 480 DMA controllers can assert the following Memory cycles.

Table 3-2. DMA Master Command Codes

Command Type	Code (C/BE[3:0]#)
Memory Read	0110 (6h)
Memory Write	0111 (7h)
Memory Read Multiple	1100 (Ch)
Dual Address Cycle	1101 (Dh)
Memory Read Line	1110 (Eh)
Memory Write and Invalidate	1111 (Fh)

3.3.2 Direct Local-to-PCI Command Codes

For direct Local-to-PCI Bus accesses, the IOP 480 asserts the cycles listed in the following tables.

Table 3-3. Local-to-PCI Memory Access

Command Type	Code (C/BE[3:0]#)
Memory Read	0110 (6h)
Memory Write	0111 (7h)
Memory Read Multiple	1100 (Ch)
Dual Address Cycle	1101 (Dh)
Memory Read Line	1110 (Eh)
Memory Write and Invalidate	1111 (Fh)

Table 3-4. Local-to-PCI I/O Access

Command Type	Code (C/BE[3:0]#)
I/O Read	0010 (2h)
I/O Write	0011 (3h)

Table 3-5. Local-to-PCI Configuration Access

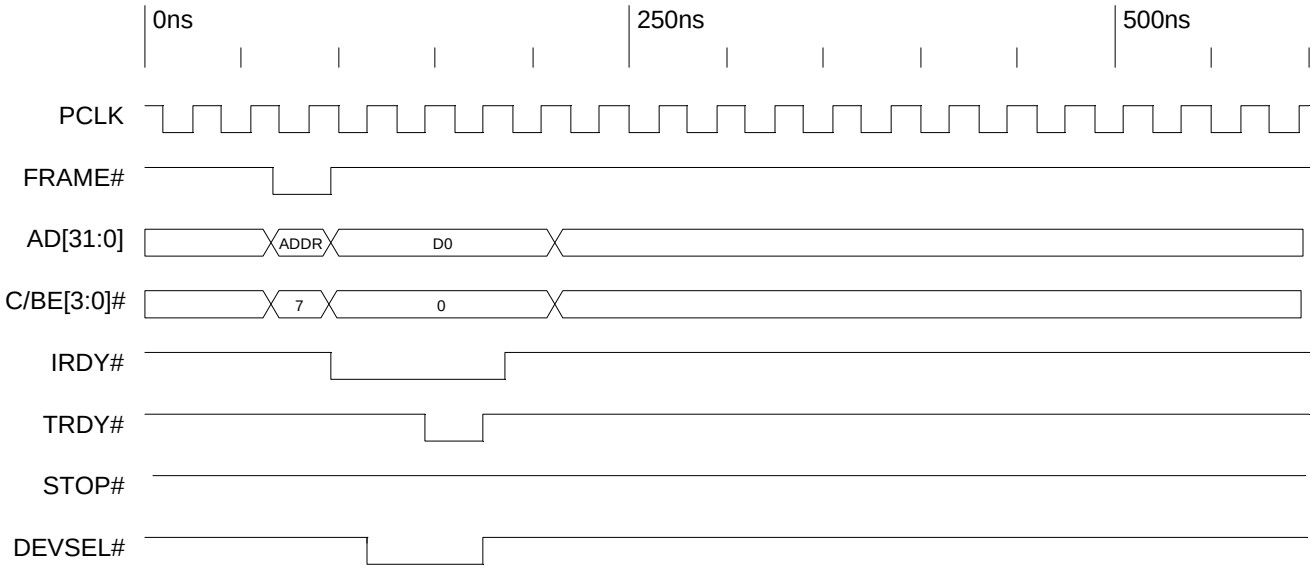
Command Type	Code (C/BE[3:0]#)
Configuration Memory Read	1010 (Ah)
Configuration Memory Write	1011 (Bh)

3.4 PCI SIGNALS

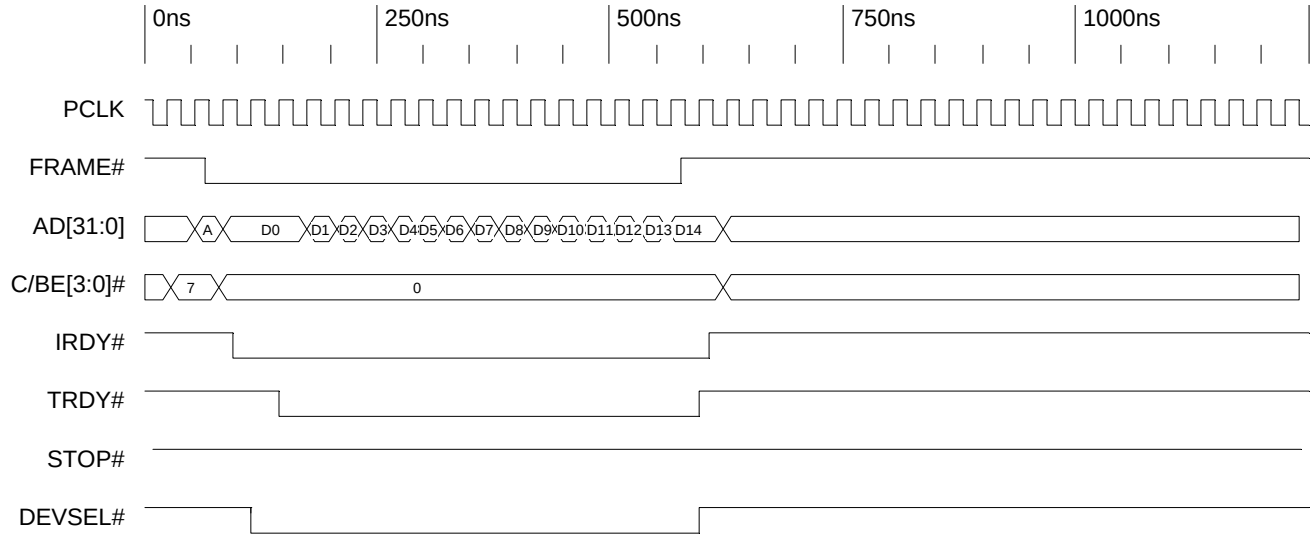
The key PCI Bus control signals are:

- FRAME#—1->0 transition indicates the start of an access, 0->1 indicates the last Data phase
- IRDY#, TRDY#—Indicate Data transfer
- DEVSEL#—Indicates target acceptance
- STOP#—Indicates premature termination
- AD—Address during Address phase
- C/BE[3:0]#—Command during address phase, byte enables during Data phase

3.4.1 PCI Signal Timing Diagrams



Timing Diagram 3-1. Direct Slave Write to 8-Bit Local Bus—PCI Signal Protocol



Timing Diagram 3-2. Direct Slave Write of 15 Lwords to 32-Bit Local Bus—PCI Signal Protocol

3.5 PCI BUS PROTOCOL

The PCI Bus Master throttles IRDY# and the PCI Bus slave throttles TRDY# to insert PCI Bus wait state(s).

A simple PCI protocol follows. The PCI Bus samples at the positive (rising) edge of the PCI Clock, PCLK. The PCI Bus is idle when FRAME# and IRDY# are both samples. (For more detail, consult additional PCI reference material.)

An address phase occurs when FRAME# is first sampled as 0[1→ 0 transition]. Thereafter, all clocks are data phases until the Bus is idle.

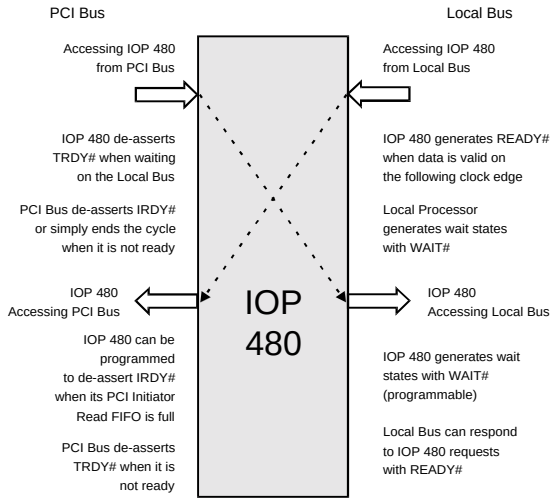


Figure 3-1. Wait States

Note: *The figure represents a sequence of bus cycles.*

3.6 BUS ARBITRATION

PCI Bus arbitration is point-to-point, rather than bused, handshaking. A potential initiator requests the bus by asserting its REQ# pin, and acquires it when its GNT# pin is asserted on an idle bus.

The IOP 480 can use either an internal or external PCI arbiter. The default is to use an external PCI arbiter. However, for maximum IOP 480 Direct Master performance, the internal PCI arbiter should be used. The selection is made in PCICTL[16].

Refer to Section 9, “PCI Bus Internal Arbiter” for more information on using the IOP 480’s internal PCI arbiters.

4 DIRECT SLAVE OPERATION

4.1 OVERVIEW

Direct Slave operations originate on the PCI Bus, transfer through the IOP 480, and finally access the Local Bus. The IOP 480 is a PCI Bus Target and a Local Bus Master.

4.1.1 Direct Slave

The PCI Bus Master reads from and writes to the Local Bus through one of four Memory- or I/O-mapped PCI spaces.

4.1.2 Direct Slave Operation (PCI Master-to-Local Bus Access)

The IOP 480 supports both Burst Memory-Mapped Transfer accesses and I/O-Mapped, Single-Transfer accesses to the Local Bus from the PCI Bus through a 64-byte Direct Slave Read FIFO and a 128-byte Direct Slave Write FIFO. The PCI Base Address registers are provided to set up the location of the adapter in PCI memory and I/O space. In addition, Local Mapping registers allow address translation from PCI Address Space to Local Address Space. There are four available spaces:

- Space 0 (Memory-mapped only)
- Space 1
- Space 2
- Expansion ROM

Expansion ROM space is intended to support a bootable ROM device for the PCI Host.

Local Bus Burst transfers depend upon Local Bus Region descriptors.

For Single-Cycle Direct Slave reads, the IOP 480 reads a single Local Bus Lword or partial Lword.

The IOP 480 disconnects after one transfer for all Direct Slave I/O accesses.

For the highest data transfer rate, the IOP 480 supports posted writes and can be programmed to prefetch data during a PCI Burst read. When the Read Prefetch register is enabled (set by bit 16 in the region descriptor), the Prefetch count can be set to be programmed as 0, 4, 8, or 16 (set by bits [18:17] in the region descriptor register). The IOP 480 prefetches, if enabled, and drops the Local Bus after reaching the prefetch counter value. In Continuous Prefetch mode, the IOP 480 prefetches as long as FIFO space is available and stops prefetching when the PCI Master completes or terminates the transfer. If read prefetching is disabled, the IOP 480 disconnects after one Read transfer.

In addition to Prefetch mode, the IOP 480 supports Read Ahead mode (refer to Section 4.6, “PCI Read Ahead Mode (PCICTL[22]),” on page 4-5).

Each Local Address Space can be programmed to operate with an 8-, 16-, or 32-bit Local Bus width. The IOP 480 has an internal wait state generator and external wait state input, READY#. READY# can be disabled or enabled in the Bus Region descriptor registers.

With or without wait state(s), the Local Bus, independent of the PCI Bus can

- Burst as long as data is available (Continuous Burst mode)
- Burst four Lwords at a time (recommended)
- Perform multiple Single-Cycle accesses

4.2 REGISTERS

4.2.1 PCI Bus Access to Internal Registers

The IOP 480 PCI Configuration registers can be accessed from the PCI Bus with a Configuration Type 0 cycle.

All other IOP 480 internal registers can be accessed at offsets from Local Address Space 0. Space 0 is memory-mapped at the address in the PCI Base Address 0 (PCIBAR0[31:10]) for the IOP 480 Memory-Mapped Configuration register. The internal registers take up 1K of Space 0.

All PCI read or write accesses to IOP 480 registers can be Byte, Word, or Lword accesses. All PCI Memory accesses to IOP 480 registers can be Burst or Non-Burst accesses.

Accessing reserved registers returns 0 for reads, and writes have no effect.

4.2.2 Direct Slave PCI-to-Local Address Mapping

Note: *Not applicable in I_2O mode.*

Four Local Address spaces—Space 0, Space 1, Space 2, and Expansion ROM—are accessible from the PCI Bus. Each is defined by a set of three registers:

- Local Address Range (LAS0RR, LAS1RR, LAS2RR, and/or EROMRR)
- Local Base Address (LAS0BA, LAS1BA, LAS2BA, and/or EROMBA)
- PCI Base Address (PCIBAR0, PCIBAR1, PCIBAR2, and/or PCIERBAR)

The Memory Controller registers define the Local Bus characteristics for the Direct Slave regions (refer to Figure 4-1 on page 4-3).

Each PCI-to-Local Address space is defined as part of reset initialization. These Local Bus characteristics can be modified at any time before actual data transactions.

4.2.2.1 Direct Slave Local Bus Initialization

- **Range**—Specifies which PCI address bits to use for decoding a PCI access to Local Bus space. Each bit corresponds to a PCI address bit. Bit 31 corresponds to Address bit 31. Write 1 to all bits that must be included in the decode and 0 to all others. (The range should be programmed before the other registers.)
- **Remap PCI-to-Local Addresses into a Local Address Space**—Bits in this register remap (replace) the PCI address bits used in decode as the Local Address bits.
- **Memory Controller Region Descriptors**—Specify the Local Bus characteristics.

4.2.2.2 Direct Slave PCI Initialization

After PCI reset, software determines the amount of address space required by writing all ones (1) to a PCI Base Address register and then reading back the value. The IOP 480 returns zeroes (0) in the Don't Care Address bits, effectively specifying the address space required. The PCI software then maps the Local Address space into the PCI Address space by programming the PCI Base Address register. (Refer to Figure 4-1.)

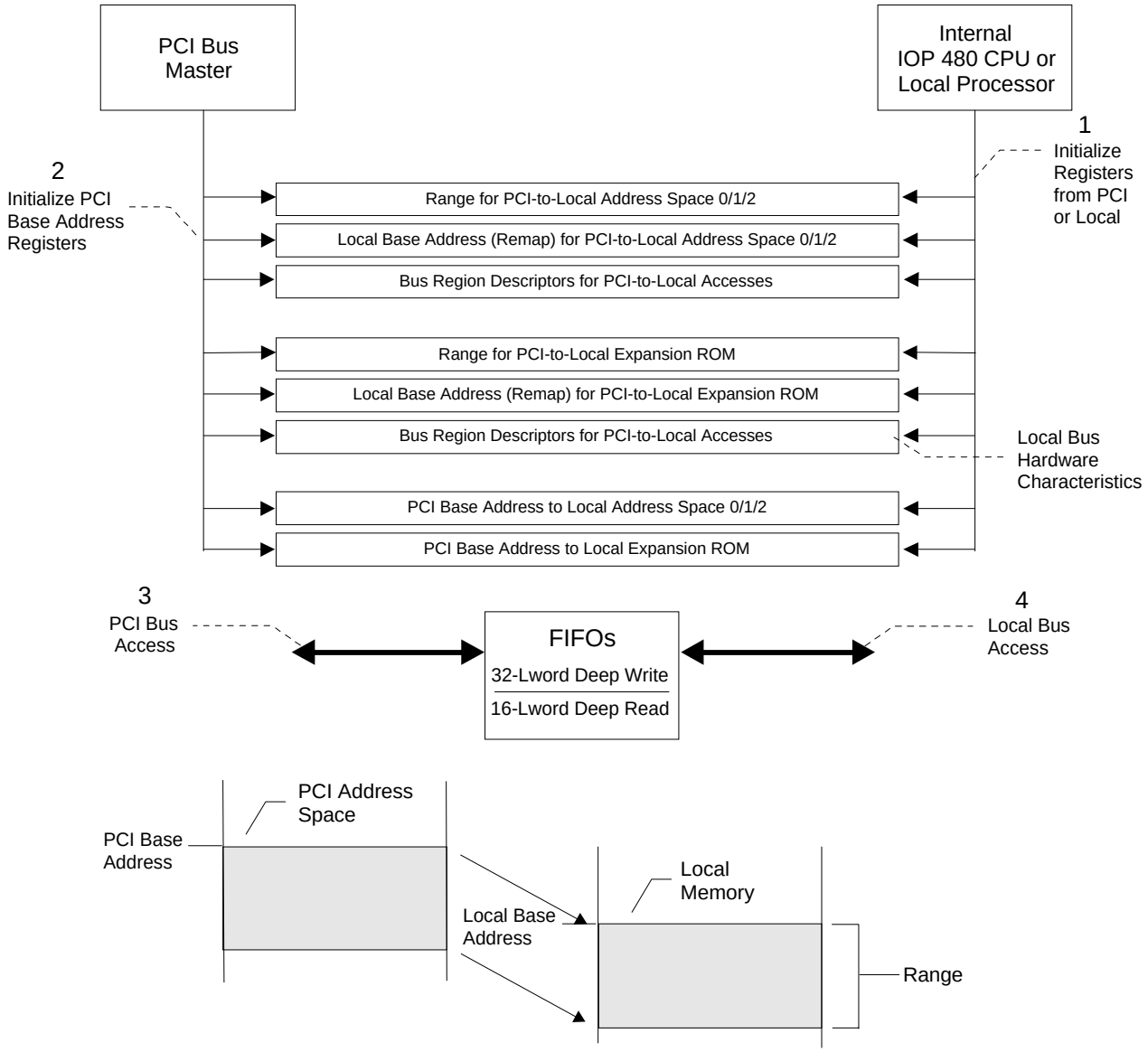


Figure 4-1. Direct Slave Access of Local Bus

4.3 INTERNAL FIFOS

For Direct Slave Memory accesses to the Local Bus, the IOP 480 utilizes a 32-Lword (128-byte) Write FIFO and a 16-Lword (64-byte) Read FIFO. The FIFOs enable the PCI Bus to operate independently of the Local Bus, and allow high-performance bursting on the PCI and Local Buses. For a Direct Slave write, the PCI Master writes data to the Local Bus Slave. For a Direct Slave read, the PCI Master reads data from the Local Bus Slave. Illustrated in Figure 4-4 and Figure 4-5 are the FIFOs that function during a Direct Slave write and read. Refer to Table 4-1, which also lists the response of the IOP 480 to full and empty FIFOs.

Table 4-1. Response to FIFO Full or Empty

Mode	Data Direction	FIFO	PCI Bus	Local Bus
Direct Master Write	Local-to-PCI	Full	Normal	De-assert READY#
		Empty	De-assert REQ# (off PCI Bus)	Normal
Direct Master Read	PCI-to-Local	Full	De-assert REQ# or throttle IRDY#[1]	Normal
		Empty	Normal	De-assert READY#
Direct Slave Write	PCI-to-Local	Full	Disconnect or throttle TRDY#[2]	Normal
		Empty	Normal	De-assert LHOLD, assert BLAST#[3]
Direct Slave Read	Local-to-PCI	Full	Normal	De-assert LHOLD, assert BLAST#[3]
		Empty	Throttle TRDY# [4]	Normal
DMA	Local-to-PCI	Full	Normal	De-assert LHOLD, assert BLAST#
		Empty	De-assert REQ#	Normal
	PCI-to-Local	Full	De-assert REQ#	Normal
		Empty	Normal	De-assert LHOLD, assert BLAST#

1. Throttle IRDY# depends on the Direct Master PCI Read Mode bit (DMPBAM[6]).
2. Throttle TRDY# depends on the Direct Slave Write bit (PCICTL[27]).
3. De-assertion of LHOLD depends upon the Local Bus Direct Slave Release Bus Mode bit (LARBR[8]).
4. Retry Throttle TRDY# depends upon Direct Slave Read bit (PCICTL[25]). De-assert TRDY# if bit is equal to zero. If bit is equal to 1, Retry.

4.4 EXCLUSIVE ACCESSES

The IOP 480 supports direct PCI-to-Local-Bus exclusive accesses (locked atomic operations). A PCI-locked operation to the Local Bus results in the entire address Space 0, Space 1, Space 2, and Expansion ROM space being locked until they are released by the PCI Bus Master. Locked operations are enabled or disabled with the Direct Slave LOCK# Enable bit (LARBR[9]) for PCI-to-Local accesses.

4.5 PCI R2.2 DELAYED READ MODE

The IOP 480 can be programmed through the PCI Specification r2.2 Mode bit (PCICTL[25]=1) to perform delayed reads, as specified in *PCI Specification r2.2*.

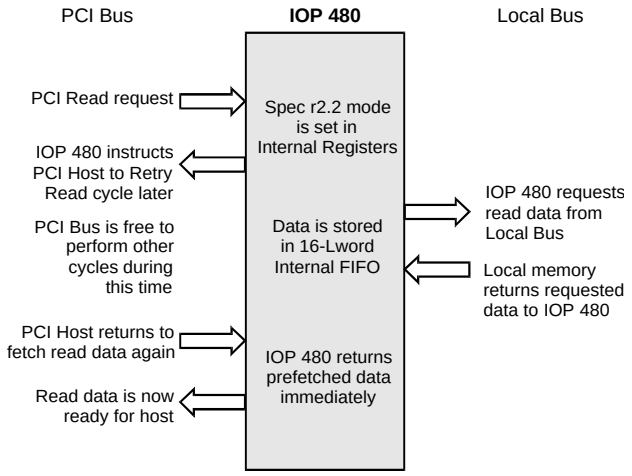


Figure 4-2. Direct Slave PCI Specification r2.2 Delayed Reads

Note: *The figure represents a sequence of bus cycles.*

In addition to delayed reads, the IOP 480 supports the following *PCI Specification r2.2* functions:

- No writes allowed while read is pending—PCI Retry for reads (PCICTL[24])
- Write and flush pending read (PCICTL[23])

4.6 PCI READ AHEAD MODE (PCICTL[22])

The IOP 480 also supports Read Ahead mode, where prefetched data can be read from the IOP 480 internal FIFO instead of the Local Bus. The address must be subsequent to the previous address and 32-bit aligned (next address=current address + 4). Read Ahead mode functions can be used with or without PCI Delayed Read mode.

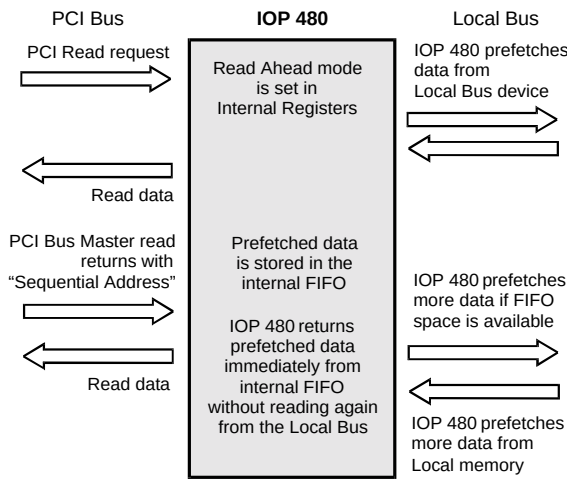


Figure 4-3. Direct Slave IOP 480 Read Ahead Mode

Note: *The figure represents a sequence of bus cycles.*

4.7 DIRECT SLAVE TRANSFER

Transactions are initiated by a PCI Bus Master addressing the Memory space decoded for the Local Bus. Upon a PCI Read/Write, the IOP 480 becomes a Local Bus Master and arbitrates for the Local Bus.

The IOP 480 then reads data into the Direct Slave Read FIFO from the Local Bus or writes data to the Local Bus from the Direct Slave Write FIFO.

The Direct Slave preempts DMA only when the Local Arbiter is in Direct Slave High-Priority mode (LARBR[3:1] =0116).

The IOP 480 can be programmed to "keep" the PCI Bus by generating a wait state(s) and de-asserting TRDY# if the Write FIFO becomes full. The IOP 480 can also be programmed to "keep" the Local Bus. LHOLD is asserted if the Direct Slave Write FIFO becomes empty or the Direct Slave Read FIFO becomes full. In either case, the Local Bus is dropped when the Local Bus Latency Timer is enabled and expires (LOCTMR[7:0]).

For Direct Slave writes, the PCI Bus writes data to the Local Bus. The Direct Slave is the "Command from the PCI Host," which has highest priority.

For Direct Slave reads, the PCI Bus Master reads data from the Local Bus Slave.

The IOP 480 supports on-the-fly Endian conversion for Direct Slave cycles. The Local Bus can be Big/Little Endian by programming the Memory Controller registers.

Note: *The PCI Bus is always Little Endian.*

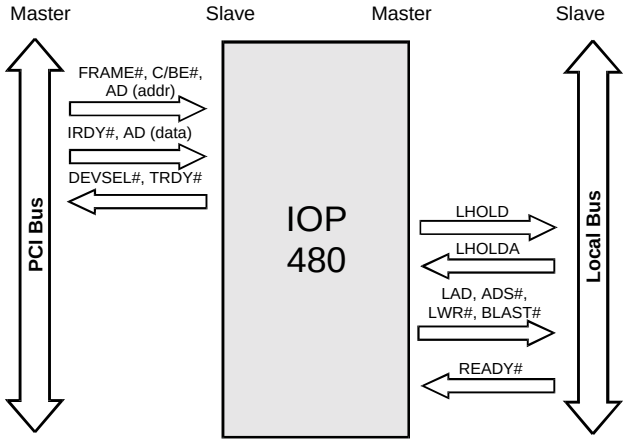


Figure 4-4. Direct Slave Write

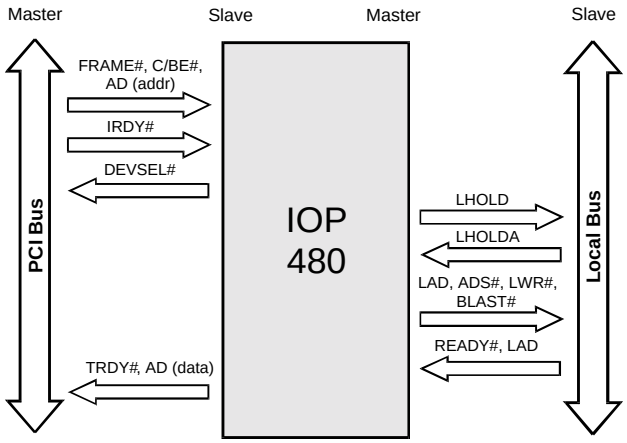


Figure 4-5. Direct Slave Read

Note: *The figures represents a sequence of bus cycles.*

4.8 LOCAL BUS BYTE ENABLES

During a Direct Slave transfer, each of four spaces (Space 0, Space 1, Space 2, and Expansion ROM) can be programmed to operate in an 8-, 16-, or 32-bit Local Bus width by encoding the Local Byte Enables (LBE[3:0]#).

LBE[3:0]# is encoded, based on the configured bus width, as follows.

32-Bit Bus—The four-byte enables indicate which of the four bytes are active during a data cycle.

- LBE3# Byte Enable 3—LAD[31:24]
- LBE2# Byte Enable 2—LAD[23:16]
- LBE1# Byte Enable 1—LAD[15:8]
- LBE0# Byte Enable 0—LAD[7:0]

16-Bit Bus—LBE3#, LBE1# and LBE0# are encoded to provide BHE#, LAD1, and BLE#, respectively.

- LBE3# Byte High Enable (BHE#)—LAD[15:8]
- LBE2# not used
- LBE1# Address bit 1 (LAD1)
- LBE0# Byte Low Enable (BLE#)—LAD[7:0]

8-Bit Bus—LBE1# and LBE0# are encoded to provide LAD1 and LAD0, respectively.

- LBE3# not used
- LBE2# not used
- LBE1# Address bit 1 (LAD1)
- LBE0# Address bit 0 (LAD0)

4.9 DIRECT SLAVE PRIORITY

If the Local Arbiter is set in Direct Slave High-Priority mode (LARBR[3:1]=011b), Direct Slave accesses have a higher priority than DMA accesses, thereby preempting DMA transfers. During a DMA transfer, if the IOP 480 detects a pending Direct Slave access, it gives up the Local Bus within two data transfers. The IOP 480 resumes operation after the Direct Slave access completes.

When the IOP 480 DMA controller owns the Local Bus, its LHOLD output and LHOLDA input are asserted. When a Direct Slave access occurs, the IOP 480 gives up the Local Bus within two Lword transfers by de-asserting LHOLD and floating the Local Bus outputs. After the IOP 480 acknowledges that LHOLDA is de-asserted, it requests the Local Bus for a Direct Slave transfer by asserting LHOLD. When the IOP 480 receives LHOLDA, it drives the bus and performs the Direct Slave transfer. Upon completing a Direct Slave transfer, the IOP 480 gives up the Local Bus by de-asserting LHOLD and floating the Local Bus outputs. After the IOP 480 samples LHOLDA as de-asserted, and the Local Bus Pause Timer is set to

zero, it requests a DMA transfer from the Local Bus by re-asserting LHOLD. When it receives LHOLDA, it drives the bus and continues the DMA transfer.

4.10 ALIGNMENT

As a Local Bus Master, the IOP 480 supports unaligned transfers.

For Direct Slave writes, the IOP 480 breaks any partial write into a single access. It only bursts on the Local Bus if all byte enables are asserted. If all bytes are disabled (LBE[3:0]# = Fh), the transfer is skipped.

For Direct Slave reads, regardless of the PCI byte enables, the IOP 480 reads all bytes on the Local Bus during a burst prefetch transaction. For single reads, the IOP 480 passes the byte enables.

Table 4-2. Direct Slave Local Byte Enable Methods

Direct Slave Methods	Byte Enables
DSR Burst	Don't Pass
DSR Single / I/O	Pass
DSW Burst	Breakup Burst for alignment
DSW Single / I/O	Pass
DSW to 8-bit bus	Skip if no byte enables allowed

4.11 DIRECT SLAVE EXAMPLE

A 1 MB Local Address Space 12300000h through 123FFFFFh is accessible from the PCI Bus at PCI addresses 78900000h through 789FFFFFh.

Local initialization software sets the Range and Local Base Address registers as follows:

- **Range**—FFF00000h (1 MB, decode the upper 12 PCI address bits)
- **Local Base Address (remap)**—123XXXXXh, (Local Base Address for PCI-to-Local accesses) [Space Enable bit(s) must be set to be recognized by the PCI Host, (LAS1BA[0]=1) and (LAS2BA[0]=1)] for Space 1 and Space 2; however, Space 0 and EROM Space are always set to enable [no Space Enable bit(s)]

PCI Initialization software writes all ones to PCI Base Address registers, then reads back the registers.

- The IOP 480 returns a value of FFF00000h. The PCI software then writes to the PCI Base Address register(s)
- **PCI Base Address**—789XXXXXh (PCI Base Address for Access to Local Address Space registers)

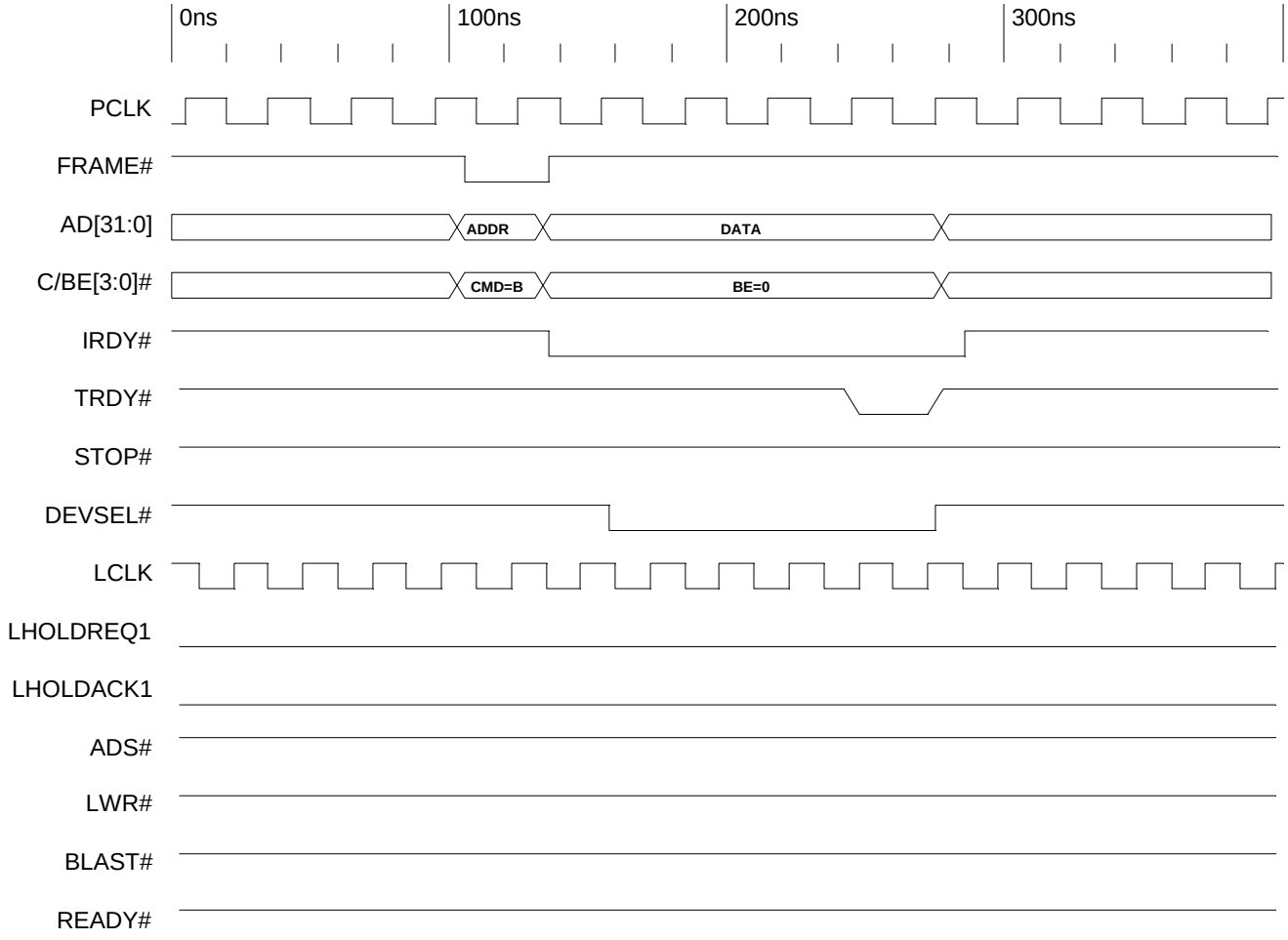
For a PCI Direct access to the Local Bus, the IOP 480 has a 32-Lword (128 bytes) Write FIFO and a 16-Lword (64 bytes) Read FIFO. FIFOs enable the Local Bus to operate independently of the PCI Bus. The IOP 480 can be programmed to return a Retry response or to throttle TRDY# for any PCI Bus transaction attempting to write to the IOP 480 Local Bus when the Write FIFO is full.

For PCI Read transactions from the Local Bus, the IOP 480 holds off TRDY# while gathering data from the Local Bus. For Read accesses mapped to PCI Memory space, the IOP 480 prefetches 0 (no prefetch), 4, 8, or 16 Lwords in Read Prefetch Count Enable mode, or prefetches continuously from the Local Bus until terminated by the PCI Bus or until a page boundary is reached. Unused Read data is flushed from the FIFO. For Read accesses mapped to PCI I/O space, the IOP 480 does not prefetch Read data. Rather, it breaks each read of a Burst cycle into a Single Address/Data cycle on the Local Bus.

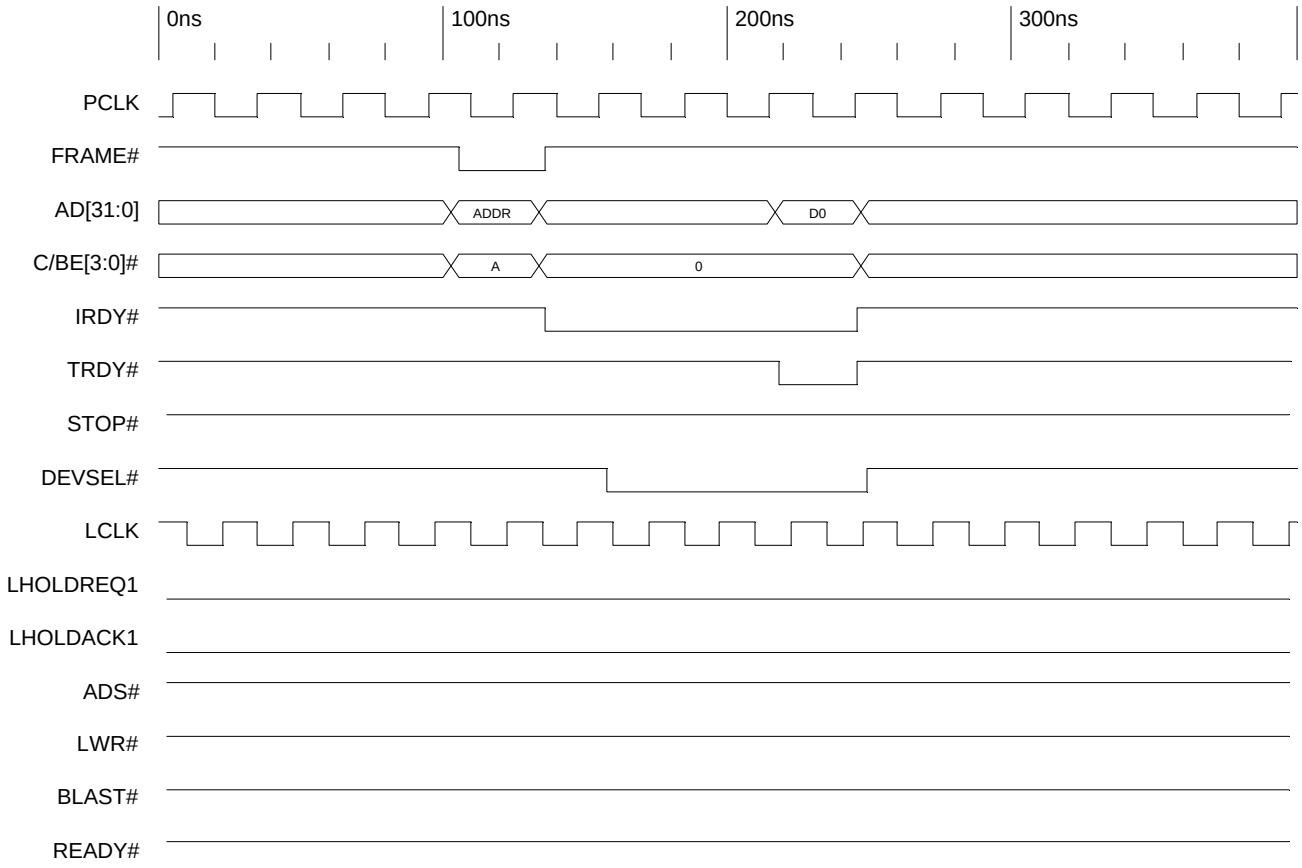
The Direct Slave Retry Delay Clocks bits (PCICTL[31:28]) can be used to program the time period in which the IOP 480 holds off asserting TRDY#. The IOP 480 issues a Retry to the PCI Bus Transaction Master when the programmed time period expires. This occurs when the IOP 480 cannot gain control of the Local Bus and return TRDY# within the programmed time period.

4.12 TIMING DIAGRAMS

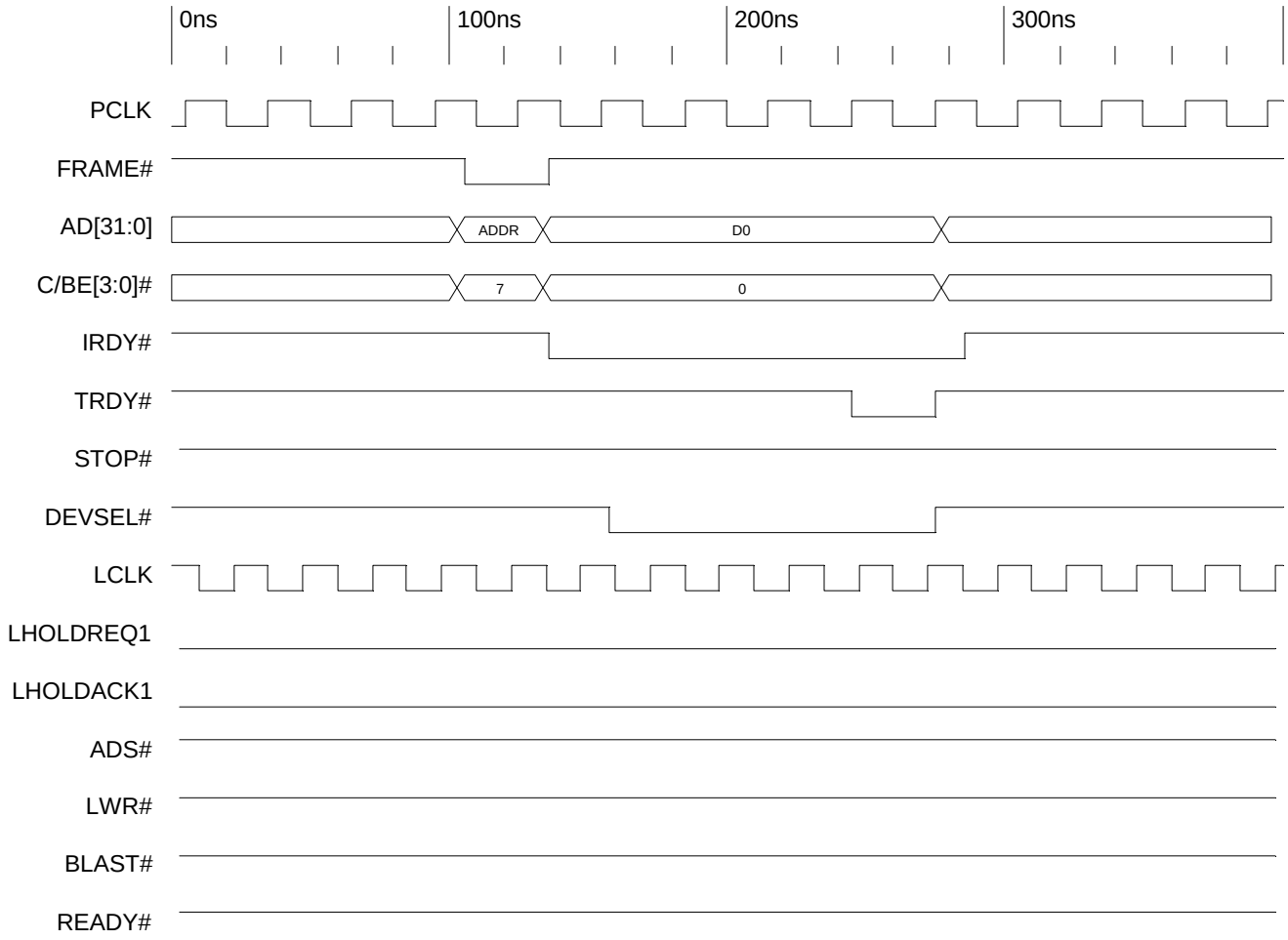
4.12.1 Direct Slave Configuration Cycle



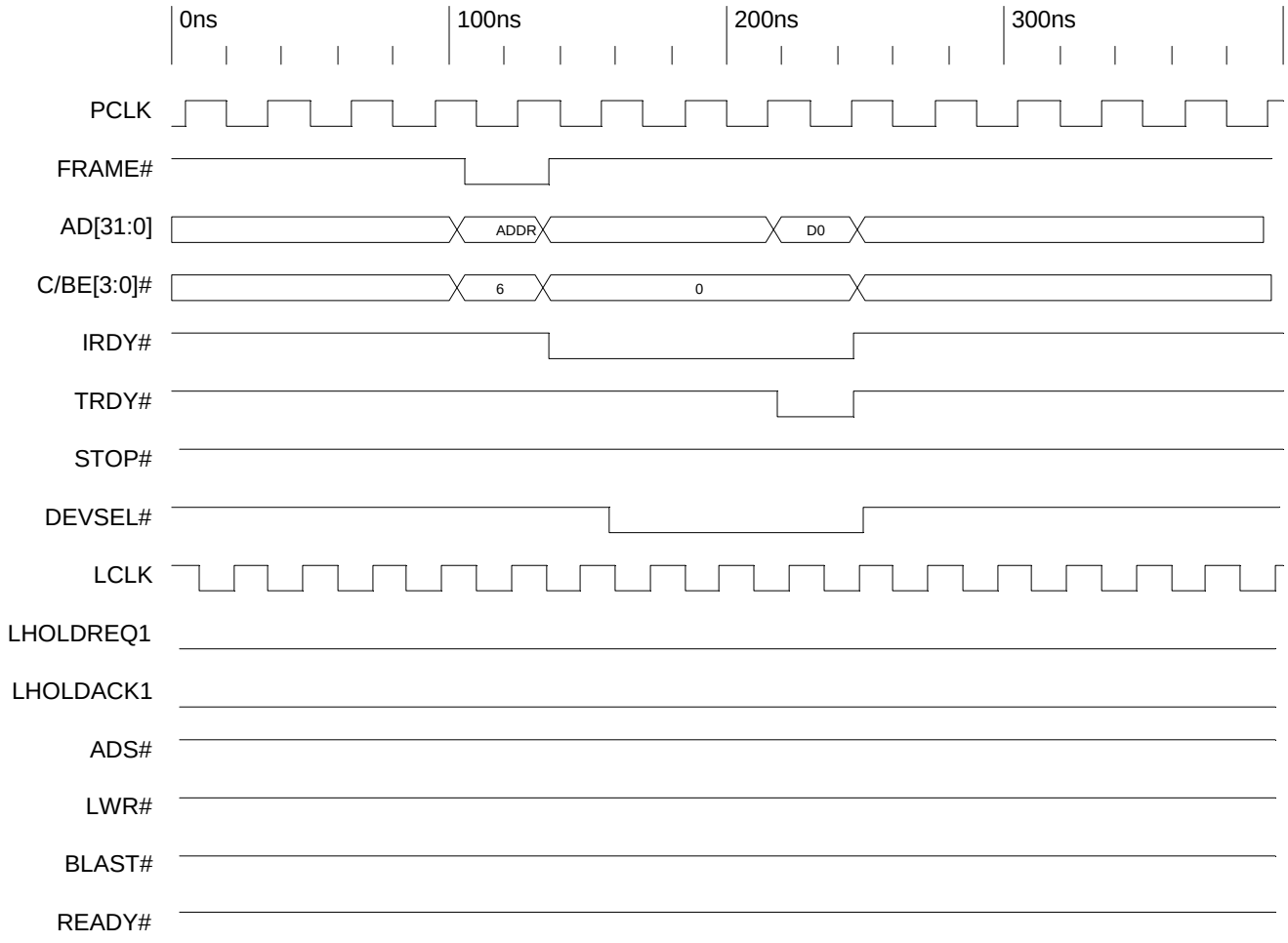
Timing Diagram 4-1. PCI Configuration Write



Timing Diagram 4-2. PCI Configuration Read

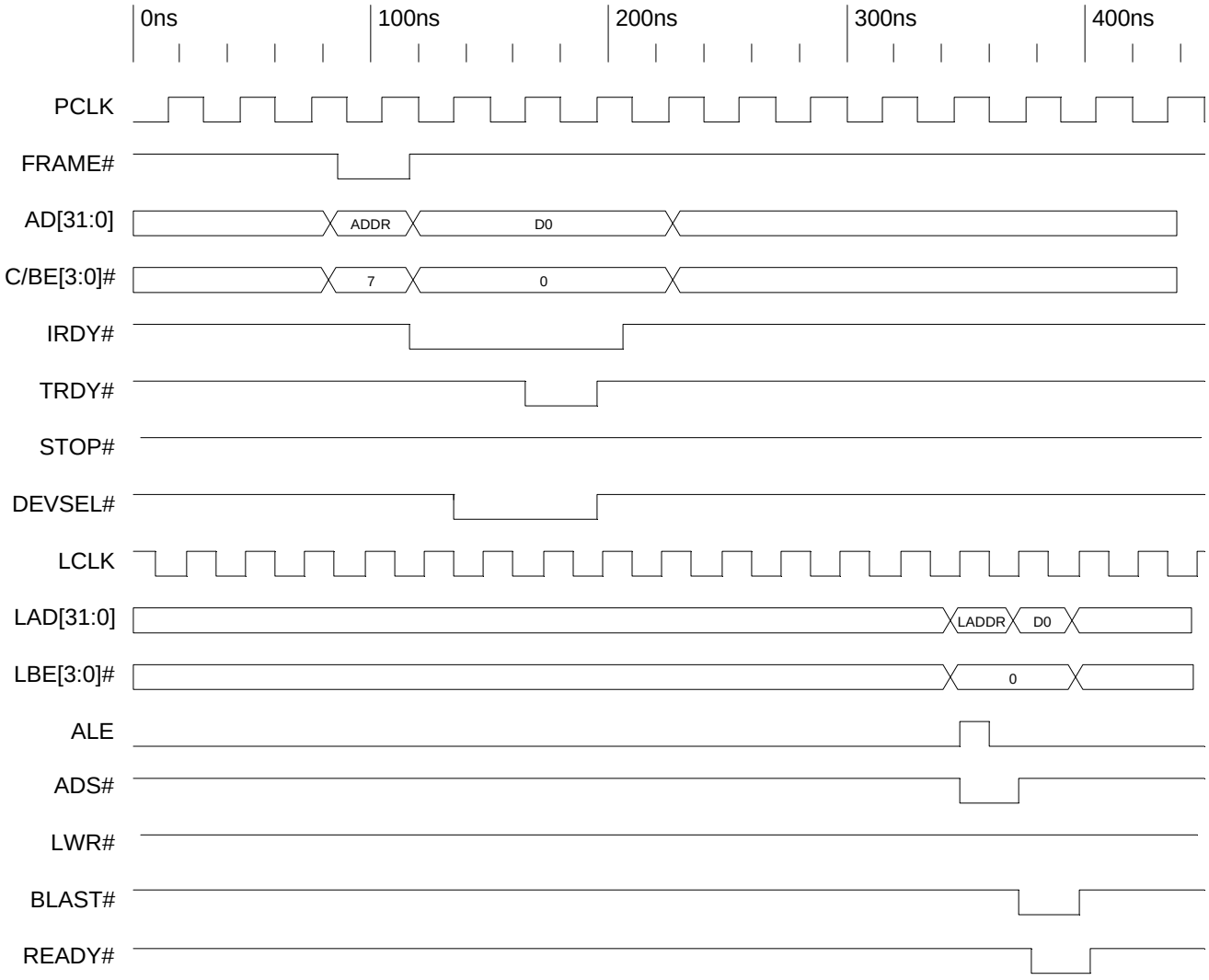


Timing Diagram 4-3. PCI Memory Write

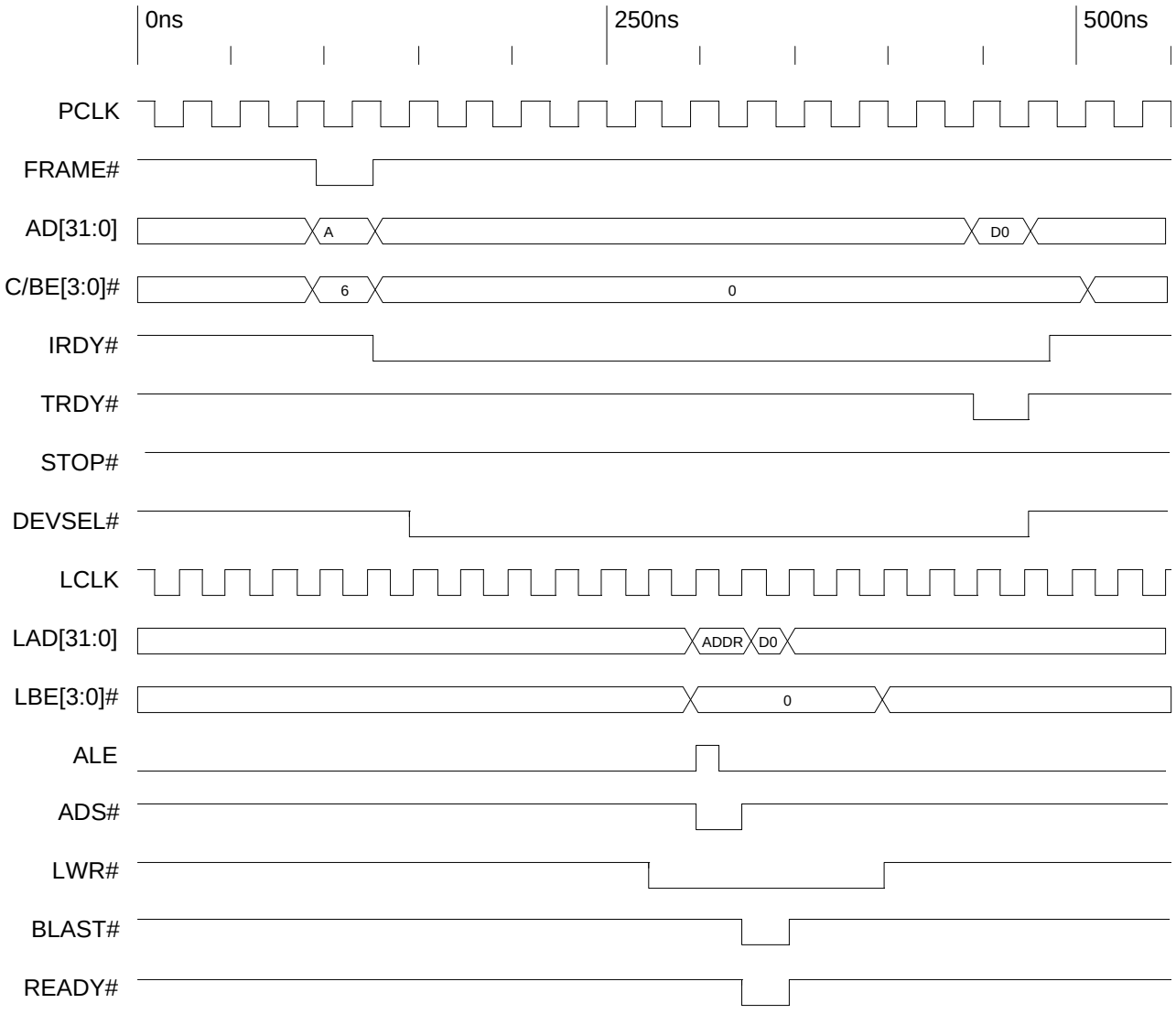


Timing Diagram 4-4. PCI Memory Read

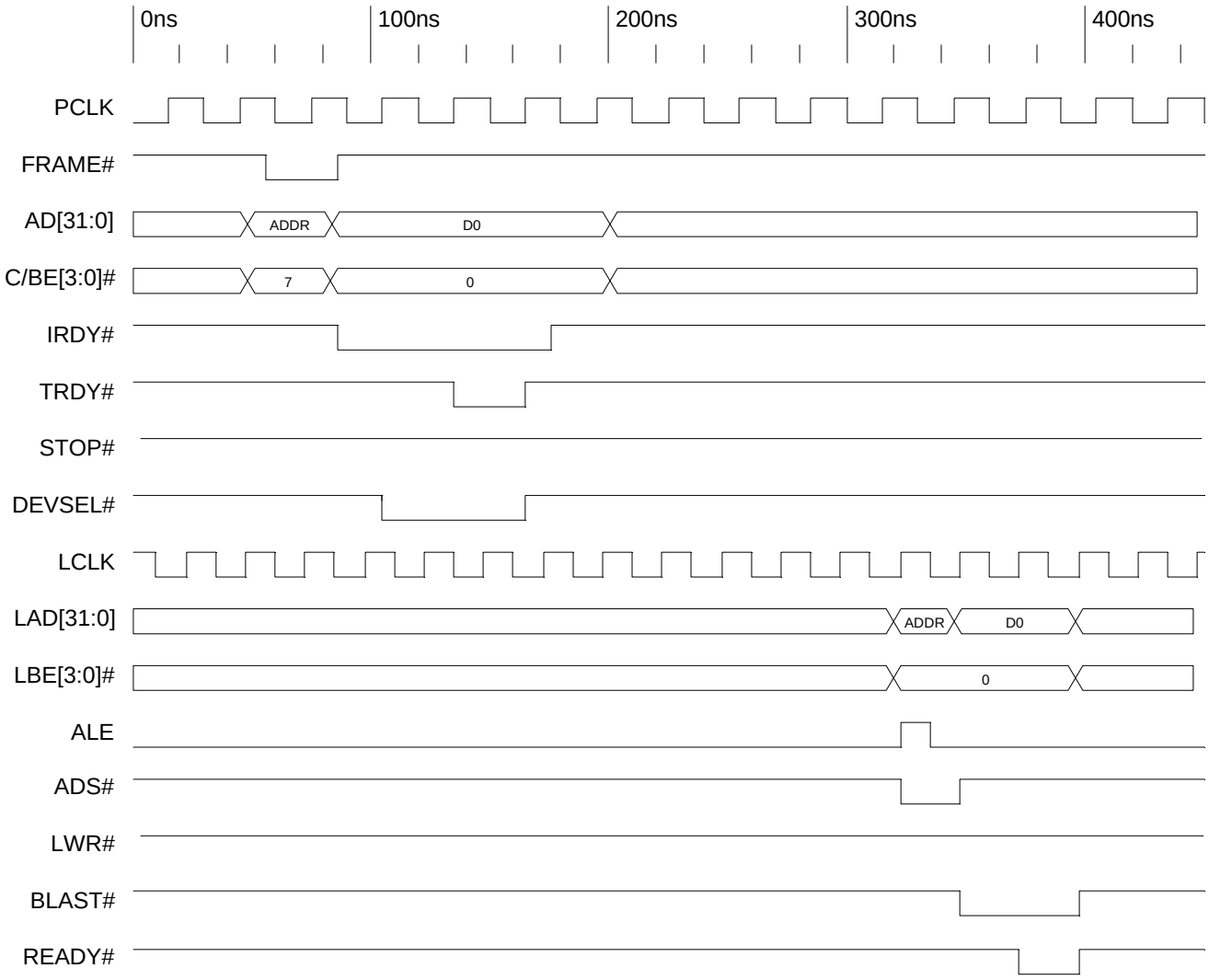
4.12.2 Direct Slave



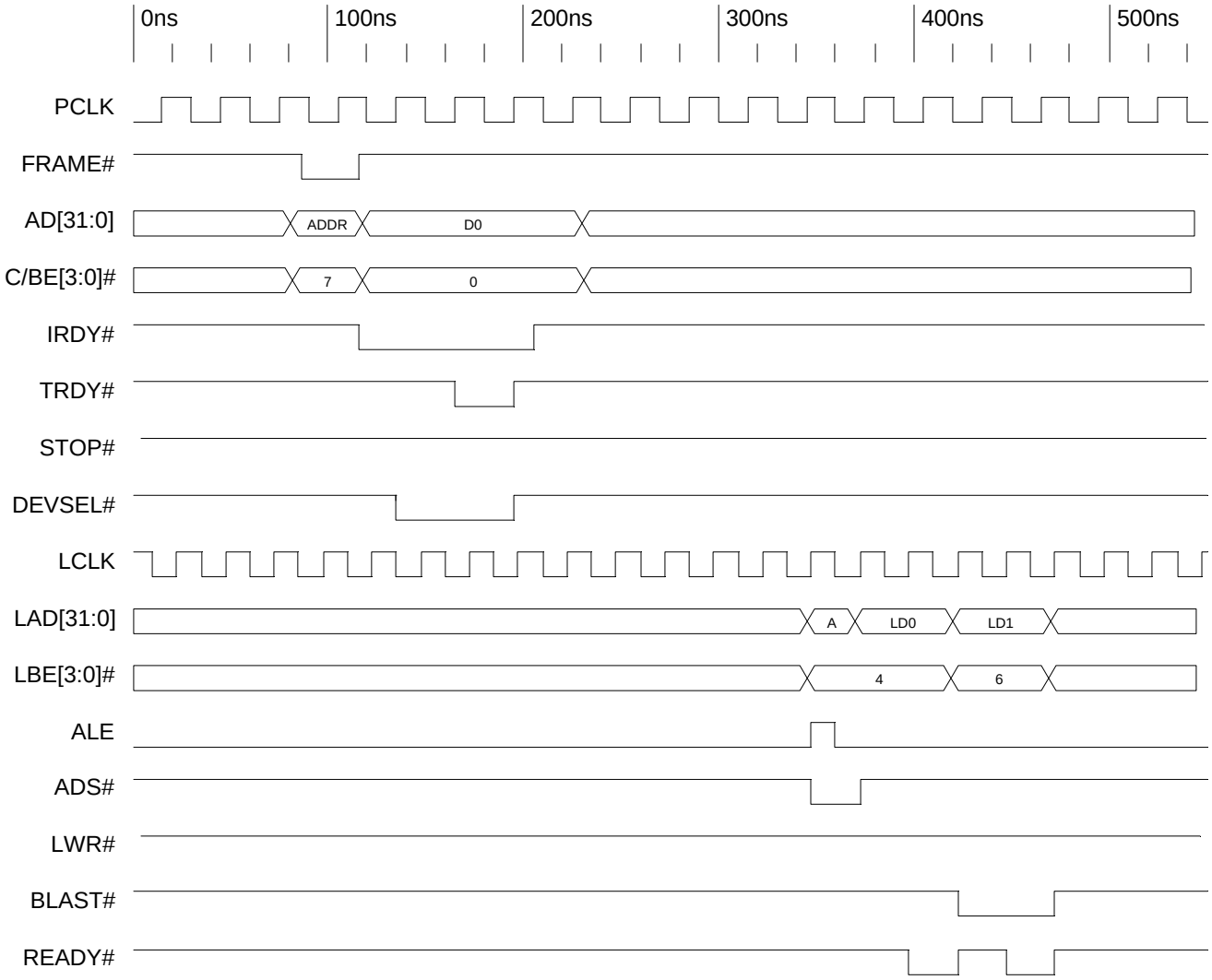
Timing Diagram 4-5. Direct Slave Write to 32-Bit Local Bus with Zero Wait States



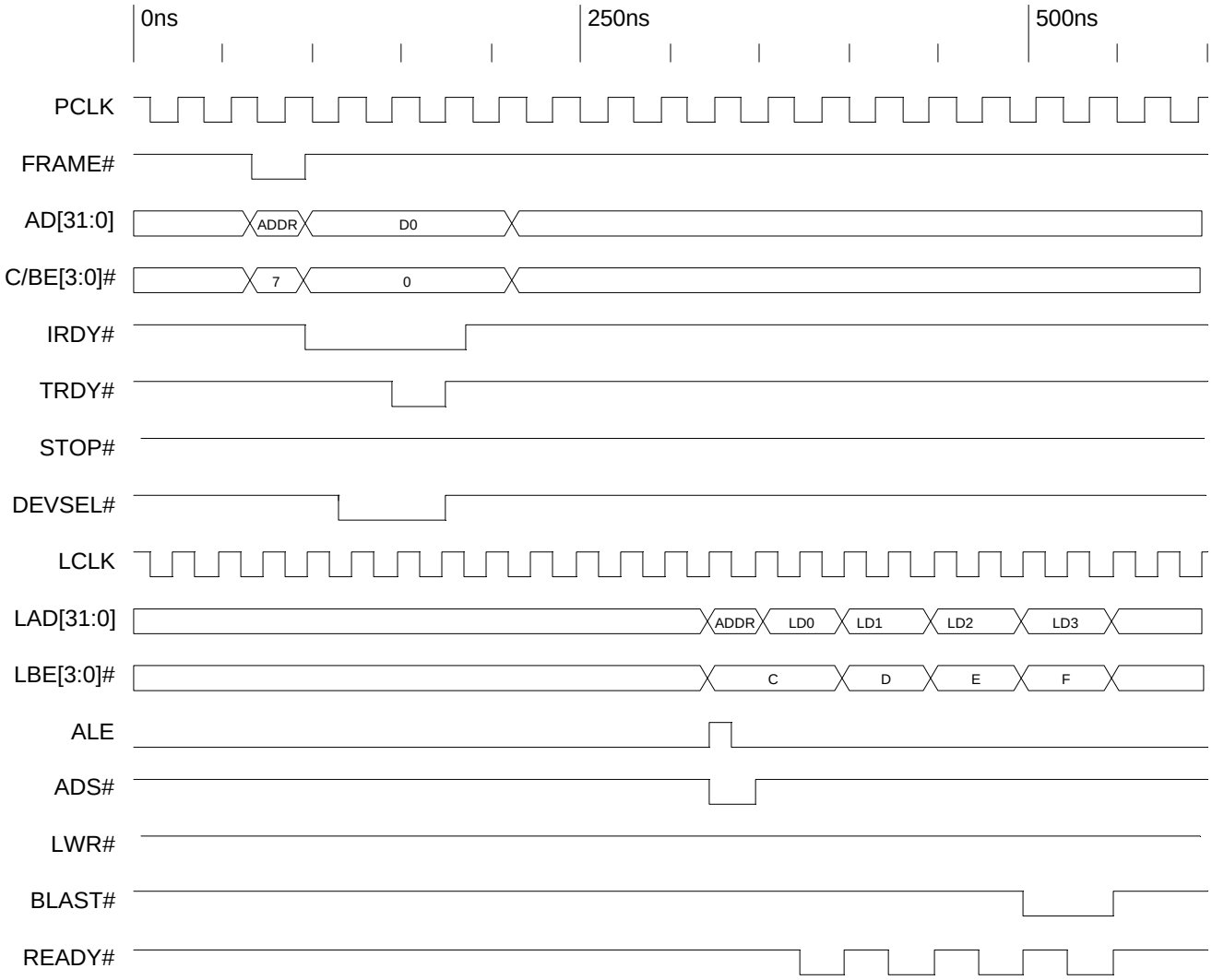
Timing Diagram 4-6. Direct Slave Read from 32-Bit Local Bus with Zero Wait States



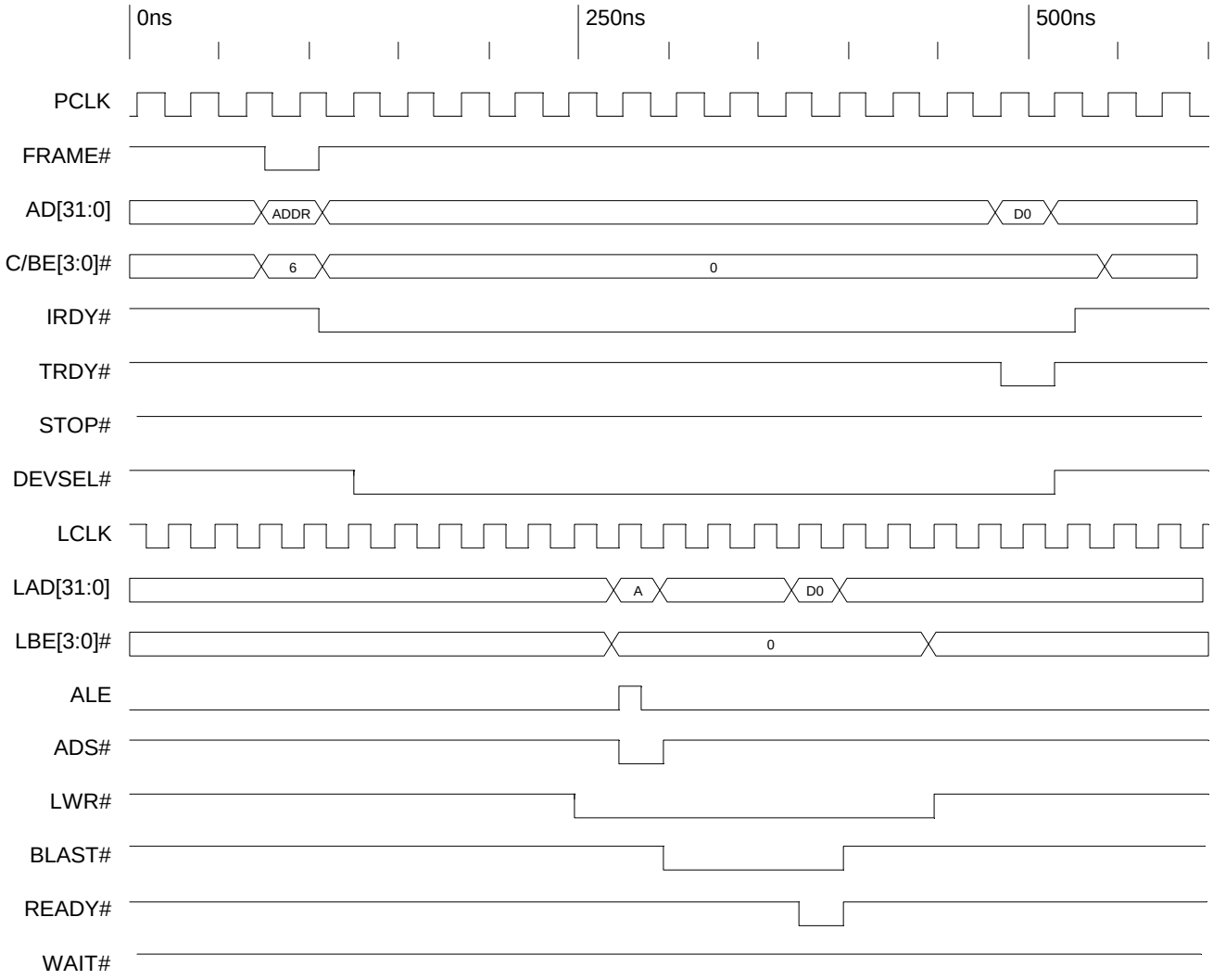
Timing Diagram 4-7. Direct Slave Write to 32-Bit Local Bus with One External (READY#) Wait State



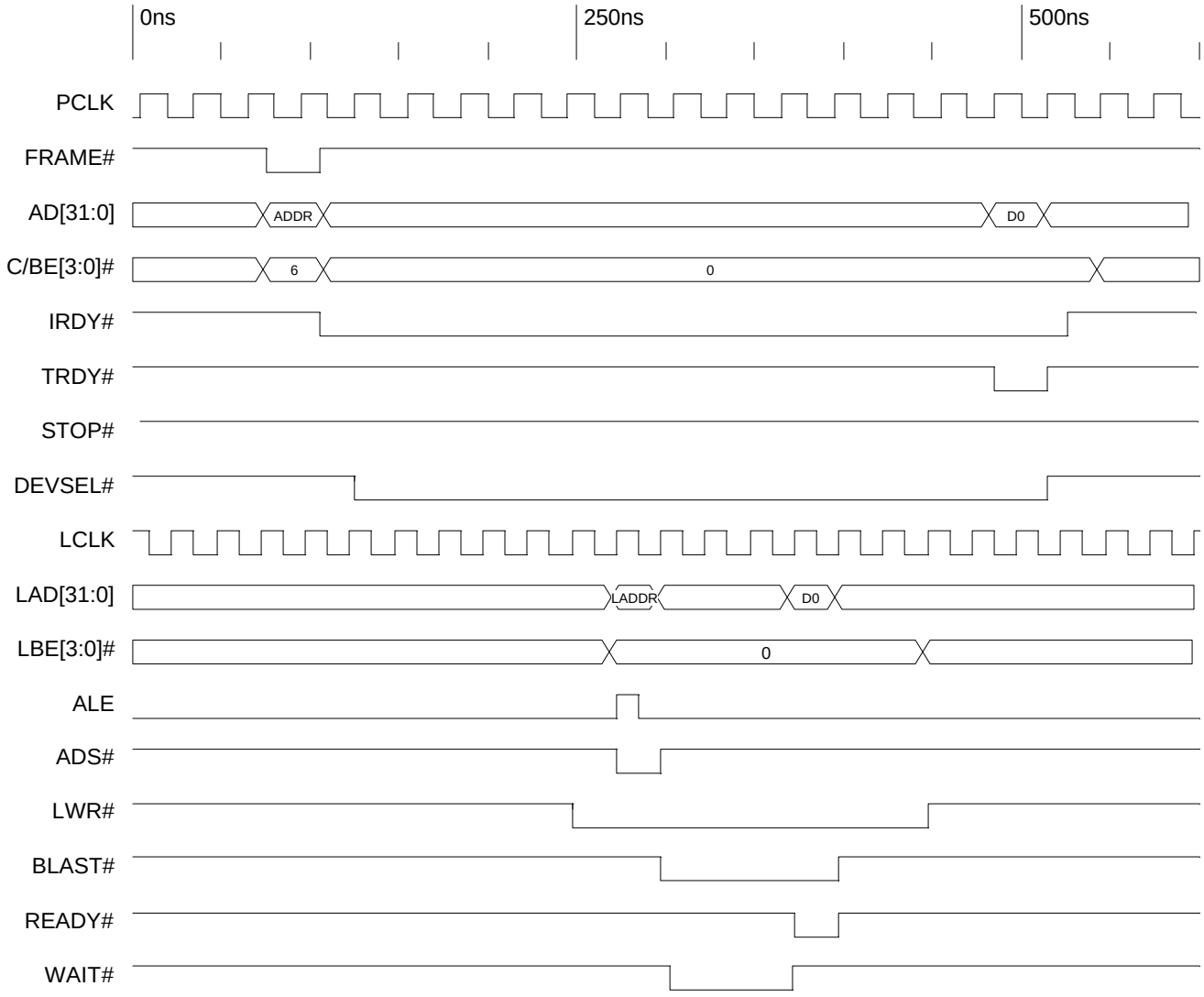
Timing Diagram 4-8. Direct Slave Write to 16-Bit Local Bus with One External (READY#) Wait State



Timing Diagram 4-9. Direct Slave Write to 8-Bit Local Bus with One External (READY#) Wait State

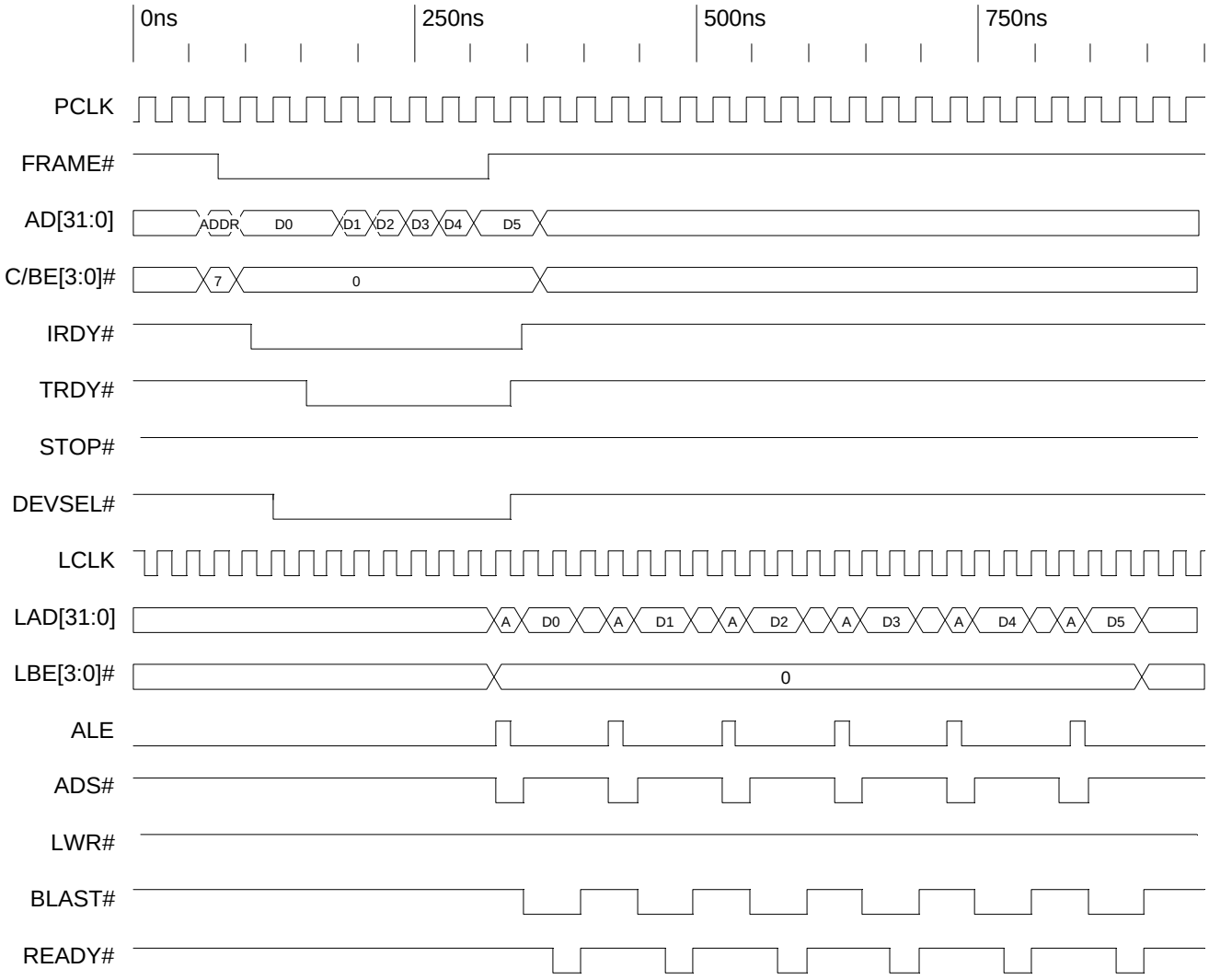


Timing Diagram 4-10. Direct Slave Read from 32-Bit Local Bus with Three External (READY#) Wait States

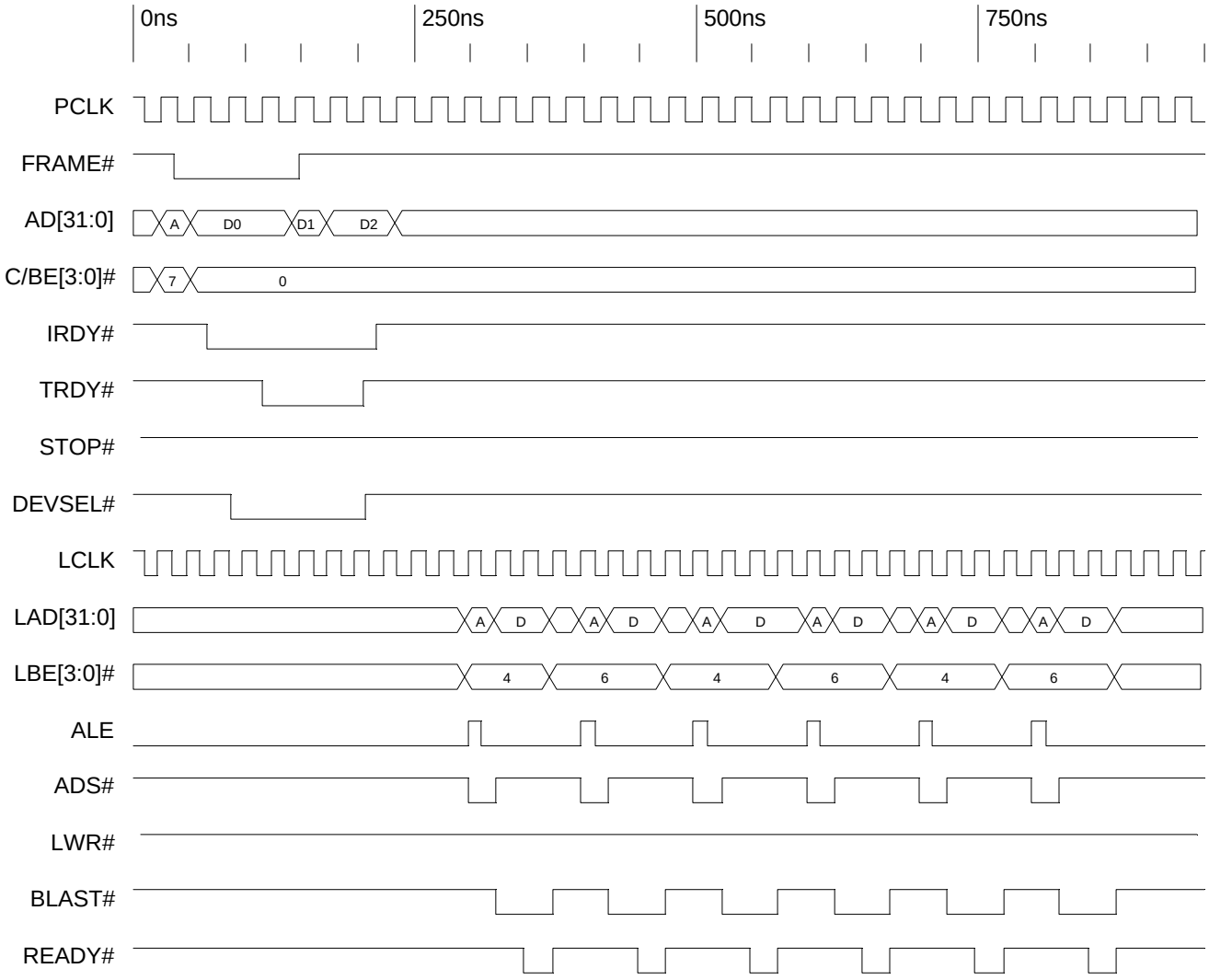


Note: BLAST# timing shown assumes LOCCTL[19]=0. If LOCCTL[19]=1, BLAST# assertion is delayed until WAIT# is de-asserted.

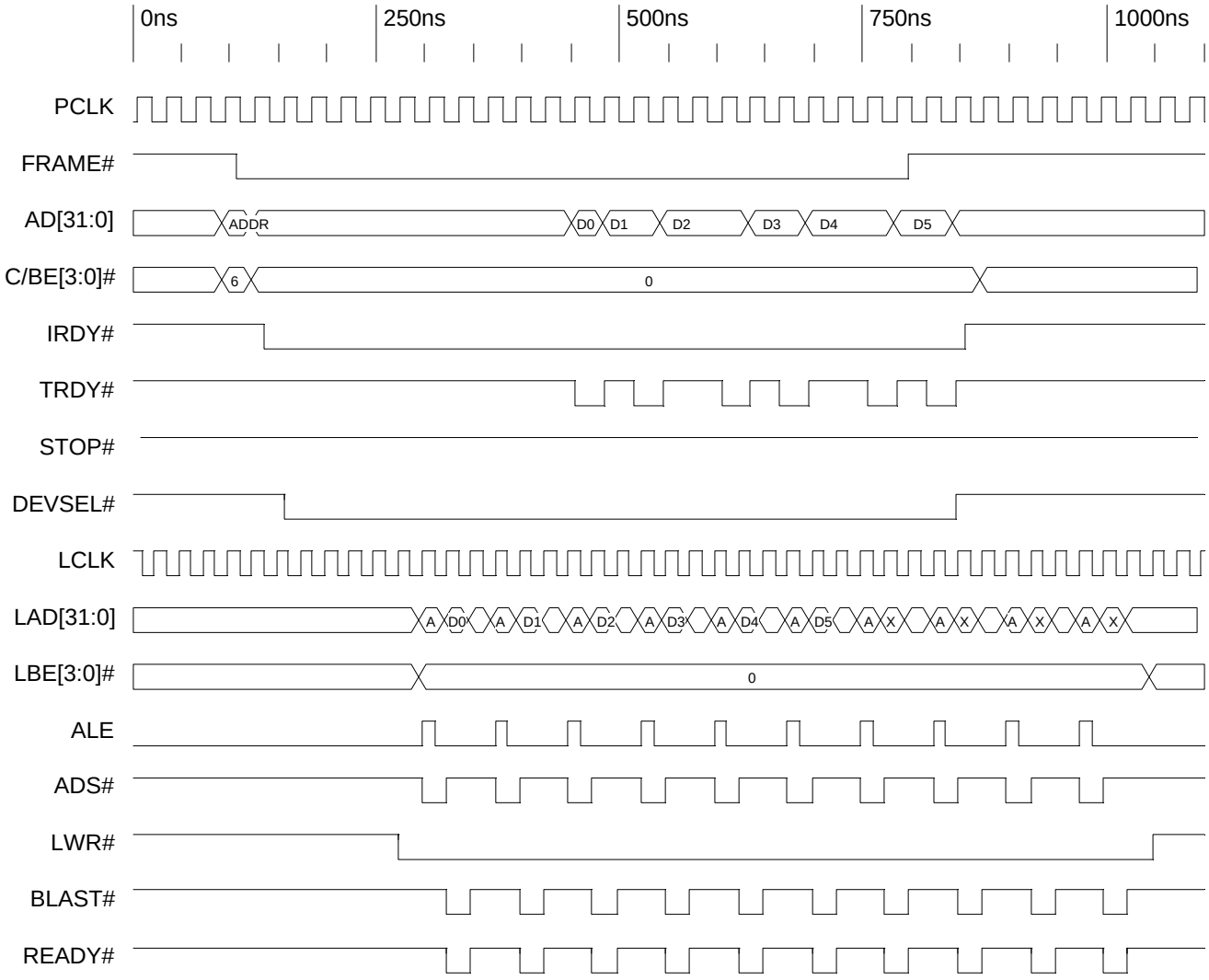
Timing Diagram 4-11. Direct Slave Read from 32-Bit Local Bus with Three Internal (WAIT#) Wait States



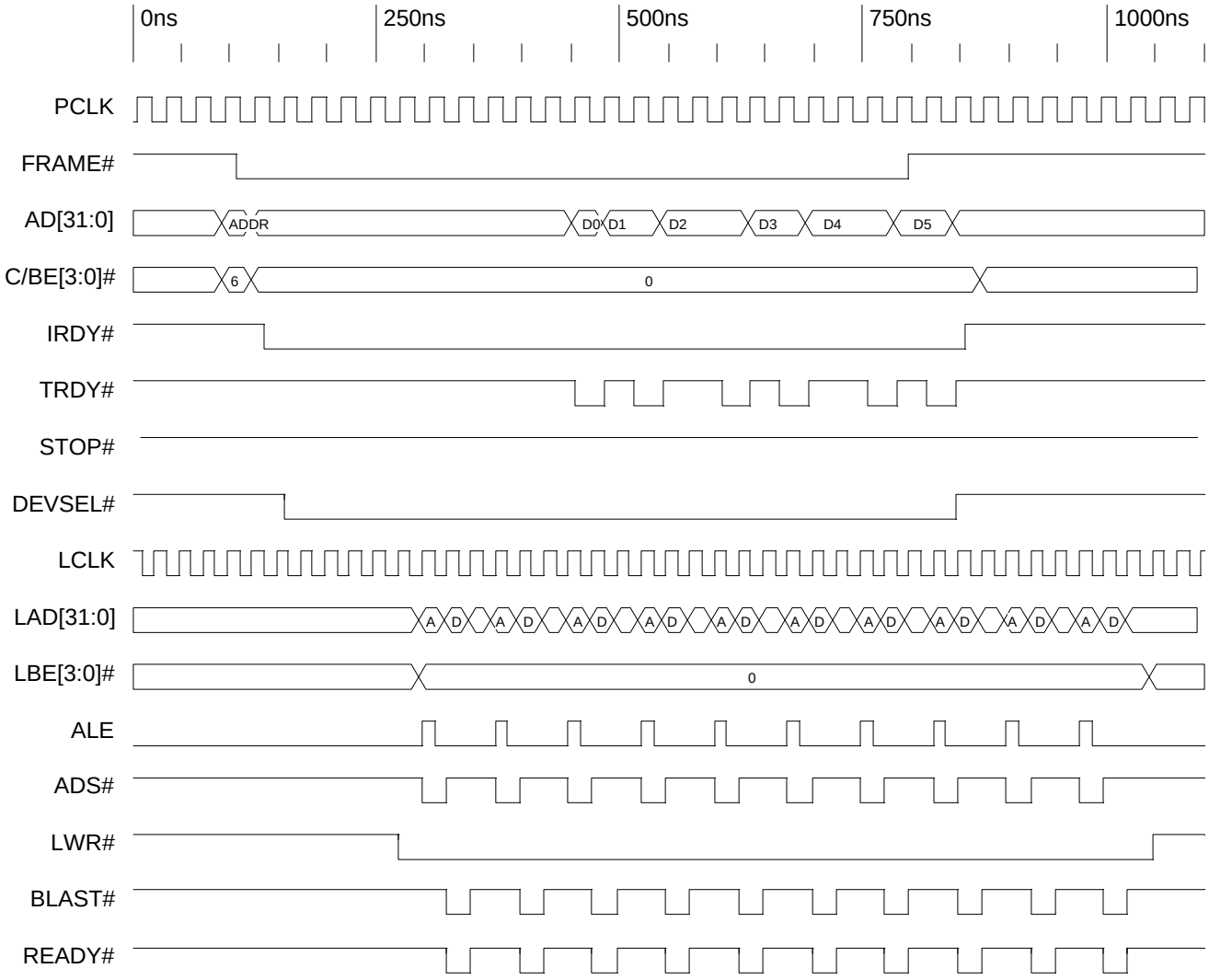
Timing Diagram 4-12. Direct Slave PCI Write of Six Lwords to 32-Bit Local Bus, Burst Disabled



Timing Diagram 4-13. Direct Slave PCI Write of Three Lwords to 16-Bit Local Bus, Burst Disabled

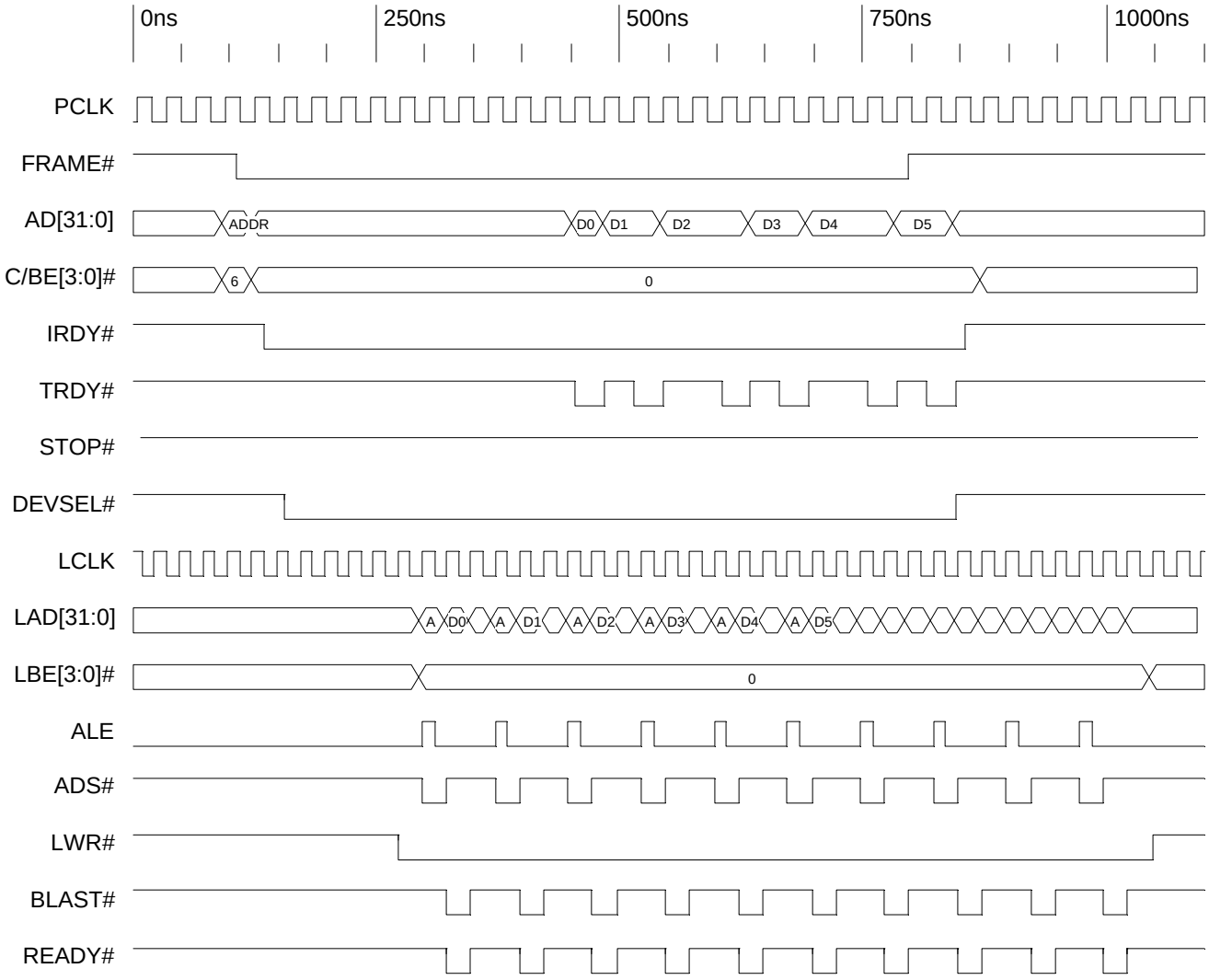


Timing Diagram 4-14. Direct Slave PCI Read of Six Lwords to 32-Bit Local Bus, Burst Disabled

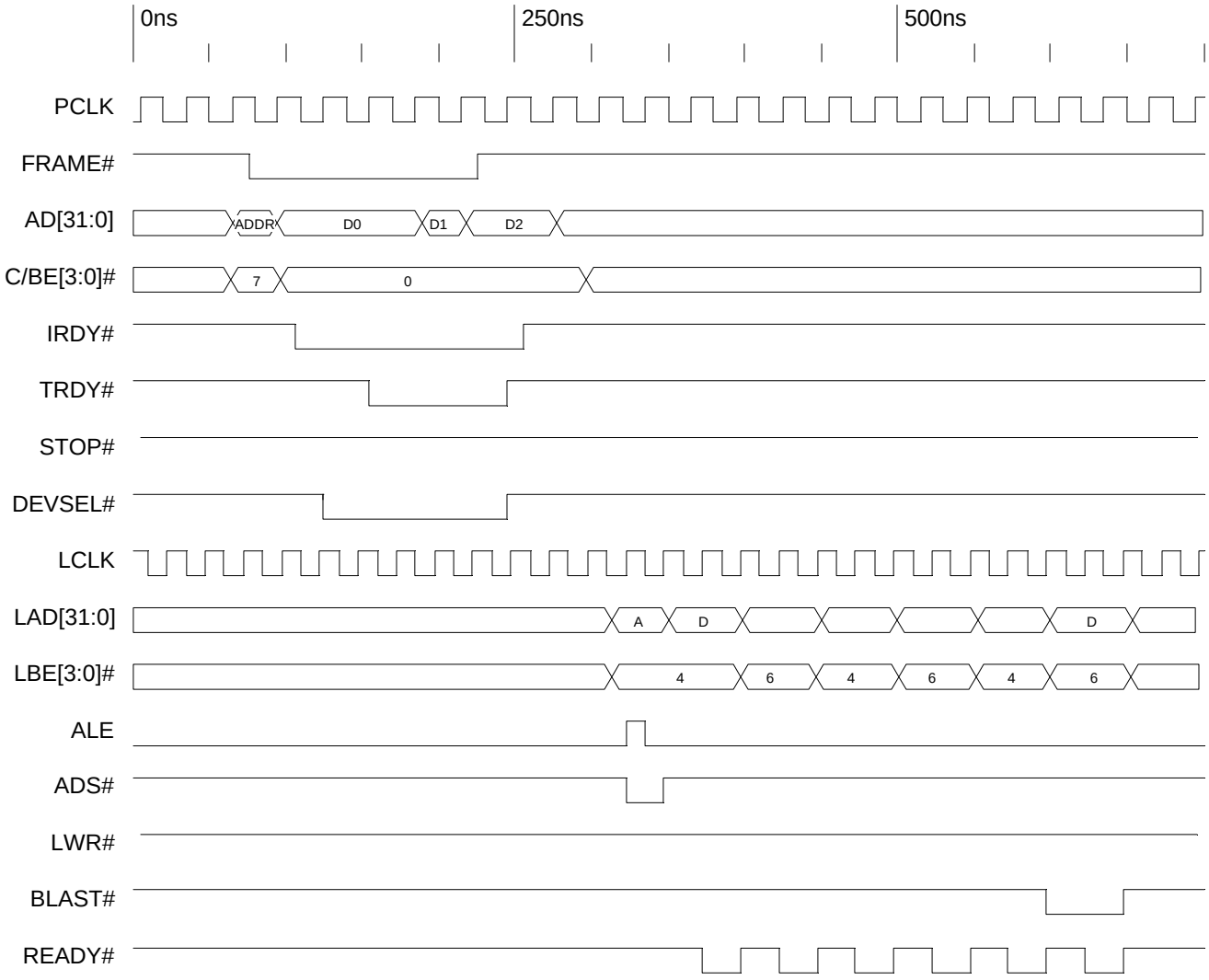


Timing Diagram 4-15. Direct Slave PCI Write of Six Lwords to 32-Bit Local Bus, Burst Disabled, One External (READY#) Wait State

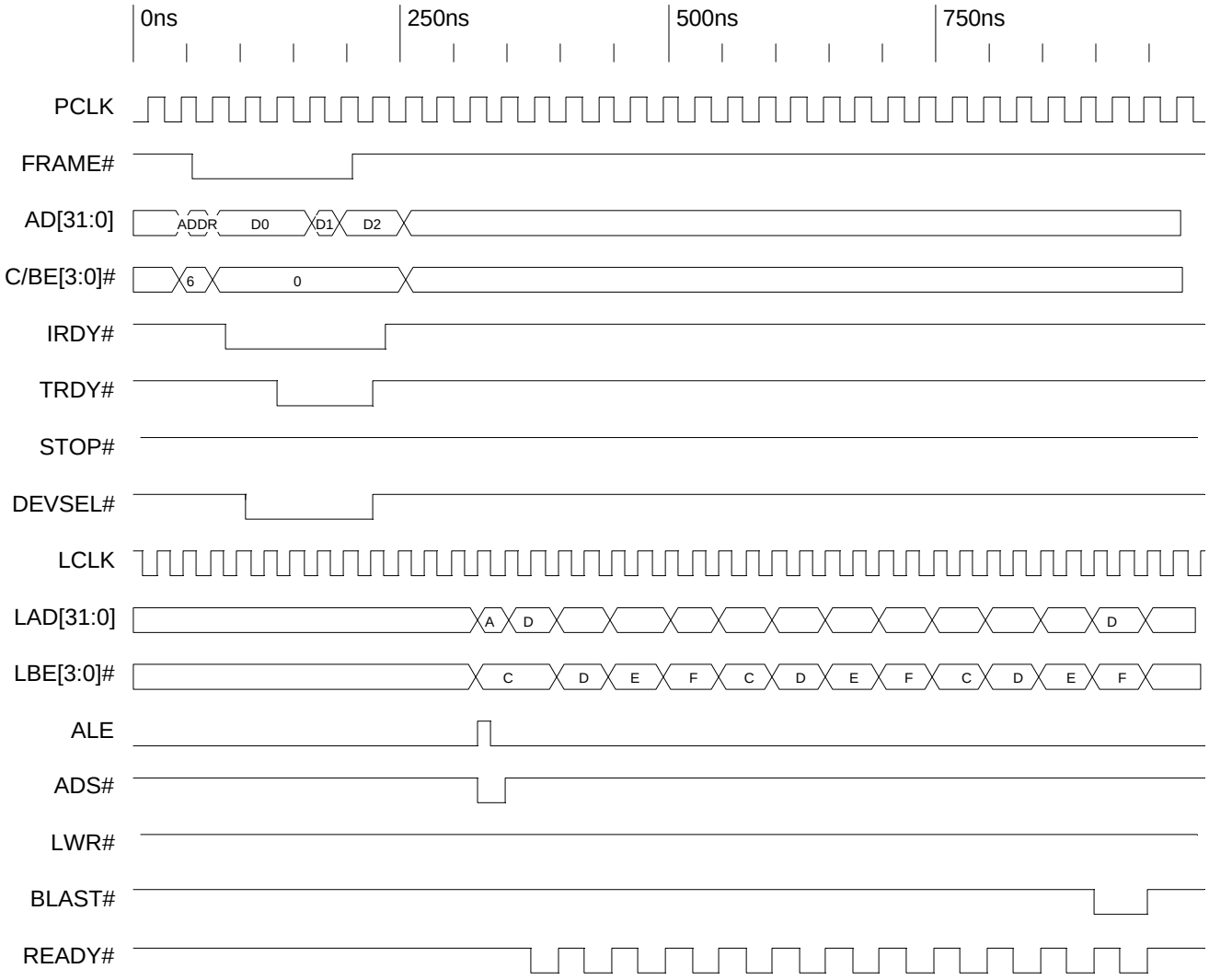
4.12.3 Direct Slave Burst



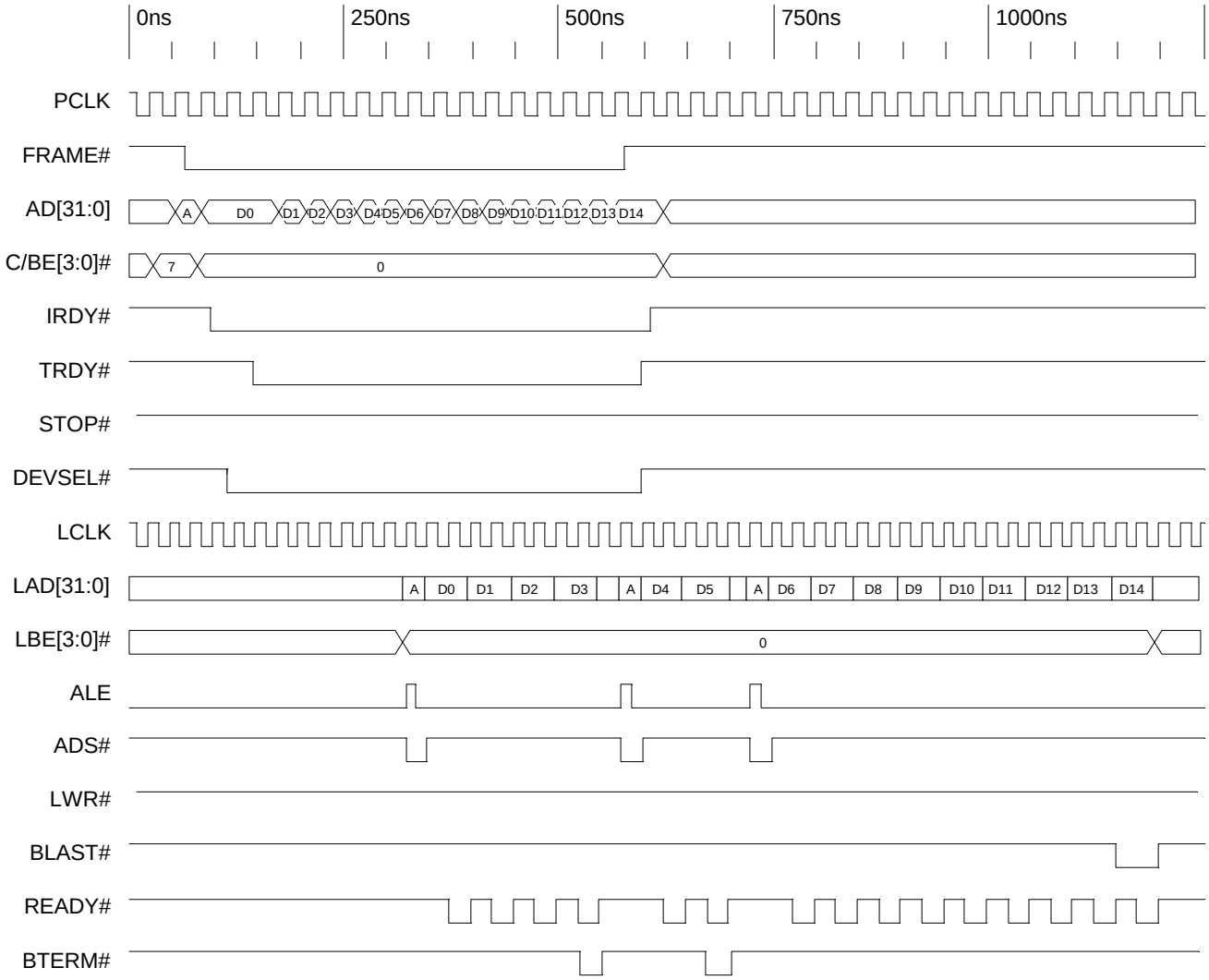
Timing Diagram 4-16. Direct Slave PCI Write of Six Lwords to 32-Bit Local Bus, Local Burst Enabled, One External (READY#) Wait State



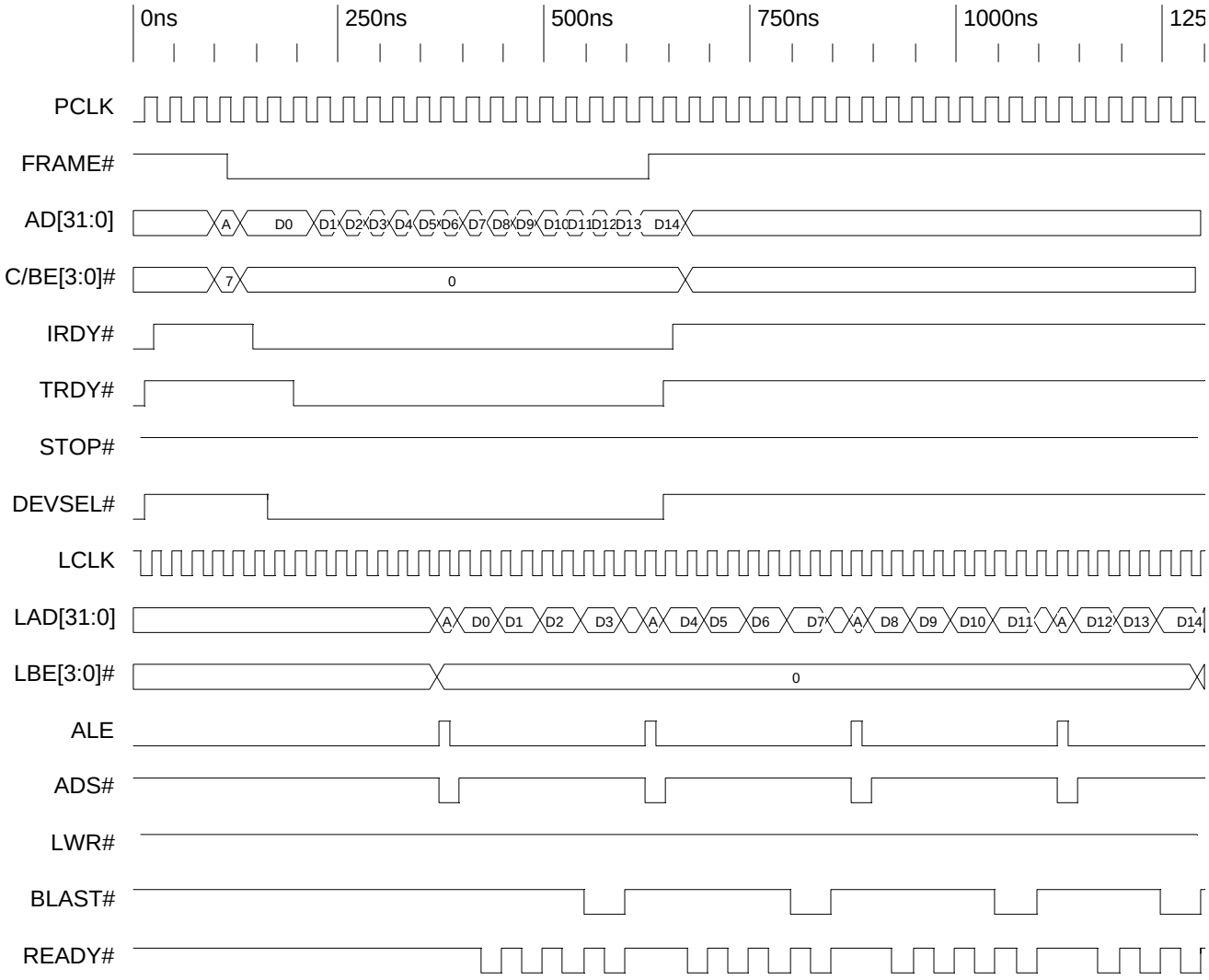
Timing Diagram 4-17. Direct Slave PCI Write of Six Lwords to 16-Bit Local Bus, Local Burst Enabled, One External (READY#) Wait State



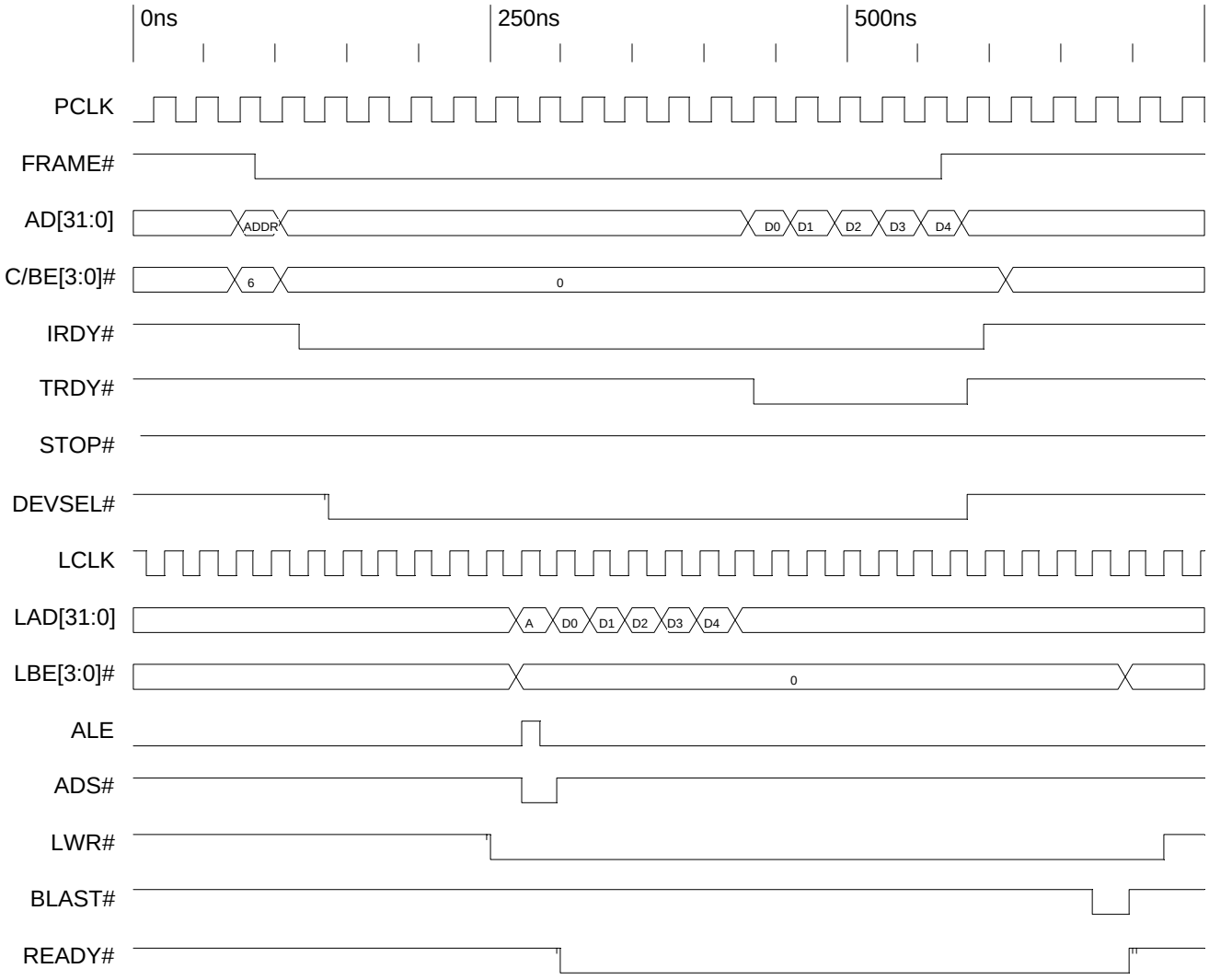
Timing Diagram 4-18. Direct Slave PCI Write of Three Lwords to 8-Bit Local Bus, Local Burst Enabled, One External (READY#) Wait State



Timing Diagram 4-19. Direct Slave PCI Write of 15 Lwords to 32-Bit Local Bus, Local Burst Enabled, One External (READY#) Wait State, Bterm Enabled



Timing Diagram 4-20. Direct Slave PCI Write of 15 Lwords to 32-Bit Local Bus, Local Burst Enabled, One External (READY#) Wait State, Bterm Disabled



Timing Diagram 4-21. Direct Slave Burst Read of Five Lwords to 32-Bit Local Bus, Burst Enabled, Prefetch Counter of 16, Zero Wait States

5 DIRECT MASTER OPERATION

5.1 OVERVIEW

Direct Master operations originate on the Local Bus, go through the IOP 480, and finally access the PCI Bus. The IOP 480 will be a Local Bus Slave and a PCI Bus Master. All Direct Master operations require a 32-bit wide bus region.

5.1.1 Direct Master

Local Bus cycles can be continuous Single or Burst cycles (programmable through the IOP 480 internal registers). As a Local Bus Target, the IOP 480 allows access to the IOP 480 internal registers and the PCI Bus.

5.1.2 Direct Master Operation (Local Master-to-Direct Slave)

The IOP 480 supports a direct access of the PCI Bus by the Local processor or an intelligent controller. Direct Master mode must be enabled in the PCI Command register. The following registers define Local-to-PCI accesses:

- Direct Master Memory and I/O Range (DMRR)
- Local Base Address for Direct Master-to-PCI Memory (DMLBAM)
- Local Base Address for Direct Master-to-PCI I/O and Configuration (DMLBAI)
- PCI Base Address (DMPBAM)
- Direct Master Configuration (DMCFGA)
- Direct Master Dual Address Cycles (DMDAC)

5.1.2.1 Direct Master Memory and I/O Decode

The Range register and Local Base Address specifies the local address bits to use for decoding a Local-to-PCI access. The range of Memory or I/O space must be a power of 2 and the Range register value must be the inverse of the Range value. In addition, the Local Base Address must be a multiple of the range value.

Any Local Master Address starting from the Direct Master Local Base Address (Memory or I/O) to the range value is recognized as a Direct Master access by the IOP 480. All Direct Master cycles are then decoded as PCI memory, I/O, or Configuration Type 0 or Type 1. Moreover, a Direct Master Memory or I/O cycle is remapped according to the Remap register value. The Remap register value must be a multiple of the Direct Master Range value.

The Local Range registers determine whether to remap addresses to PCI memory or to PCI I/O space.

5.1.3 PCI Command Codes

For direct Local-to-PCI Bus accesses, the IOP 480 asserts the cycles listed in the following tables.

Table 5-1. Local-to-PCI Memory Access

Command Type	Code (C/BE[3:0]#)
Memory Read	0110 (6h)
Memory Write	0111 (7h)
Memory Read Multiple	1100 (Ch)
Dual Address Cycle	1101 (Dh)
Memory Read Line	1110 (Eh)
Memory Write and Invalidate	1111 (Fh)

Table 5-2. Local-to-PCI I/O Access

Command Type	Code (C/BE[3:0]#)
I/O Read	0010 (2h)
I/O Write	0011 (3h)

Table 5-3. Local-to-PCI Configuration Access

Command Type	Code (C/BE[3:0]#)
Configuration Memory Read	1010 (Ah)
Configuration Memory Write	1011 (Bh)

5.2 INTERNAL FIFOS

For Direct Master Memory accesses to the PCI Bus, the IOP 480 has a 32-Lword (128-byte) Write FIFO and a 16-Lword (64-byte) Read FIFO. The FIFOs enable the Local Bus to operate independently with the PCI Bus and allow high-performance bursting on the PCI and Local Buses. For a Direct Master Write, the Local processor (master) writes data to the PCI Bus (slave). For a Direct Master Read, the Local processor (master) reads data from the PCI Bus (slave). FIFOs functioning during a Direct Master write and read are illustrated in Figure 5-1 and Figure 5-2. (Refer also to Table 5-4.)

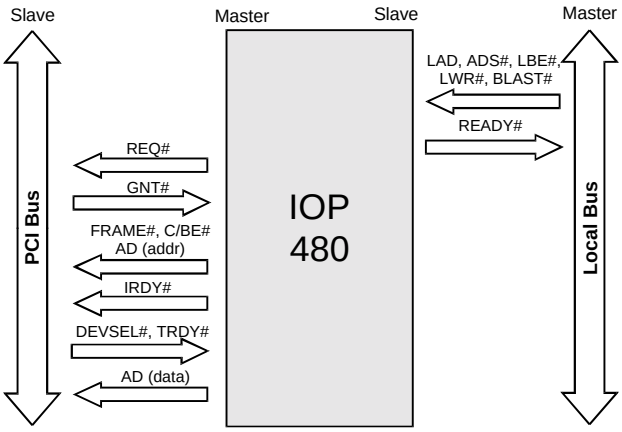


Figure 5-1. Direct Master Write

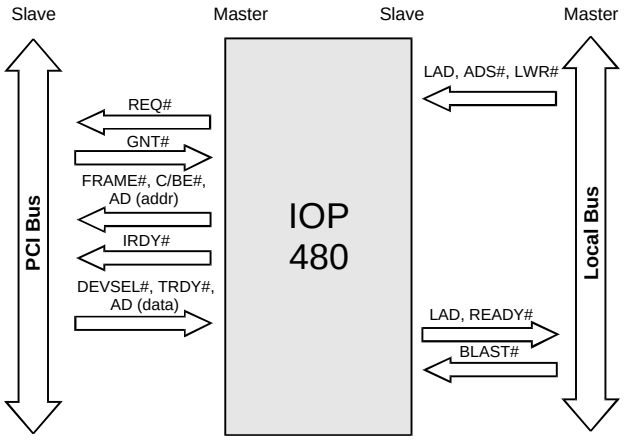


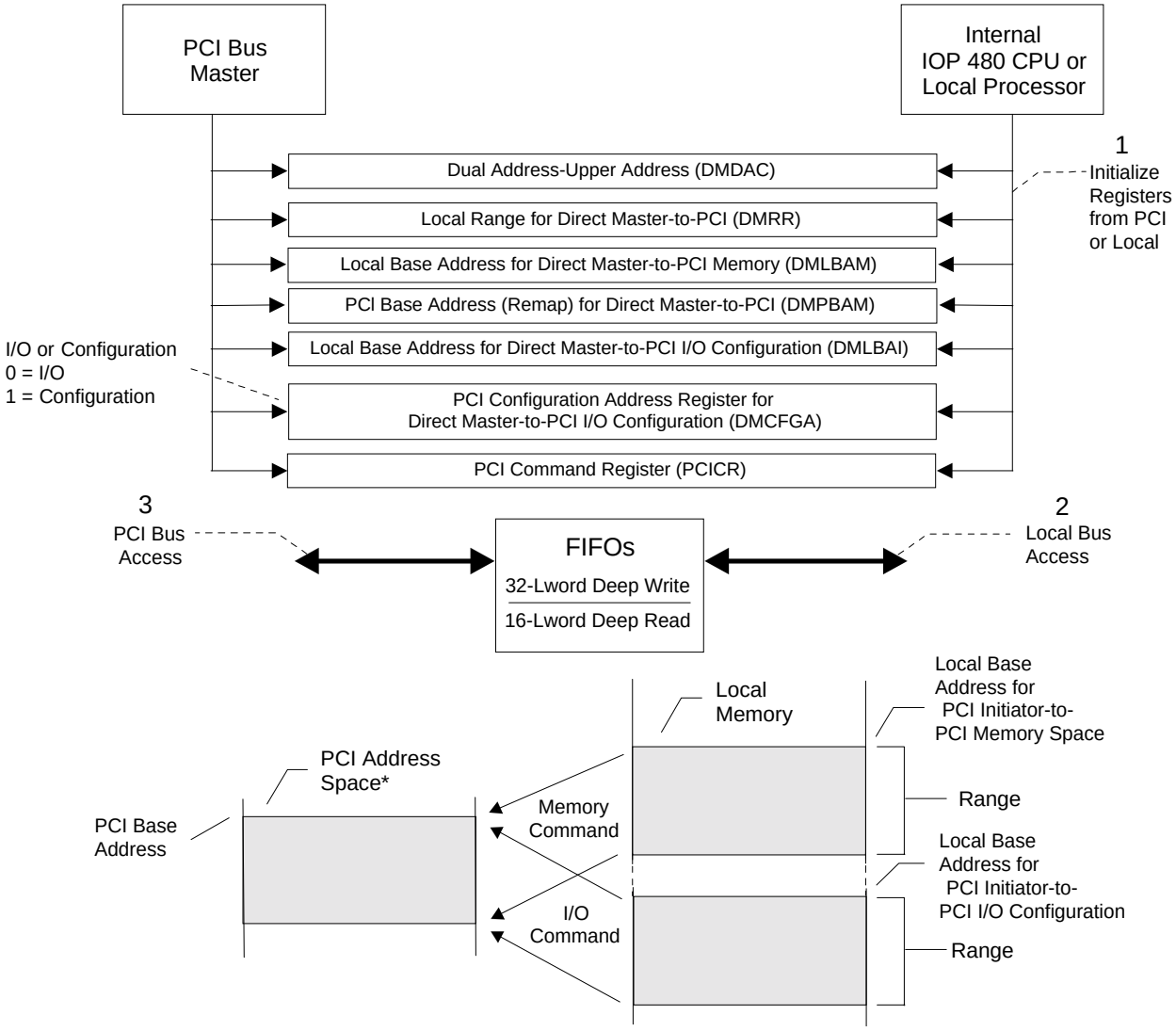
Figure 5-2. Direct Master Read

Note: *The figures represent a sequence of bus cycles.*

Table 5-4. Response to FIFO Full or Empty

Mode	Data Direction	FIFO	PCI Bus	Local Bus
Direct Master Write	Local-to-PCI	Full	Normal	De-assert READY#
		Empty	De-assert REQ# (off PCI Bus)	Normal
Direct Master Read	PCI-to-Local	Full	De-assert REQ# or throttle IRDY#[1]	Normal
		Empty	Normal	De-assert READY#
Direct Slave Write	PCI-to-Local	Full	Disconnect or throttle TRDY#[2]	Normal
		Empty	Normal	De-assert LHOLD, assert BLAST#[3]
Direct Slave Read	Local-to-PCI	Full	Normal	De-assert LHOLD, assert BLAST#[3]
		Empty	Throttle TRDY#[4]	Normal
DMA	Local-to-PCI	Full	Normal	De-assert LHOLD, assert BLAST#
		Empty	De-assert REQ#	Normal
	PCI-to-Local	Full	De-assert REQ#	Normal
		Empty	Normal	De-assert LHOLD, assert BLAST#

1. Throttle IRDY# depends on the Direct Master PCI Read Mode bit (DMPBAM[6]).
2. Throttle TRDY# depends on the Direct Slave Write bit (PCICTL[27]).
3. De-assertion of LHOLD depends upon the Local Bus Direct Slave Release Bus Mode bit (LARBR[8]).
4. Retry Throttle TRDY# depends upon Direct Slave Read bit (PCICTL[25]).
De-assert TRDY# if the bit is equal to zero. If the bit is equal to 1, Retry.



* Note: Memory commands are in Memory space, I/O commands are in I/O space, and Configuration commands are in Configuration space.

Figure 5-3. Direct Master Access of PCI Bus

5.3 PCI MEMORY ACCESS

The Local processor can read or write to the PCI memory. The IOP 480 converts the Local Read/Write access. The Local Address Space starts from the Direct Master Local Base Address up to the range. Remap (PCI Base Address) defines the PCI starting address.

- **Writes**—IOP 480 continues to accept writes and returns READY# until the Write FIFO is full. It then holds off READY# until space becomes available in the Write FIFO. A programmable Direct Master FIFO “almost full” status output is provided (DMPAF). (Refer to Figure 5-1.)
- **Reads**—IOP 480 holds off READY# while gathering an Lword from the PCI Bus. Programmable prefetch modes are available if prefetch is enabled: prefetch, 4, 8, 16, or continuous until the Direct Master cycle ends. The Read cycle is terminated when Local BLAST# input is asserted. Unused read data is flushed from the FIFO. (Refer to Figure 5-2.)

The IOP 480 does not prefetch read data for single-cycle Direct Master reads (Local BLAST# input asserted during the first data phase). In this case, the IOP 480 reads a single PCI Lword.

For Direct Master single-cycle reads, the IOP 480 sets the same PCI Bus byte enables as the Local Bus.

For Multiple-Cycle reads, the IOP 480 reads multiple entire Lwords (all PCI Bus byte enables are set).

For Multiple-Cycle writes, the IOP 480 sets the same PCI Bus byte enables as the Local Bus.

If the Direct Master Prefetch Limit bit is enabled (DMPBAM[5]=1), the IOP 480 does not prefetch past a 4-KB boundary. Also, the Local Bus must not cross a 4-KB boundary during a Burst read.

The IOP 480 never prefetches beyond the region specified for Direct Master accesses.

5.4 PCI I/O ACCESS

When a Local Direct Master I/O access to the PCI Bus occurs, the PCI Configuration Address Register for Direct Master-to-PCI I/O Configuration Enable bit (DMCFGA[31]) determines whether an I/O or Configuration access is to be made to the PCI Bus.

Local Burst accesses are broken into single PCI I/O Address/Data cycles. The IOP 480 does not prefetch Read data for I/O or Configuration reads.

For Direct Master I/O or Configuration cycles, the IOP 480 asserts the same PCI Bus byte enables as set on the Local Bus.

If the Configuration Enable bit is cleared (DMCFGA[31]=0), a single I/O access is made to the PCI Bus. The Local Address, Remapped Decode Address bits, and Local byte enables are encoded to provide the address and are output with an I/O Read or Write command during a PCI Address cycle.

When the I/O Remap Select bit is set (DMPBAM[15]=1), the PCI Address bits [31:16] are forced to 0 for the 64K I/O address limit.

For writes, data is loaded into the Write FIFO and READY# is returned to the Local Bus. For reads, the IOP 480 holds off READY# while receiving an Lword from the PCI Bus.

5.5 PCI CONFIGURATION ACCESS

When a Local Direct Master I/O access to the PCI Bus occurs, the PCI Configuration Address Register for Direct Master-to-PCI I/O Configuration Enable bit (DMCFGA[31]) determines whether an I/O or Configuration access is to be made to the PCI Bus.

If the Configuration Enable bit (DMCFGA[31]) is set, a Configuration access is made to the PCI Bus. In addition to enabling configuration of this bit, the user must provide all register information. The Register Number bits and Device Number bits (DMCFGA[7:2, 15:11], respectively) must be modified and a new Configuration Read/Write cycle must be performed before accessing other registers or devices.

If the PCI Configuration Address register selects a Type 0 command, bits [10:0] of the register are copied to address bits [10:0]. Bits [15:11] (device number) are translated into a single bit being set in the PCI Address bits [31:11]. The PCI address bits [31:11] can be used as a device select. For a Type 1 command, bits [23:0] are copied from the register to bits [23:0] of the PCI address. The PCI Address bits [31:24] are 0. A Configuration Read or Write command code is output with the address during the PCI Address cycle (refer to the DMCFGA register).

For writes, Local data is loaded into the Write FIFO and READY# is returned. For reads, the IOP 480 holds off READY# while gathering an Lword from the PCI Bus.

5.5.1 Configuration Cycle Example

To perform a Type 0 configuration cycle to PCI device on AD[21]:

1. The IOP 480 must be configured to allow Direct Master access to the PCI Bus. The IOP 480 must also be set to respond to I/O Space accesses. These bits must be set (PCICR[2:0]=111b).

   In addition, Direct Master Memory and I/O access must be enabled (DMPBAM[1:0]=11b).

2. The local memory map selects the Direct Master range. For this example, use a range of 1 MB:

   1 MB=2^{20}=000FFFFFh

   The value to program into the Range register is the inverse of 000FFFFFh (FFF00000h):

   DMRR=FFF00000h

3. The local memory map determines the Local Base address for the Direct Master-to-PCI I/O Configuration register. For this example, use 40000000h:

   DMLBAI=40000000h

4. The PCI Address (Remap) for Direct Master-to-PCI Memory register must enable the Direct Master I/O access. The Direct Master I/O Access Enable bit must be set (DMPBAM[1]=1).

5. The user must know which PCI device and PCI Configuration register the PCI Configuration cycle is accessing. This example assumes the IDSEL signal of the target PCI device is connected to AD[21] (logical device #10=0Ah). Also access PCIBAR0 (the fourth register, counting from 0). Set DMCFGA[31, 23:0) as follows:

Bit	Description	Value
1:0	Configuration Type 0.	00b
7:2	Register Number. Fourth register. Must program a “4” into this value, beginning with bit 2.	000100b
10:8	Function Number.	000b
15:11	Device Number n-11, where n is the value in AD[n]=21-11=10.	01010b
23:16	Bus Number.	00000000b
31	Configuration Enable.	1

   After these registers are configured, a simple Local Master Memory cycle to the I/O Base address is necessary to generate a PCI Configuration Read or Write cycle. Offset to the Base address is not necessary because the register offset for the read or write is specified in the Configuration register. The IOP 480 takes the Local Bus Master Memory cycle and checks for the Configuration Enable bit (DMCFGA[31]). If set, the IOP 480 converts the current cycle to a PCI Configuration cycle, using the DMCFGA register and the Write/Read signal (LWR#).

6. The Register Number bits and Device Number bits (DMCFGA[7:2, 15:11], respectively) must be modified and a new Configuration Read/Write cycle must be performed before accessing other registers or devices.

5.6 PCI DUAL ADDRESS CYCLE

The IOP 480 supports PCI Dual Address Cycle (DAC) when it is a PCI Bus Master using the DMDAC register for Direct Master transactions. The DAC command is used to transfer a 64-bit address to devices that support 64-bit addressing when the address is not in the low 4 GB address space. The IOP 480 performs a DAC within two PCI clock periods, where the first PCI address is a Lo-Addr with the command (C/BE[3:0]#) “D” and the second PCI address is a Hi-Addr with the command (C/BE[3:0]#) “6” or “7”, depending upon it being a PCI Read or a PCI Write cycle. When the DMDAC register contains a value of 0h, the IOP 480 performs a Single Address Cycle (SAC) on the PCI Bus.

5.7 TARGET ABORT

The IOP 480 Master/Target abort logic enables a Local Bus Master to perform a Direct Master bus poll of devices to determine whether devices exist (typically when the Local Bus performs configuration cycles to the PCI Bus). When a PCI Master device attempts to access and does not receive DEVSEL# within six PCI clocks, it results in a Master Abort. The Local Bus Master must clear the Received Master Abort bit or Target Abort bit (PCISR[13 or 12]=0, respectively) and continue by processing the next task.

If a PCI Master, Target Abort, or Retry Timeout is encountered during a transfer, the IOP 480 asserts INTO if enabled (LINTENB[1:0]=1). If a Local Bus Master is waiting for READY#, it is asserted along with BTERM#. The Local Master's interrupt handler can take the appropriate application-specific action. It can then clear the Target Abort bit (PCISR[12]) to de-assert the INTO interrupt and re-enable Direct Master transfers.

If a Local Bus Master is attempting a Burst read from a nonresponding PCI device (Master/Target abort), it receives READY# and BTERM# for the first cycle only. In addition, the IOP 480 asserts INTO if the Enable Local Bus INTO bits are enabled (LINTENB[1:0]). If the Local processor cannot terminate its Burst cycle, it may cause the Local processor to hang. In this case, look at LOCCTL[25].

5.8 MEMORY WRITE AND INVALIDATE

The IOP 480 can be programmed to perform Memory Write and Invalidate cycles to the PCI Bus for Direct Master transfers, as well as for DMA transfers. The IOP 480 supports Memory Write and Invalidate transfers for cache line sizes of 8 or 16 Lwords. Size is specified in the System Cache Line Size bits (PCICLSR[7:0]). If a size other than 8 or 16 is specified, the IOP 480 performs Write transfers rather than Memory Write and Invalidate transfers.

Direct Master Memory Write and Invalidate transfers are enabled when Invalidate Enable (DMPBAM[7]) and Memory Write and Invalidate Enable (PCICR[4]) are set.

In Memory Write and Invalidate mode, if the start address of the Direct Master transfer is on a cache line boundary, the IOP 480 waits until the number of Lwords required for the specified cache line size are written from the Local Bus before starting a PCI Memory Write and Invalidate access. This ensures a complete cache line write can complete in one PCI Bus ownership.

If the start address is not on a cache line boundary, the IOP 480 starts a normal PCI Write access (PCI command 7h). The IOP 480 terminates a cycle at a cache line boundary if it is performing a normal write or a Memory Write and Invalidate cycle and another cache line of data is not available. If an entire cache line is available by the time IOP 480 regains use of the PCI Bus, the IOP 480 resumes Memory Write and Invalidate cycles; otherwise, it continues with a normal write. If a target disconnects before a cache line is completed, the IOP 480 completes the remainder of that cache line, using normal writes.

5.9 DEADLOCK CONDITIONS

Deadlock can occur when a PCI Bus Master must access the IOP 480 Local Bus at the same time a Master on the IOP 480 Local Bus must access the PCI Bus.

There are two types of deadlock:

- **Partial Deadlock**—A Local Bus Master is performing a Direct Bus Master access to a PCI Bus device other than the PCI Bus device concurrently trying to access the Local Bus
- **Full Deadlock**—A Local Bus Master is performing a Direct Bus Master access to the same PCI Bus device concurrently trying to access the Local Bus

This applies only to Direct Master and Direct Slave accesses through the IOP 480. Deadlock does not occur in transfers through the IOP 480 DMA channels or the IOP 480 internal registers (such as mailboxes).

For partial deadlock, the PCI access to the Local Bus times out (the Direct Slave Retry Delay Clock (LBRD0[31:28]), which is programmable through the Local Bus Region Descriptor register) and the IOP 480 responds with a PCI Retry. The PCI Specification requires that a PCI Master release its request for the PCI Bus (de-assert REQ#) for a minimum of two PCI clocks after receiving a Retry. This allows the PCI Bus arbiter to grant the PCI Bus to the IOP 480 so that it can complete its Direct Master access and free up the Local Bus. Possible solutions are described below for cases in which the PCI Bus arbiter does not function as described (PCI Bus architecture dependent), waiting for a time out is undesirable, or a full deadlock condition exists.

For full deadlock, the only solution is to backoff the Local Bus Master.

5.9.1 Backoff

The IOP 480 BOFF# signal indicates whether a possible deadlock condition exists. The IOP 480 starts the Backoff Timer (programmable through registers) when it detects the following conditions:

- A PCI Bus Master is attempting to access memory or an I/O device on the Local Bus and is not gaining access (*for example*, LHOLDA is not received).
- A Local Bus Master is performing a Direct Bus Master Read access to the PCI Bus. Or, a Local Bus Master is performing a Direct Bus Master Write access to the PCI Bus and the IOP 480 Direct Master Write FIFO cannot accept another Write cycle.

If the Local Bus Backoff Enable bit is enabled (EROMBA[4]=1) and expires and the IOP 480 has not received LHOLDA, the IOP 480 asserts BOFF#. External bus logic can use this signal to perform backoff.

The Backoff cycle is device/bus architecture dependent. External logic (an arbiter) can assert the necessary signals necessary to cause a Local Bus Master to release a Local Bus (backoff). After the Local Bus Master backs off, it can grant the bus to the IOP 480 by asserting LHOLDA.

Once BOFF# is asserted, READY# for the current Data cycle is never asserted (the Local Bus Master must perform backoff). When the IOP 480 detects LHOLDA, it proceeds with the PCI Master-to-Local Bus access. When this access completes and the IOP 480 releases the Local Bus, external logic can release the backoff and the Local Bus Master can resume the cycle interrupted by the Backoff cycle. The IOP 480 Write FIFO retains all data it acknowledged (*that is*, the last data for which READY# was asserted).

After the backoff condition ends, the Local Bus Master restarts the last cycle with ADS#. For writes, data following ADS# should be the data the IOP 480 did not acknowledge prior to the Backoff cycle (*for example*, the last data for which READY# is not asserted).

If a PCI Read cycle completes when the Local Bus is backed off, the Local Bus Master receives that data if the Local Master restarts the same last cycle (data is not read twice). A new read is performed if the resumed Local Bus cycle is not the same as the Backed Off cycle.

5.9.2 Software/Hardware Solution for Systems without Backoff Capability

For adapters that do not support backoff, a possible deadlock solution is as follows.

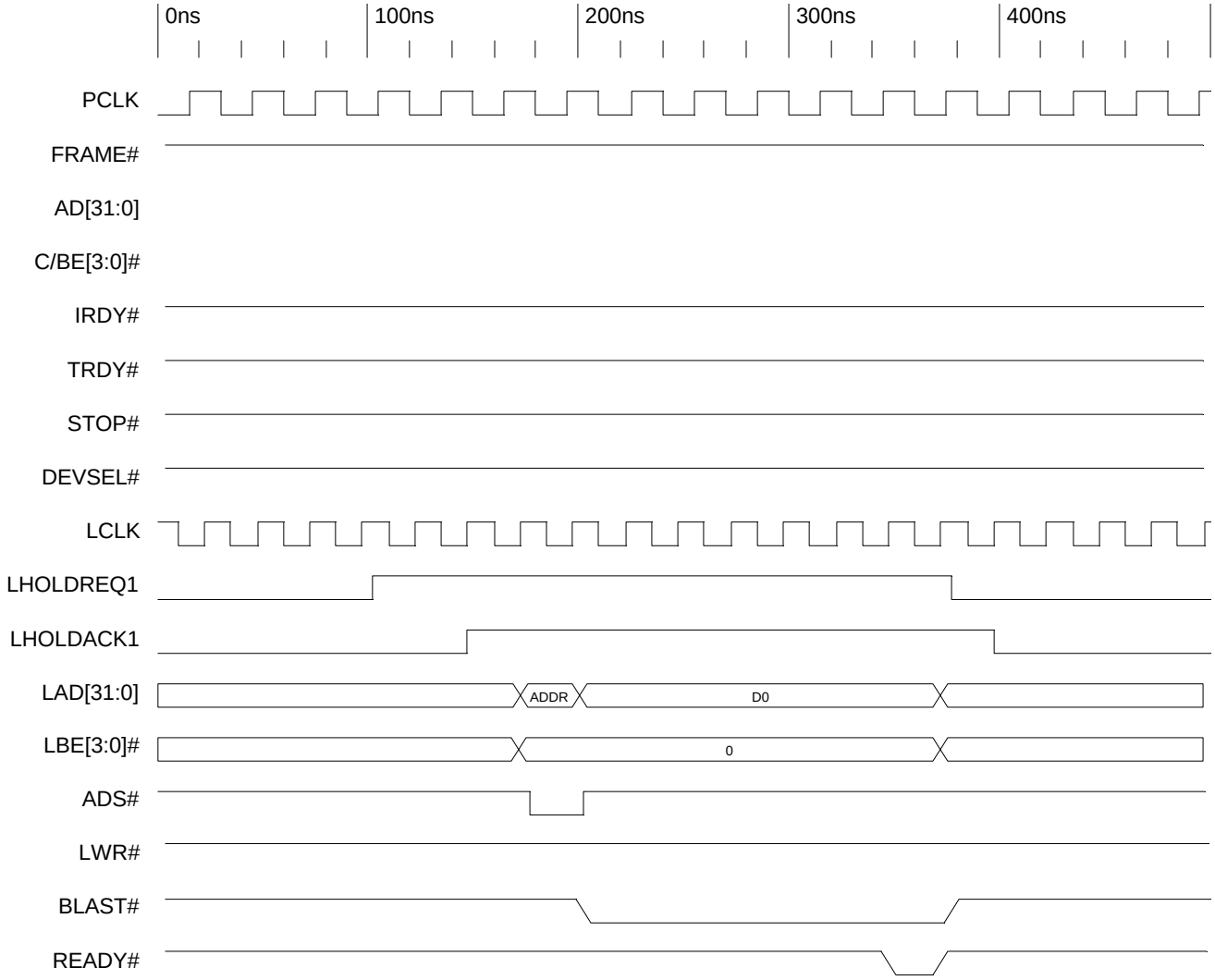
PCI Host software, external Local Bus hardware, general purpose output USER0/USER1 and general purpose input USER3/USER4 can be used by the PCI Host software to prevent deadlock. USER0/ USER1 can be asserted to request that the external arbiter not grant the bus to any Local Bus Master except the IOP 480. Status output from the Local arbiter can be connected to the general purpose input USER3/USER4 to indicate that no Local Bus Master owns the Local Bus, or the PCI Host to determine that no Local Bus Master that currently owns the Local Bus can read input. The PCI Host can then perform Direct Slave access. When the host finishes, it de-asserts USER0/USER2.

5.9.3 Software Solution to Deadlock

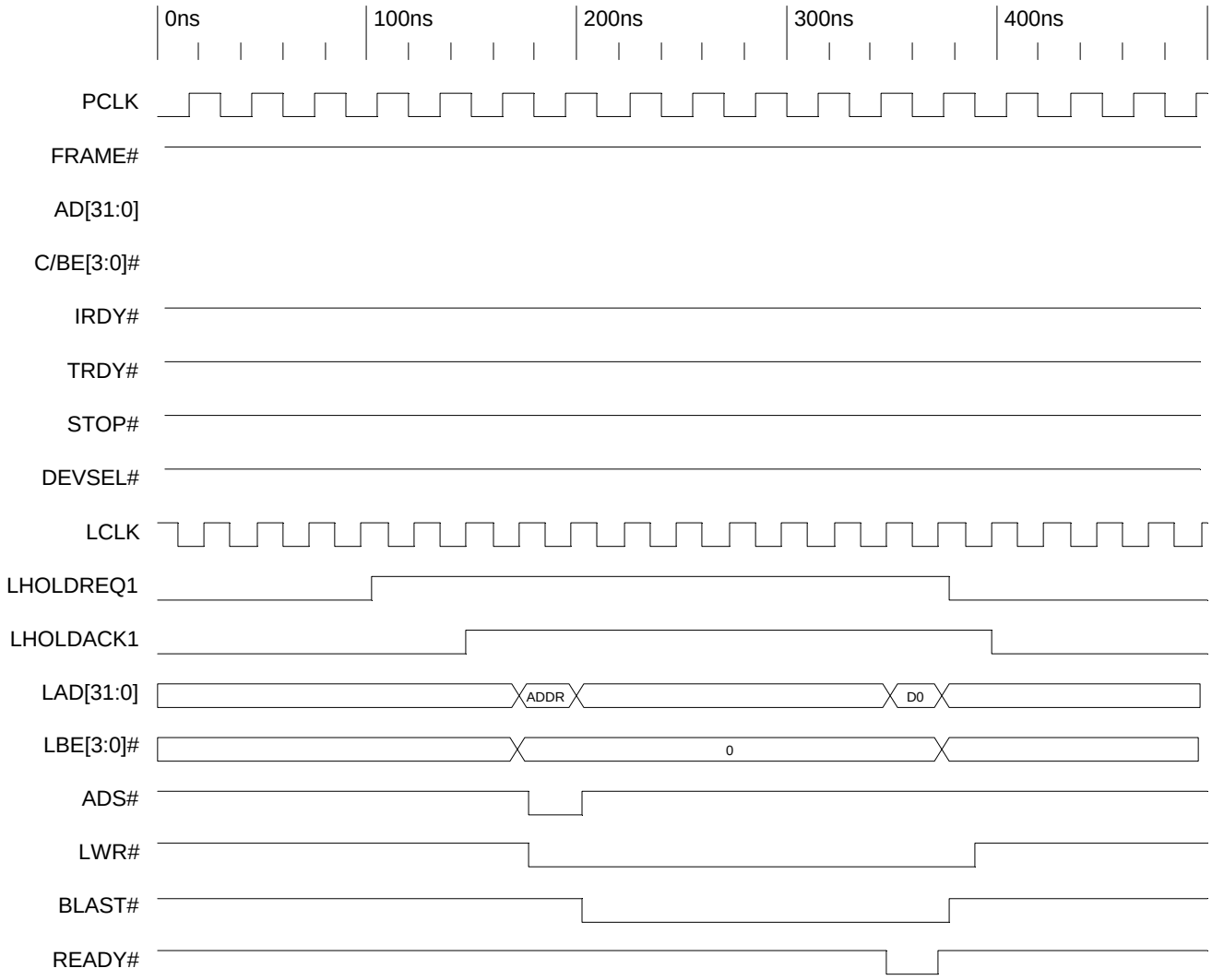
PCI Host software and Local Bus software can use a combination of Mailbox registers, Doorbell registers, interrupts, direct Local-to-PCI accesses and direct PCI-to-Local accesses to avoid deadlock.

5.10 TIMING DIAGRAMS

5.10.1 Direct Master Configuration Cycle

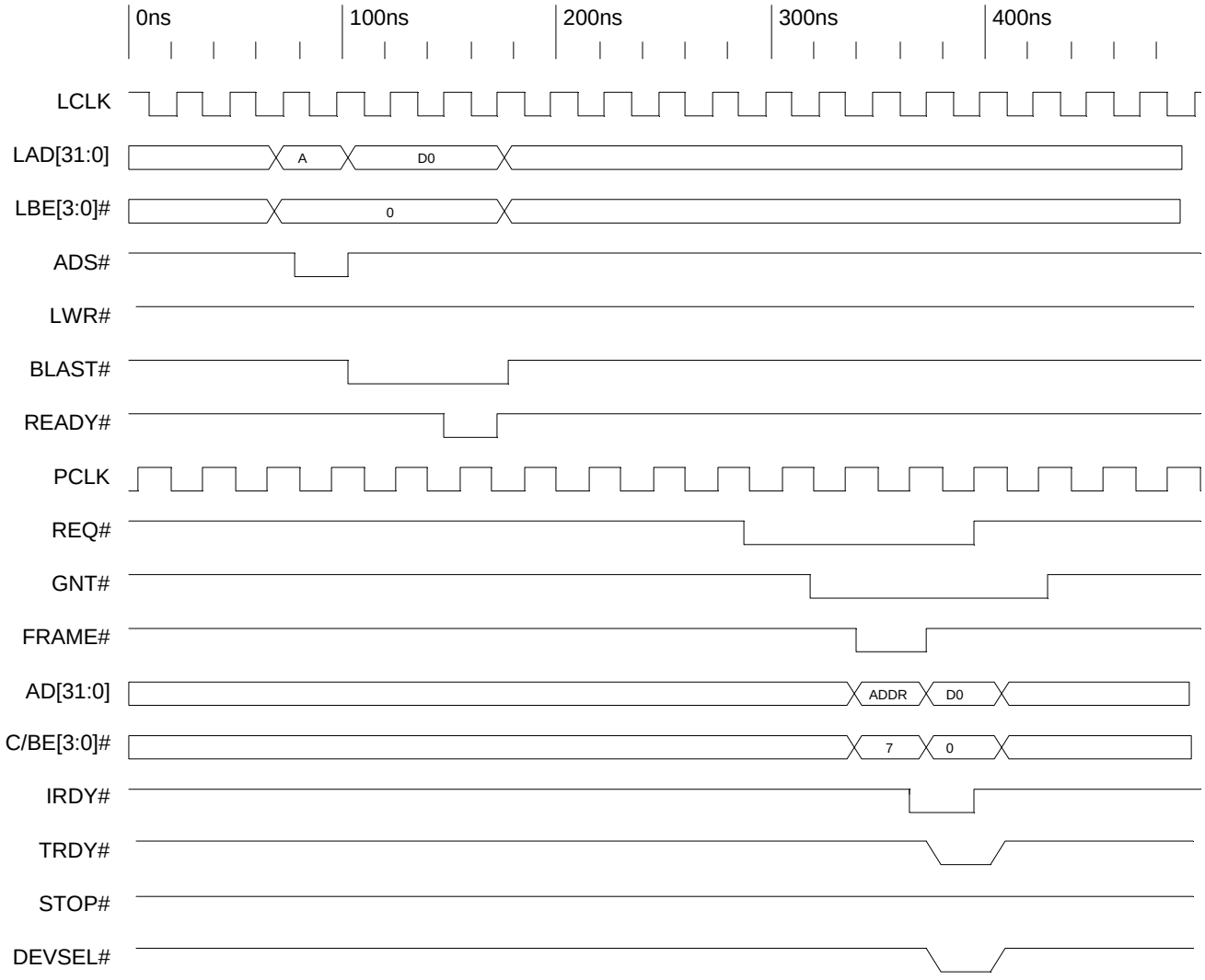


Timing Diagram 5-1. Local Master Configuration Write

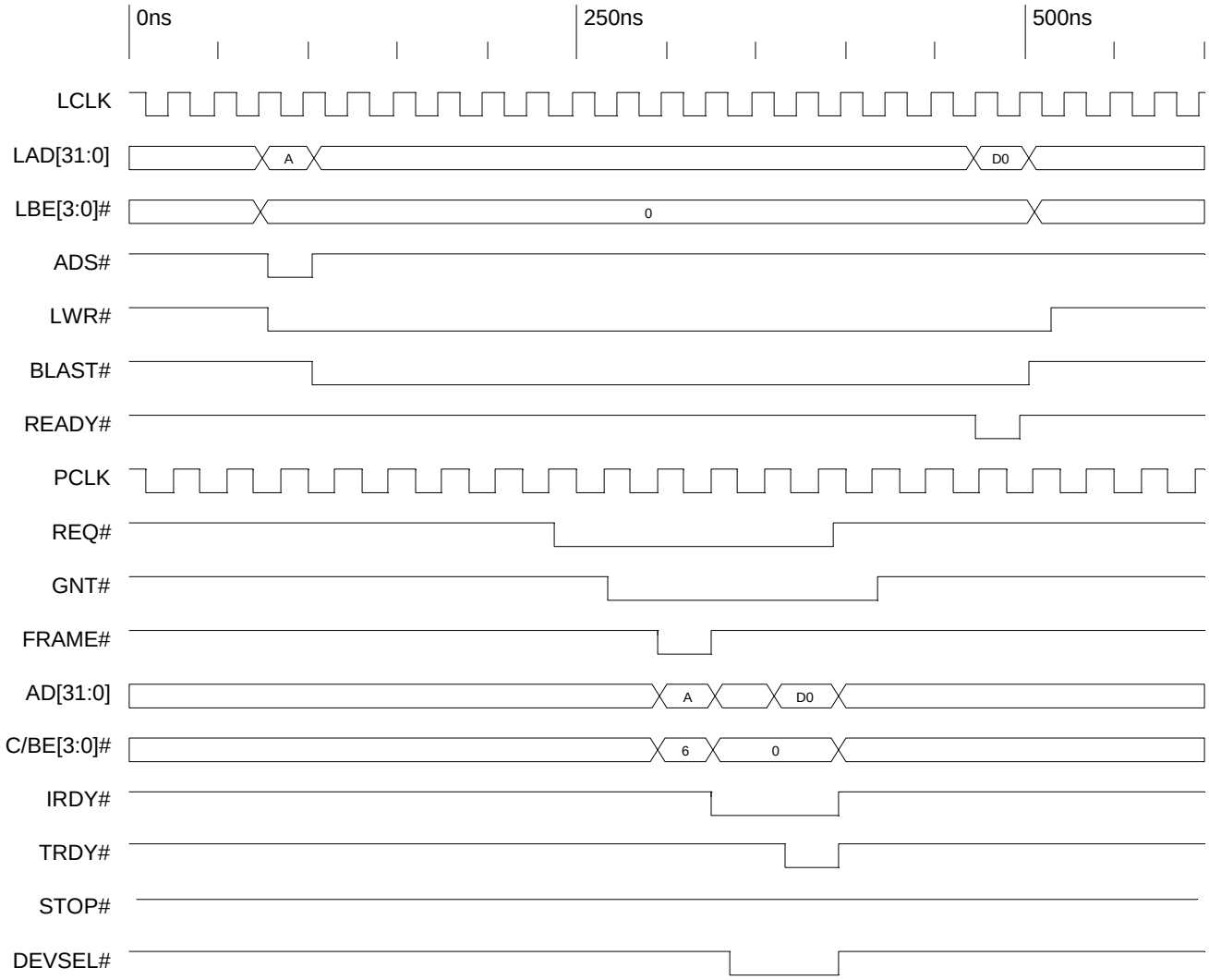


Timing Diagram 5-2. Local Master Configuration Read

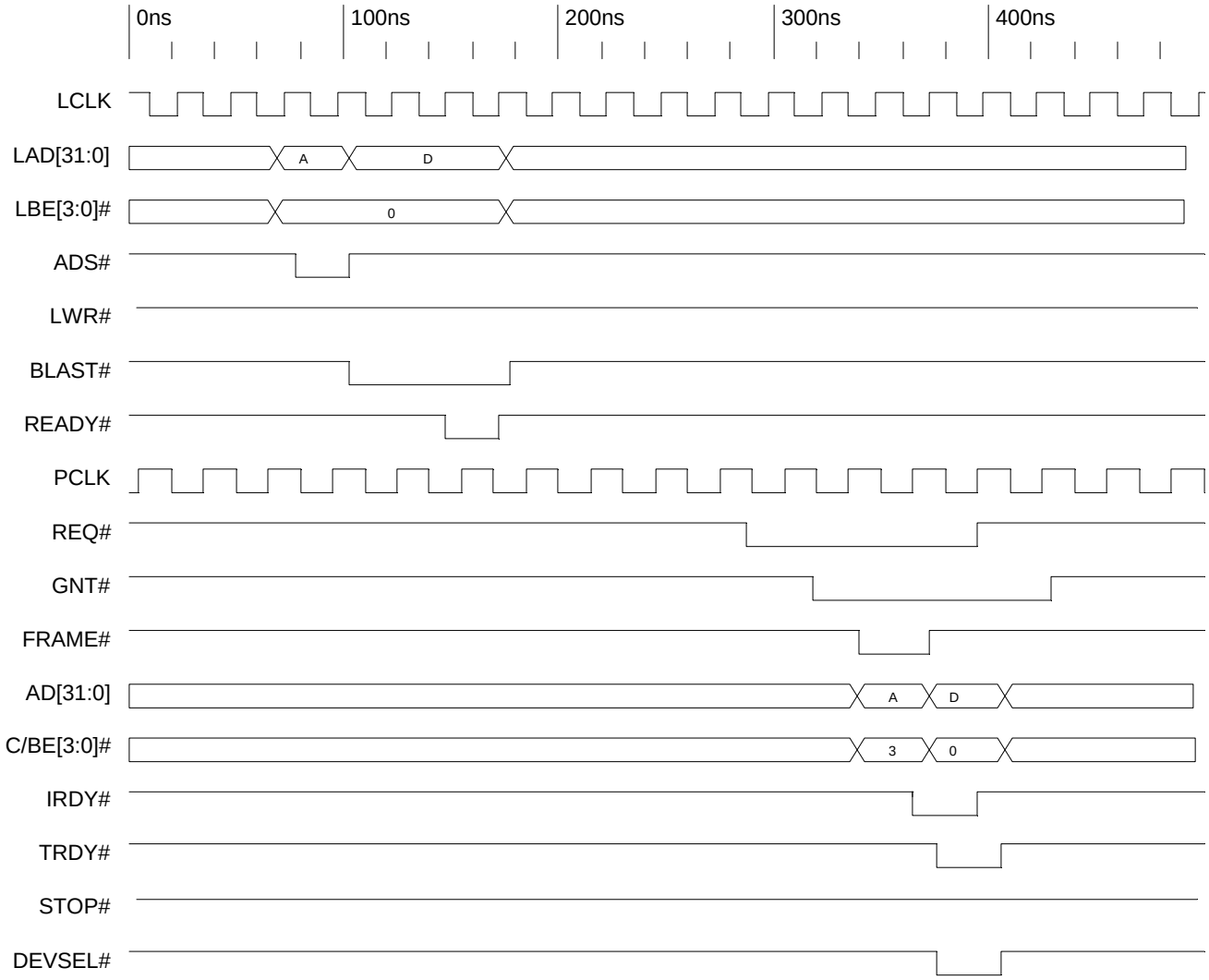
5.10.2 Direct Master Operation



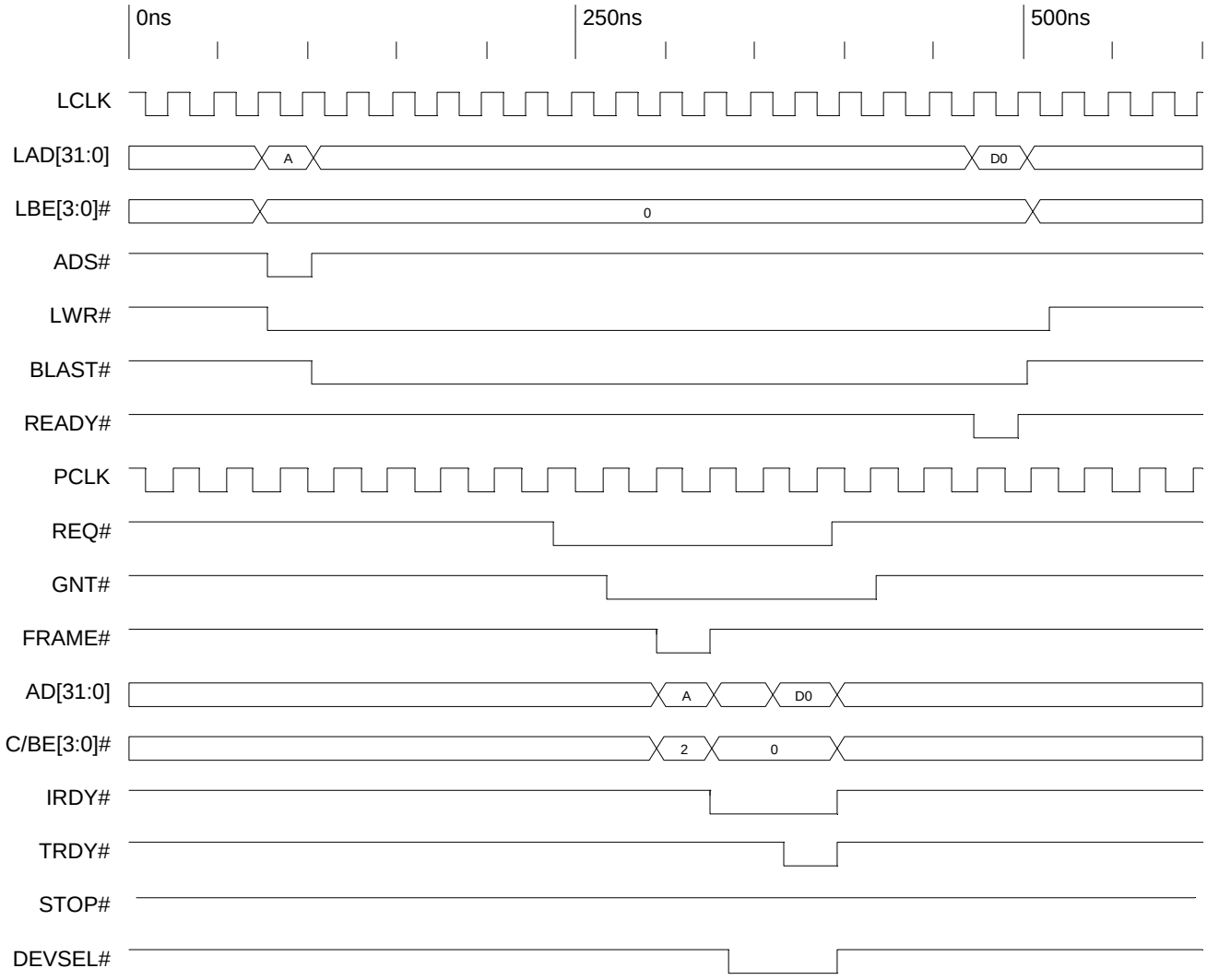
Timing Diagram 5-3. Direct Master Single Memory Write



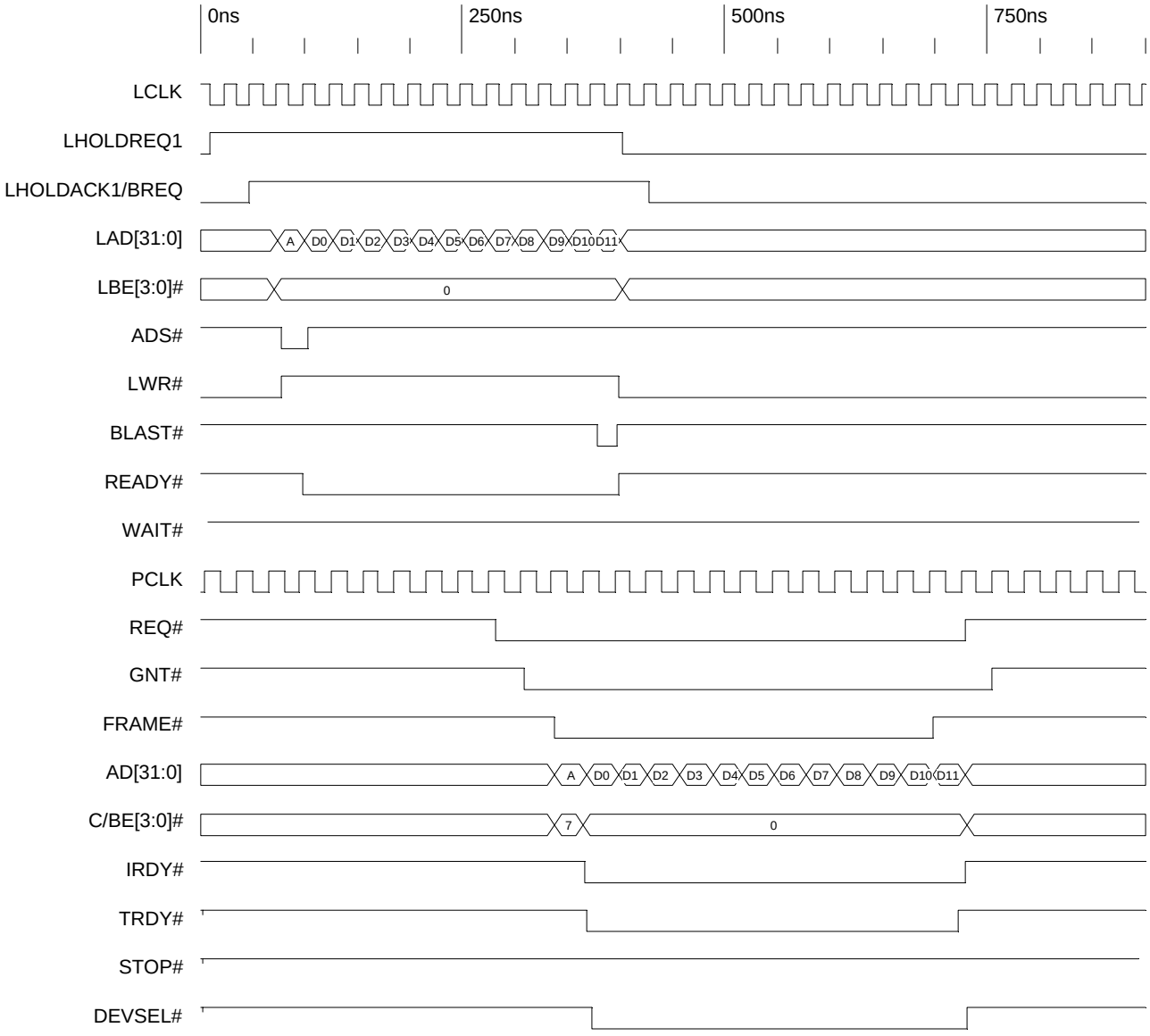
Timing Diagram 5-4. Direct Master Single Memory Read



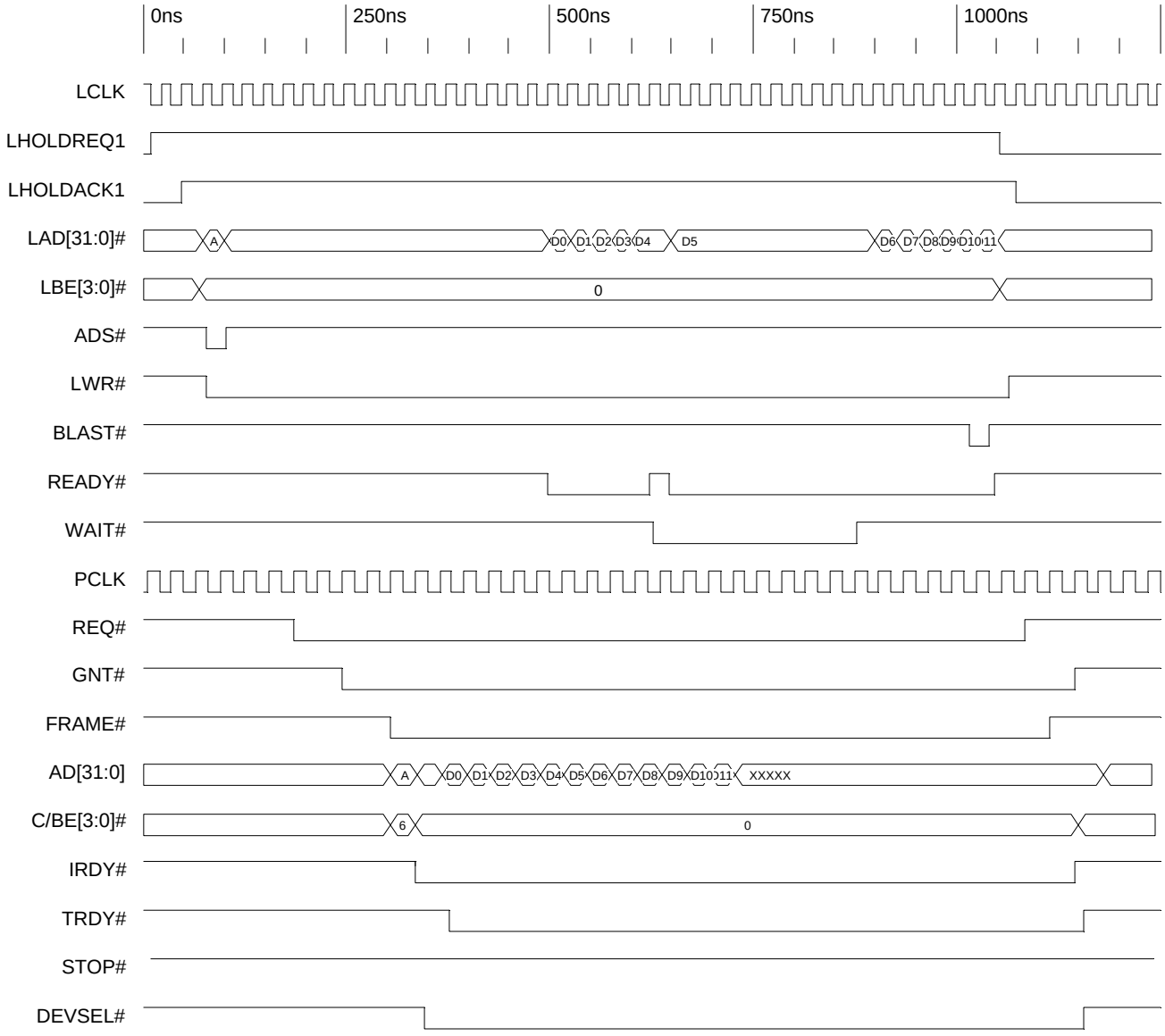
Timing Diagram 5-5. Direct Master I/O Write



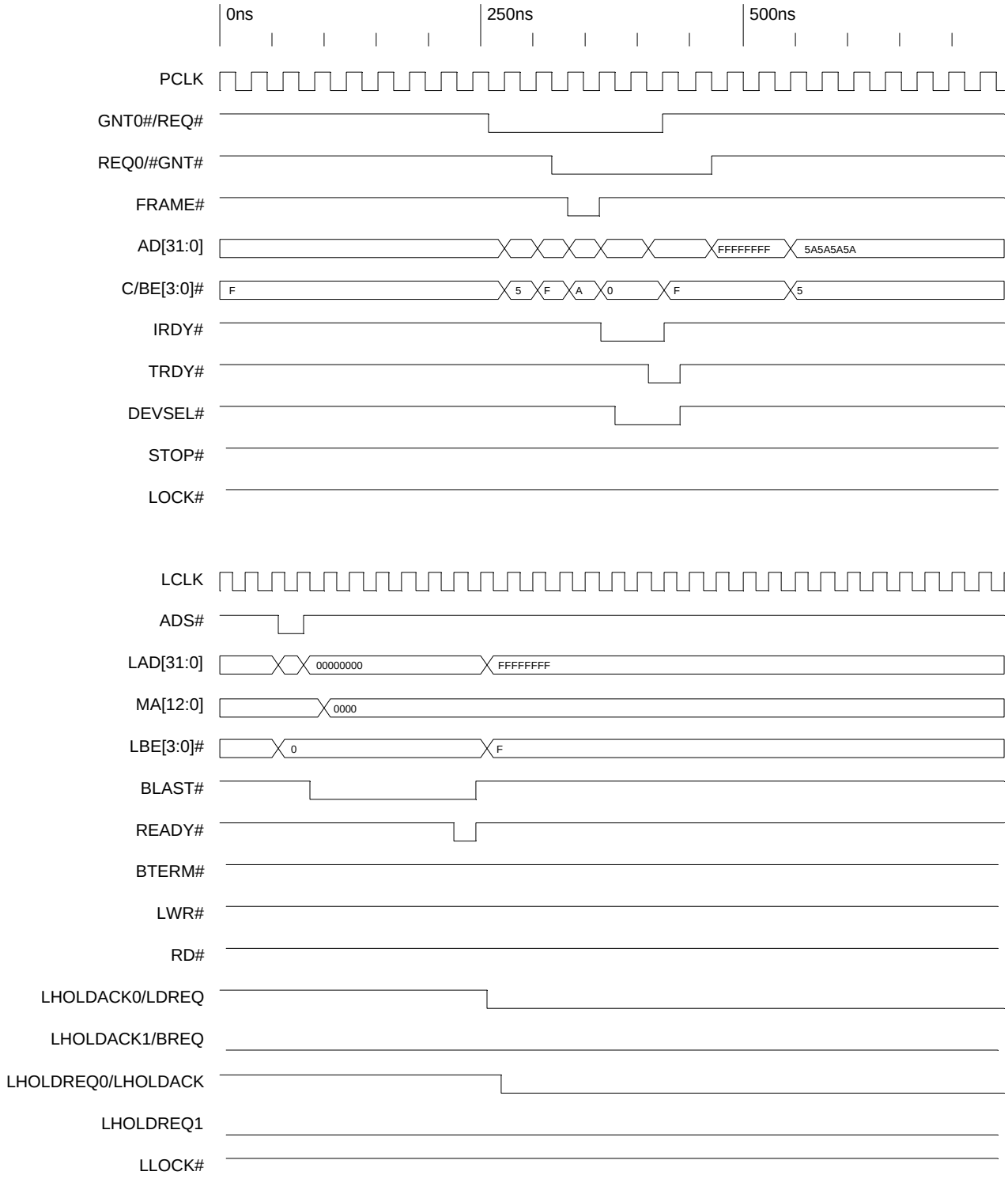
Timing Diagram 5-6. Direct Master I/O Read



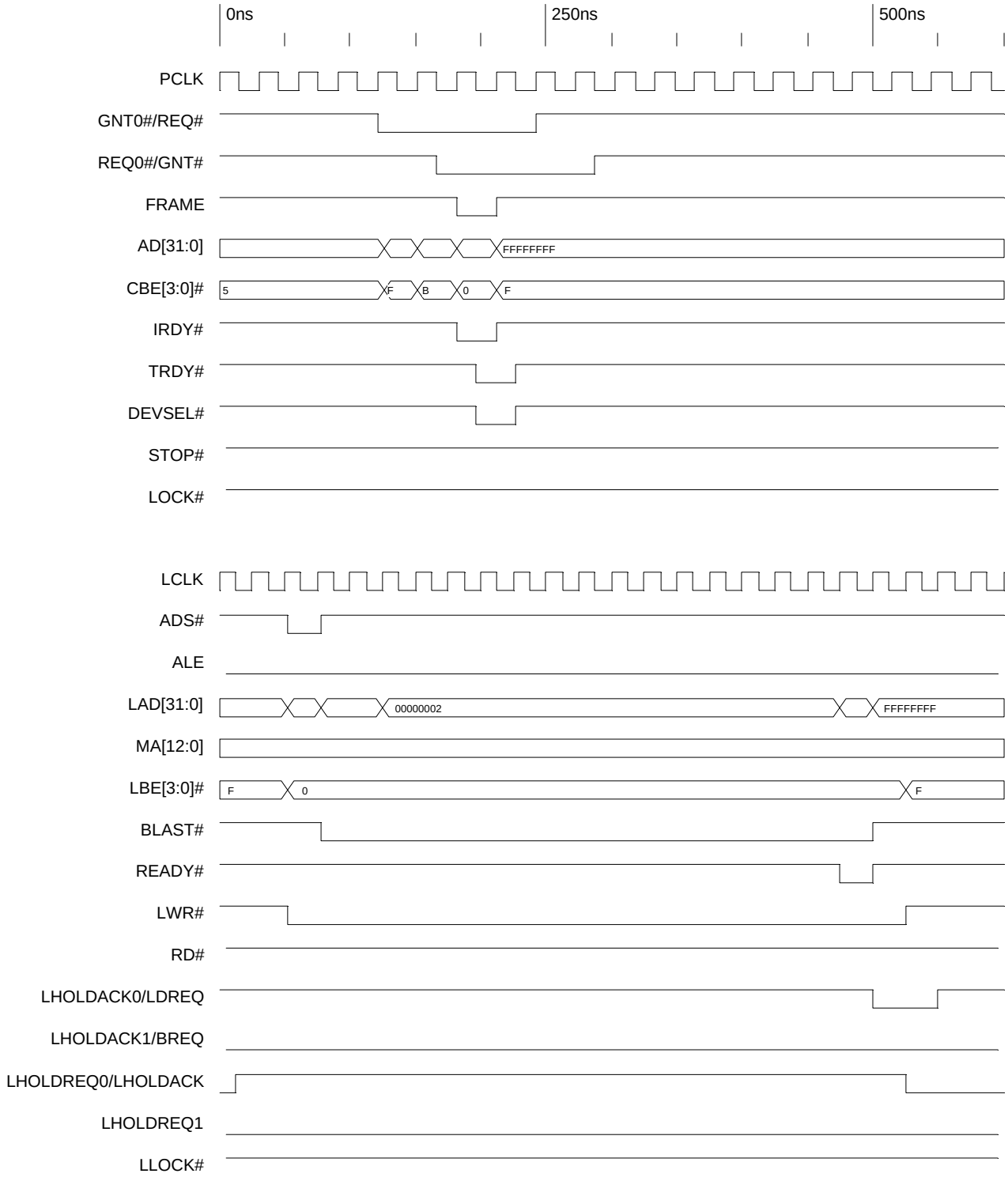
Timing Diagram 5-7. Direct Master Burst Write of 12 Lwords



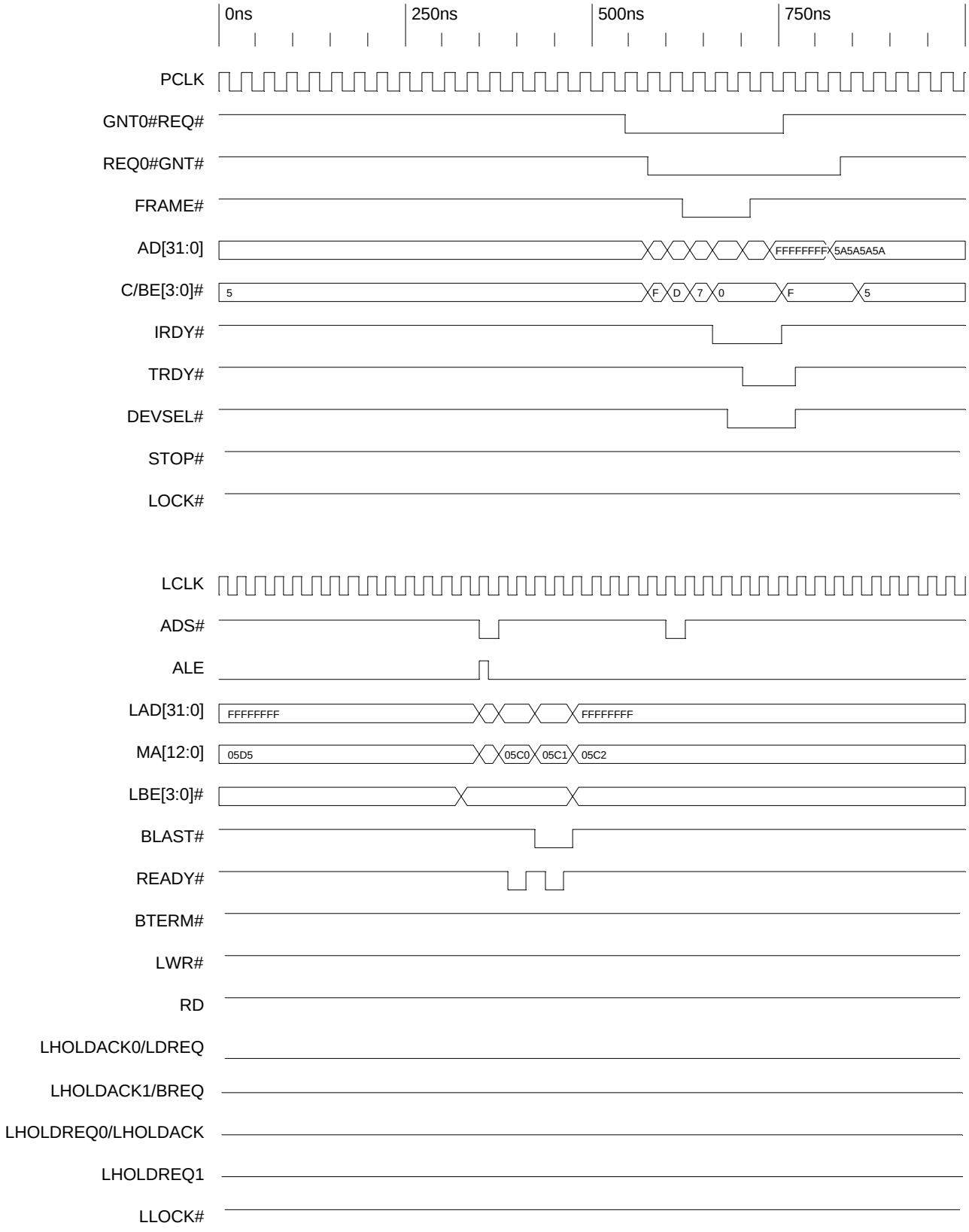
Timing Diagram 5-8. Direct Master Burst Read of 12 Lwords with WAIT#



Timing Diagram 5-9. Direct Master Configuration Write Type 0



Timing Diagram 5-10. Direct Master Configuration Read Type 1



Timing Diagram 5-11. Direct Master PCI Dual Address Cycle

6 IOP 480 CPU BUS INTERFACE

6.1 OVERVIEW

The IOP 480 CPU can access both the Local and PCI Buses, as well as the Serial Port Unit (SPU). When accessing the Local Bus, it acts very similar to a Direct Slave operation accessing the Local Bus. When accessing the PCI Bus, it acts exactly as a Direct Master operation. The IOP 480 CPU is the only device that can access the SPU, and no external bus activity is generated.

6.2 INITIALIZATION

The IOP 480 CPU, after reset, reads from Local address 0xfffffffc. The default (if not programmed differently by way of the serial EEPROM) is for this to be an 8-bit-wide bus region, as described by the Local Chip Select 0 Bus Region Descriptor register (LCS0BRD). This address should be programmed with a branch instruction to branch to wherever (in Local Address space) the rest of the initialization code lies.

If the default IOP 480 register values are used, the initialization code can be located anywhere from 0xfff00000 to 0xfffffffc, which is located in the register described by [LCS0BRA].

6.3 ACCESSING THE SPU

The IOP 480 CPU is the only device that can access the internal Serial Port Unit (SPU), and this is only at the fixed address range of 0x4000000 to 0x40000020. The SPU is on an internal 8-bit-wide bus, so only byte instructions should be used for SPU access. Refer to Section 22, “Serial Port Operation,” for more information.

6.4 ACCESSING THE LOCAL BUS

The IOP 480 CPU accesses the Local Bus in much the same way as a Direct Slave operation accesses the Local Bus.

6.4.1 Internal IOP 480 CPU Burst

When the internal IOP 480 CPU is the Local Bus Master, the maximum number of bytes the IOP 480 bursts is 16.

6.4.1.1 Loads and Stores

For load and store operations, the IOP 480 CPU reads and writes a maximum of four bytes.

Word address is specified by LAD[31:2] during an Address cycle, and byte lanes involved in the transfer are specified by LBE[3:0]#.

6.4.1.2 Cache Line Fills/Flushes

For cache line fills and flushes, the IOP 480 CPU always transfers 16 bytes.

Byte lanes involved in the transfers are specified by LBE[3:0]#.

6.5 ACCESSING THE PCI BUS

The IOP 480 CPU can access the PCI Bus in exactly the same method as a Direct Master transaction. The IOP 480 CPU first acquires the Local Bus, and then hits are to the Direct Master regions, either Memory or I/O. Refer to Section 5, “Direct Master Operation,” for more information.

6.6 ALIGNMENT

The IOP 480 CPU supports aligned and unaligned Bus transfers. Alignment rules for load and stores are based on address offsets from word boundaries. Table 6-1 through Table 6-6 describe the Bus accesses resulting from aligned and unaligned loads and stores for various bus widths.

Table 6-1. IOP 480 CPU Byte Load/Store Transactions

Address Offset from Lword Boundary (in Bytes)	8-Bit Bus	16-Bit Bus	32-Bit Bus
N/A	Byte access	Byte access	Byte access

Table 6-2. Three-Byte Load/Store Transactions

Address Offset from Lword Boundary (in Bytes)	8-Bit Bus	16-Bit Bus	32-Bit Bus
+0 (aligned)	3-byte burst	Burst of one word and one byte	3-byte access
+1	3-byte burst	Burst of one byte and one word	3-byte access
+2	N/A		
+3			

Table 6-3. Single-Lword Load/Store Transactions

Address Offset from Lword Boundary (in Bytes)	8-Bit Bus	16-Bit Bus	32-Bit Bus
+0	4-byte burst	Two word bursts	Lword access
+1	3-byte burst; byte access	Burst of one byte and one word; byte access	3-byte access; byte access
+2	2-byte burst; 2-byte burst	Word access; Word access	Word access; Word access
+3	Byte access; 3-byte burst	Byte access; burst of one byte and one word	Byte access; 3-byte access

Table 6-4. Four-Lword Cache Line Fills/Flushes

Address Offset from Lword Boundary (in Bytes)	8-Bit Bus	16-Bit Bus	32-Bit Bus
+0	16-byte burst	8-word burst	4-Lword burst
+1	N/A		
+2			
+3			

Table 6-5. IOP 480 CPU Loads/Stores

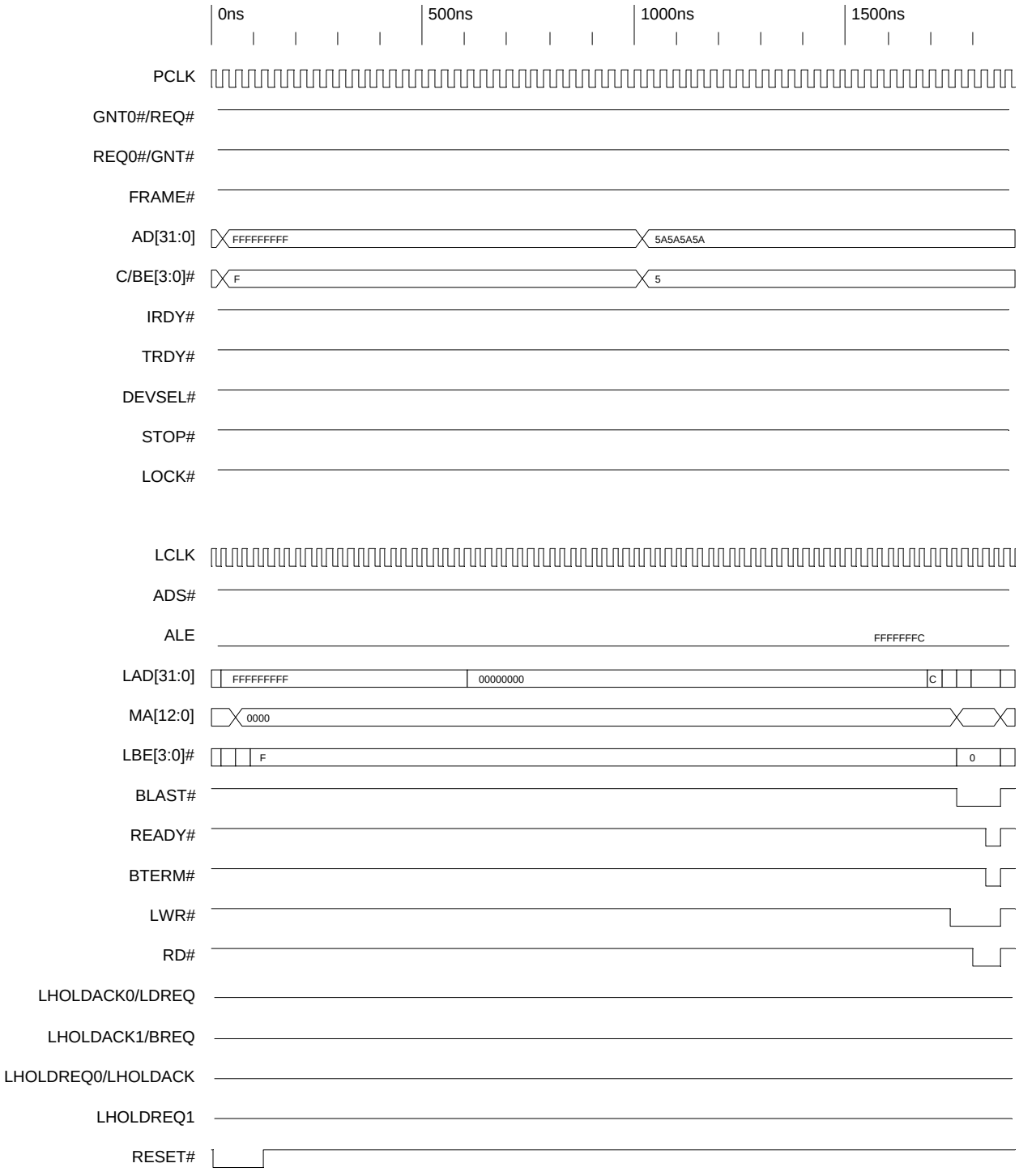
Local Bus Width	Transfers	Bytes Transferred	Starting Addresses	Address Bits Incremented during Transfer
8 bits	4 max	4 max	LBE[1:0]# (A[1:0])	LBE[1:0]# (A[1:0])
16 bits	2 max	4 max	LBE1# (A1)	LBE1# (A1)
32 bits	1 max	4 max	—	None

Table 6-6. IOP 480 CPU Cache Line Fills/Flushes

Local Bus Width	Transfers	Bytes Transferred	Starting Addresses	Address Bits Incremented during Transfer
8 bits	16	16	MA[3:2] = 00 LBE[1:0]# (A[1:0]) = 00	MA[3:2], LBE[1:0]# (A[1:0])
16 bits	8	16	MA[3:2] = 00 LBE1# (A1) = 0	MA[3:2], LBE1# (A1)
32 bits	4	16	MA[3:2] = 00	MA[3:2]

6.7 TIMING DIAGRAMS

6.7.1 IOP 480 CPU Bootup Cycle



Timing Diagram 6-1. IOP 480 CPU after Reset Start Address FFFFFFFC

7 DMA OPERATION

7.1 DMA CHANNELS 0 AND 1

7.1.1 Overview

The IOP 480 supports three independent DMA channels capable of transferring data from:

- Local-to-PCI Bus (DMA Ch 0/Ch1)
- PCI-to-Local Bus (DMA Ch 0/Ch1)
- Local-to-Local Bus (DMA Ch 2 only)

DMA Channel 0 has a programmable 32-Lword bi-directional FIFO, while DMA Channel 1 has a programmable 16-Lword bi-directional FIFO. Both channels support Block transfers, Scatter/Gather transfers, and an End of Transfer pin (EOT#). Both channels also support Demand Mode DMA transfers. For DMA transfers, the PCI Master Enable bit must be enabled in PCICR[2] before the IOP 480 can become a PCI Bus Master. In addition, both DMA Channels 0 and 1 can be programmed to:

- Operate with 8-, 16-, or 32-bit Local Bus width
- Use 0-15 internal wait states (Local Bus) data-to-data
- Enable/disable Local Bus burst capability
- Limit Local Bus bursts to four (BTERM# enable/disable)
- Hold Local address constant (Local Target is FIFO) or increment
- Perform PCI Memory Write and Invalidate (command code = Fh) or normal PCI Memory Write (command code = 7h)
- End Local transfer with/without BLAST# (DMA Fast/Slow termination)
- Assert PCI interrupt (INTA#) or Local interrupt (INTO) when DMA transfer is complete or Terminal Count is reached during Scatter/Gather DMA mode transfers
- Operate in DMA Clear Count mode
- Pause/Poll/Skip invalid descriptors
- EOT# to end a link or whole chain

The IOP 480 DMA controllers also support Dual Address PCI registers (C0PCIHADR and C1PCIHADR).

The Local Bus Latency Timer determines the number of Local clocks the IOP 480 can hold the Local Bus before relinquishing it. The Local Pause Timer starts after LHOLDA is de-asserted. It sets the number of Local Bus clocks to wait after relinquishing the bus before the DMA channel can request the Local Bus.

7.1.2 PCI Dual Address Cycle

The IOP 480 supports PCI Dual Address Cycle (DAC) for DMA. For Scatter/Gather DMA, set C0MODE[17] and/or C1MODE[17] to 1. The DAC command is used to transfer a 64-bit address to devices that support 64-bit addressing when the address is not in the low 4-GB address space. The IOP 480 performs a DAC within two PCI clock periods, where the first PCI address is a Lo-Addr with the command (C/BE[3:0]#) “D” and the second PCI address is a Hi-Addr with the command (C/BE[3:0]#) “E” or “7” (PCI Read or Write cycle). When the C0PCIHADR or C1PCIHADR register contains a value of 0h, the IOP 480 performs a Single Address Cycle (SAC) on the PCI Bus. (Refer to Figure 7-3 on page 7-3.)

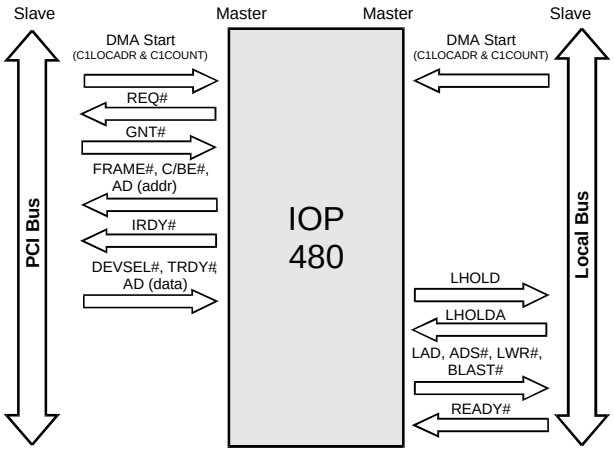


Figure 7-1. DMA, PCI-to-Local Bus

Note: *The figure represents a sequence of bus cycles.*

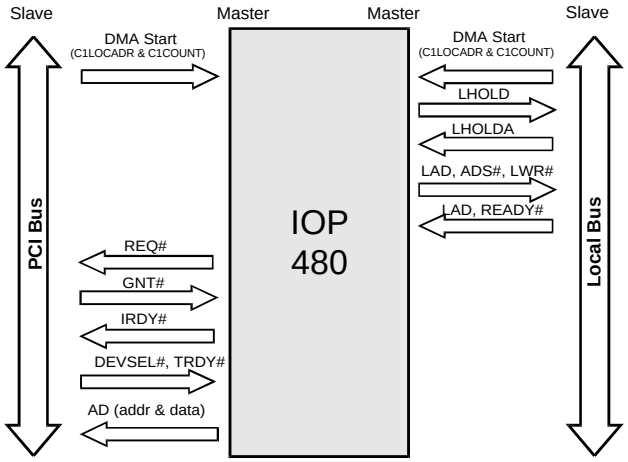


Figure 7-2. DMA, Local-to-PCI Bus

Note: *The figure represents a sequence of bus cycles.*

Table 7-1. DMA Local Burst Mode

Burst Enable Bit	BTERM# Enable Bit	Result
0	X	Single cycle
1	0	Burst up to four data cycles
1	1	Burst forever (terminate when BTERM# is asserted or transfer complete)

Note: *In Table 7-1, X = don't care, 0 = disable and 1 = enable.*

7.1.3 Block DMA Mode

To perform Block DMA mode, set (C0MODE[9]=0) for DMA Channel 0 and/or (C1MODE[9])=0 for DMA Channel 1.

The Host Processor or the Local Processor sets the Local and PCI starting Addresses, transfer byte count, and transfer direction. The Host or Local Processor then sets the start bit to initiate a transfer. The IOP 480 arbitrates for the PCI and Local Buses and transfers data. Once the transfer completes, the IOP 480 sets the Channel Done bit(s) (C0CSR[4]=1 and/or C1CSR[4]=1) and, if enabled, asserts an interrupt(s) (C0MODE[10] and/or C1MODE[10]) to the Local Processor or the PCI Host (programmable). The Channel Done bit(s) can be polled to indicate the DMA Transfer status.

DMA registers are accessible from the PCI and Local Buses (refer to Figure 7-3).

During a DMA transfer(s), the IOP 480 is a Master on both the PCI and Local Buses.

The IOP 480 releases the PCI Bus if one of the following conditions occurs:

- FIFO is full (PCI-to-Local)
- FIFO is empty (Local-to-PCI)
- Terminal count is reached
- PCI Bus Latency Timer expires (PCILTR[7:0])—normally programmed by the Host PCI BIOS—and PCI GNT# de-asserts
- PCI Host asserts STOP#
- Direct Master request is pending

The IOP 480 releases the Local Bus if one of the following occurs:

- FIFO is empty (PCI-to-Local)
- FIFO is full (Local-to-PCI)
- Terminal count is reached
- Local Bus Latency Timer is enabled and expires (LOCTMR[7:0])
- BOFF# is asserted
- Direct Slave request is pending

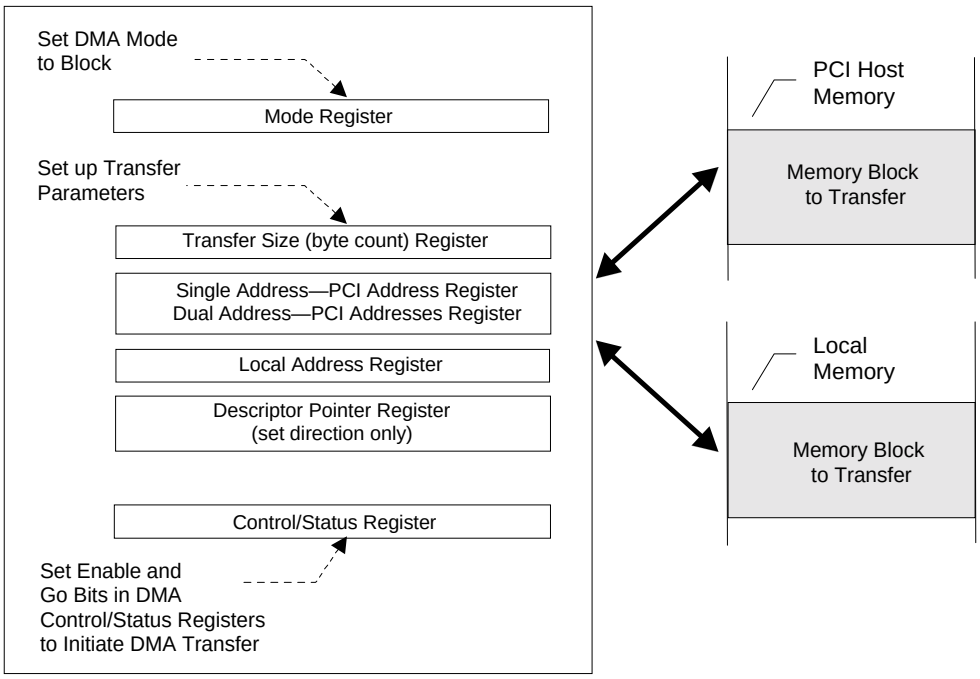


Figure 7-3. Block DMA Mode Initialization (Single Address or PCI Dual Address Cycle)

7.1.4 Scatter/Gather DMA Mode

In Scatter/Gather DMA mode, the Host Processor or Local Processor sets up descriptor blocks in Local or Host memory composed of PCI and Local addresses, transfer count, transfer direction, and address of next descriptor block (refer to Figure 7-6 on page 7-5). The Host or Local Processor then:

- Enables the Scatter/Gather Mode bit(s) (C0MODE[9]=1 and/or C1MODE[9]=1)
- Sets up the address of the initial descriptor block in the IOP 480 Descriptor Pointer register(s) (C0DESCPTR and/or C1DESCPTR)
- Initiates the transfer by setting a start bit(s) (C0CSR[1:0] and/or C1CSR[1:0])

The IOP 480 loads the first descriptor block and initiates the data transfer. The IOP 480 continues to load descriptor blocks and transfer data until it detects the End of Chain bit(s) (C0DESCPTR[1] and/or C1DESCPTR[1]) are set. (These bits are part of each descriptor.) When the End of Chain bit(s) is detected, the IOP 480 completes the current descriptor block and sets the DMA Done bit(s) (C0CSR[4] and/or C1CSR[4]). If the Done Interrupt Enable bit is set (C0MODE[10]/C1MODE[10]) to 1, the IOP 480 asserts a PCI interrupt (INTA#) and/or Local interrupt (INTO) at the end of the Scatter/Gather and interrupt enabled.

The IOP 480 can also be programmed to assert PCI or Local interrupts after each descriptor block transfer is complete.

The DMA controller can be programmed to clear the transfer size at completion of each DMA, using the DMA Clear Count Mode bit(s) (C0MODE[16] and/or C1MODE[16]).

Notes: *In Scatter/Gather DMA mode, the descriptor includes the PCI and Local Address space, transfer size, and next descriptor pointer. The Descriptor Pointer register(s) (C0DESCPTR and/or C1DESCPTR) contains end of chain (bit 1), direction of transfer (bit 3), next descriptor address (bits [31:4]), interrupt after terminal count (bit 2), and descriptor location (bit 0) bits.*

The Local Bus width must be the same as Local memory bus width.

A DMA descriptor can be in either Local or PCI memory (for example, first descriptor in Local memory, and second descriptor in PCI memory).

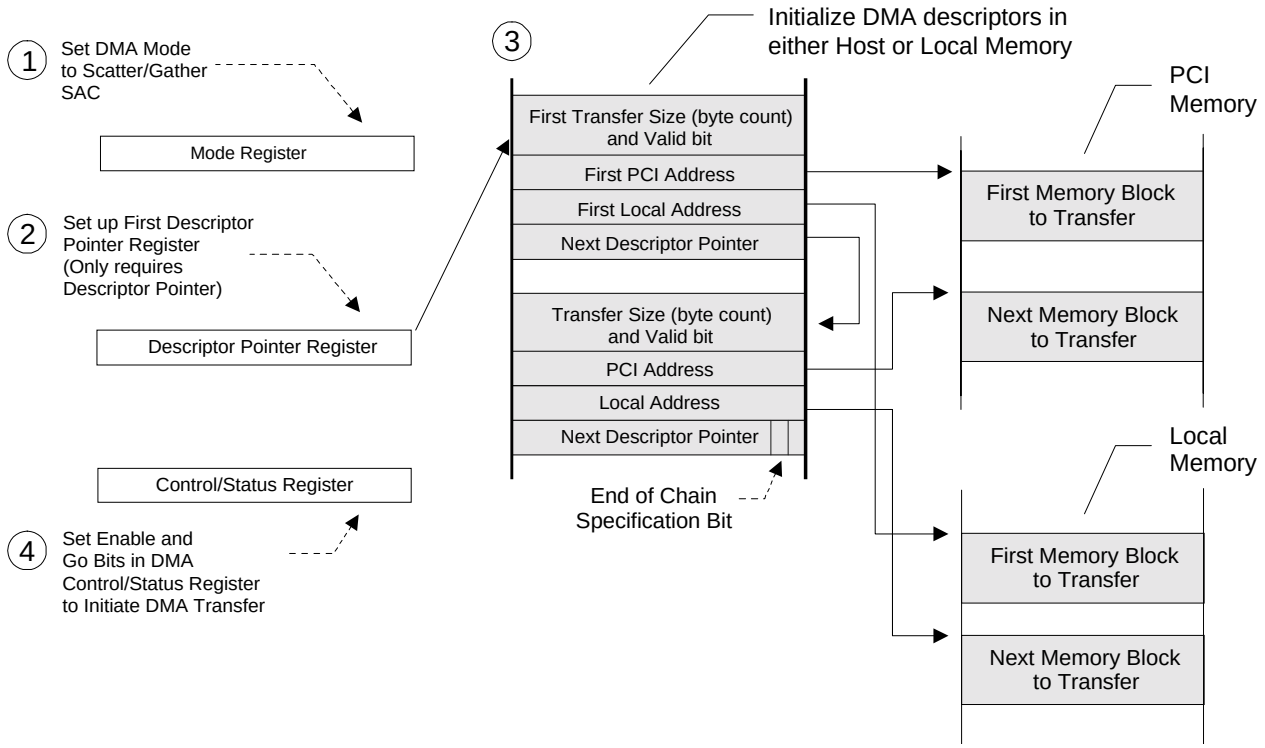


Figure 7-4. Scatter/Gather DMA Mode Initialization (Single Address Cycle)

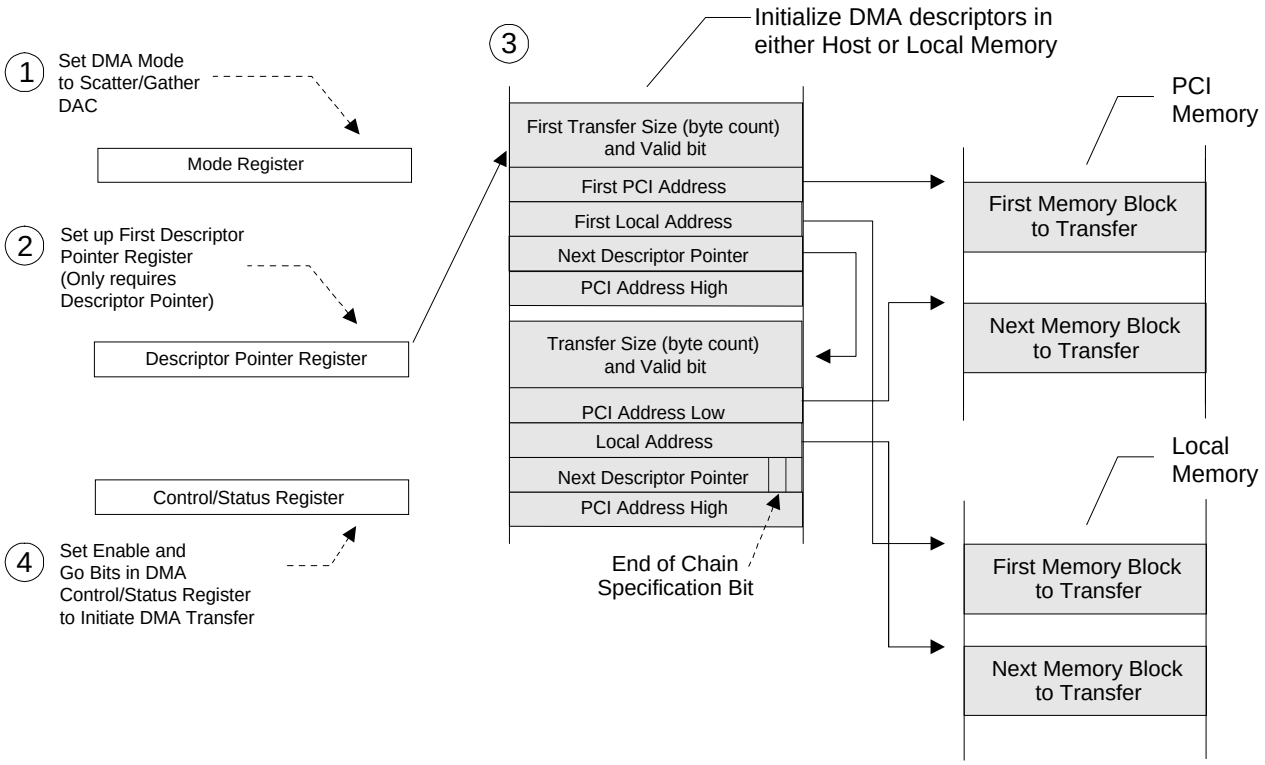


Figure 7-5. Scatter/Gather DMA Mode Initialization (PCI Dual Address Cycle)

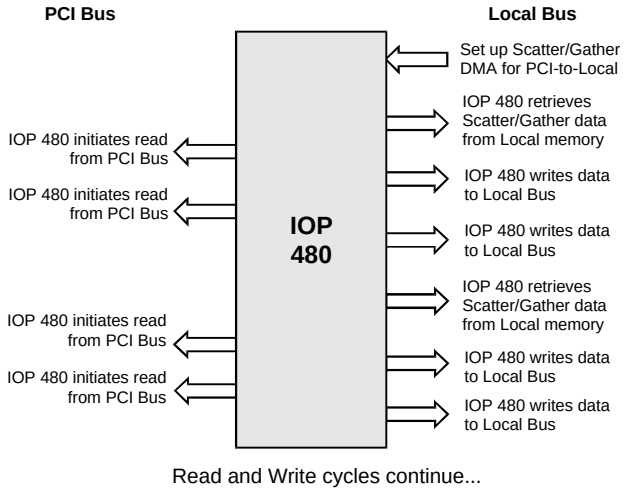


Figure 7-6. Scatter/Gather DMA Mode from PCI-to-Local Bus (Arbitration from Local Bus)

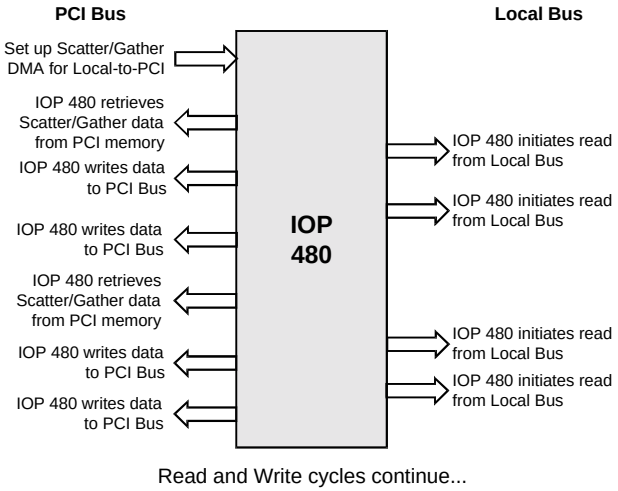


Figure 7-7. Scatter/Gather DMA Mode from Local-to-PCI Bus (Arbitration from PCI Bus)

Note: *The figures represent a sequence of bus cycles.*

7.1.5 Local-to-PCI Transfer

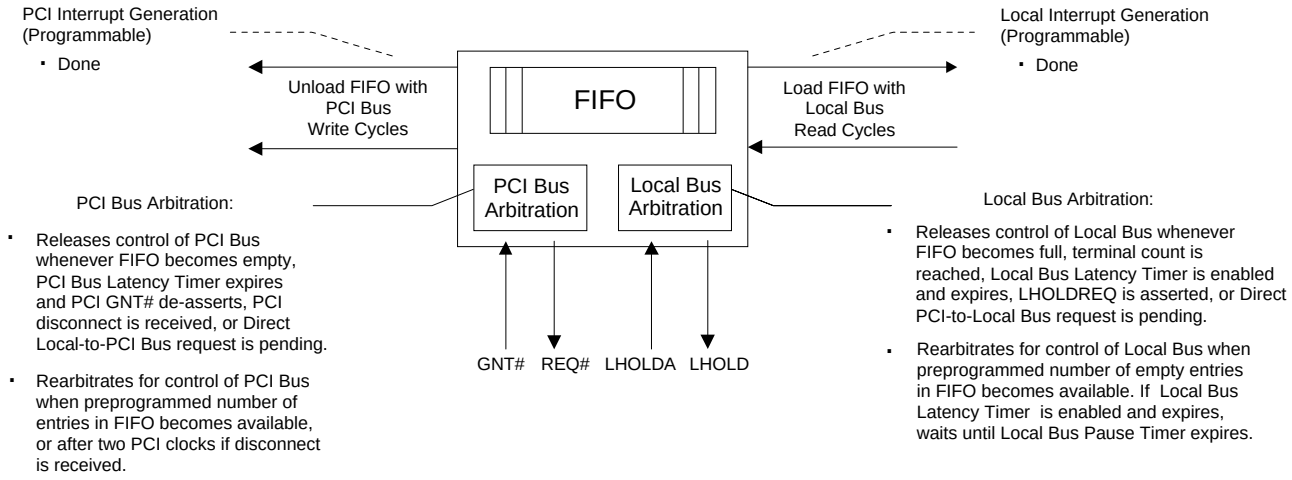


Figure 7-8. Local-to-PCI Bus DMA Data Transfer Operation

7.1.6 PCI-to-Local Transfer

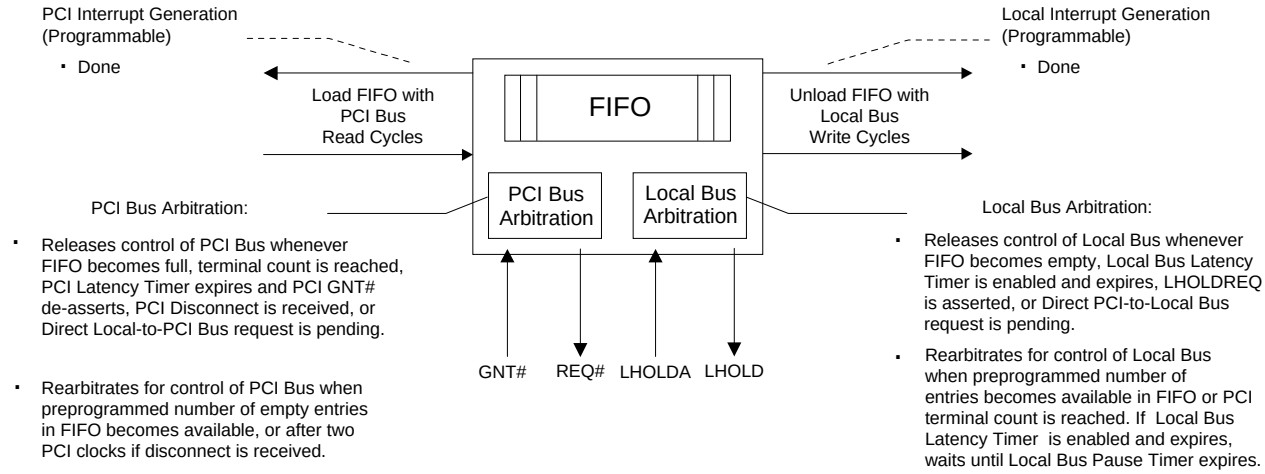


Figure 7-9. PCI-to-Local Bus DMA Data Transfer Operation

7.1.7 Demand Mode

A bit in the DMA Configuration register specifies that the channel operates in Demand mode. In Demand mode, the user sets up the Configuration registers of the DMA controller and initiates a transfer. The DMA controller transfers data when the DREQ[1:0]# input of the DMA channel is asserted. The DMA controller then asserts DACK[1:0]# to indicate that the current Local Bus transfer is in response to the DREQ[1:0]# input.

The DMA controller continues to transfer data until it reaches the transfer count or until DREQ[1:0]# is de-asserted. The minimum transfer size per DREQ[1:0]# input is one Lword (32 bits). This may result in multiple transfers for an 8- or 16-bit bus. The DREQ [1:0]# input can be thought of as a DMA Pause button.

7.1.8 Demand Mode/Fast Terminate

The DMA Fast/Slow Terminate Mode Select bit(s) (C0MODE[15] and/or C1MODE[15]) determines whether BLAST# is asserted after the DMA controller DREQ[1:0]# input is de-asserted.

If BLAST# output is not required for the last Lword of a DMA transfer (bit [15]=1), the DMA controller releases the Data bus after it receives an external READY# or the internal wait state counter decrements to 0 for the current Lword. If the DMA controller is currently bursting data, which is not the last Data phase for the burst, BLAST# is not asserted.

If BLAST# output is required for the last Lword of the DMA transfer (bit [15]=0), the DMA controller transfers one or two Lwords. If DREQ[1:0]# is de-asserted during the Address phase of the first transfer in the IOP 480 Local Bus ownership, the DMA controller completes the current Lword, and one additional Lword (this allows BLAST# output to be asserted during the final Lword). If the DMA FIFO is full/empty after the Data phase in which DREQ[1:0]# is de-asserted, the second Lword is not transferred.

DREQ[1:0]# controls only the number of Lword transfers. For an 8-bit bus, the IOP 480 releases the bus after transferring the last byte of the Lword. For a 16-bit bus, the IOP 480 releases the bus after transferring the last word of the Lword.

7.1.9 Local Bus EOT

The DMA controller terminates a transfer on an Lword boundary after EOT# is asserted. For an 8-bit bus, the IOP 480 terminates after transferring the last byte of the Lword. For a 16-bit bus, the IOP 480 terminates after transferring the last word of the Lword.

For Scatter/Gather DMA (Channels 0 and 1), a bit exists for continuing the DMA transfer of next links after EOT# is asserted (C0MODE[20] and/or C1MODE[20]). If this bit is set to 1, next links are transferred after EOT# is asserted. If this bit is set to 0, EOT# stops all links of DMA.

During the descriptor loading from the Local Bus, assertion of EOT# causes a complete descriptor load and no subsequent Data transfer; however, this is not recommended. This has no effect when the descriptor is loaded from the PCI Bus.

The DMA Fast/Slow Terminate Mode Select bit(s) (C0MODE[15] and/or C1MODE[15]) determines whether BLAST# is asserted after a DMA controller detects EOT# input.

If BLAST# output is not required for the last Lword of the DMA transfer (C0MODE[15]=1 and/or C1MODE[14]=1, the DMA controller releases the data bus and terminates DMA after it receives an external READY#. When used, the internal wait state counter decrements to 0 for the current Lword. If the DMA controller is currently bursting data that is not the last data phase for the burst, BLAST# output is not asserted.

If BLAST# output is required for last Lword of the DMA transfer (C0MODE[15]=1 and/or C0MODE[15]=1), the DMA controller transfers one or two Lwords. If EOT# is asserted, the DMA controller completes the current Lword and one additional Lword if BLAST# is not already being asserted (this allows BLAST# output to be asserted during the final Lword).

EOT# can be used to end all transfers on the current channel or to simply skip to the next link in a chain (C0MODE[20] and/or C1MODE[20]).

When EOT# is used to end a Scatter/Gather DMA cycle, the number of bytes remaining to be transferred can automatically be written to the descriptor transfer count field by enabling the DMA Clear Count Mode bit(s) (C0MODE[16]=1 and/or C1MODE[16]=1).

7.1.10 DMA Abort

DMA transfers can be aborted, in addition to using the EOT# signal, by performing the following sequence:

1. Set the Channel Enable bit(s) (C0CSR[0]=1 and/or C1CSR[0]=1).
2. Set the Channel Start bit(s) (C0CSR[1]=1 and/or C1CSR[1]=1).
3. Clear the DMA Channel Enable bit(s) (C0CSR[0]=0 and/or C1CSR[0]=0).
4. Abort DMA by setting the Channel Abort bit(s) (C0CSR[2]=1 and/or C1CSR[2]=1).
5. Wait until the Channel Done bit(s) is set (C0CSR[4]=1 and/or C1CSR[4]=1).

Note: *One to two data transfers occur after the Abort bit is set. Aborting when no DMA cycles are in progress causes the next DMA to abort.*

7.1.11 Local Bus Latency and Pause Timers

The Local Bus Latency and Pause Timers are programmable using the Local Bus Timers register (LOCTMR[7:0] and/or LOCTMR[23:16], respectively). If the Local Bus Latency Timer is enabled and expires (LOCTMR[7:0]=0h), the IOP 480 completes the current Lword transfer and releases LHOLD. After the programmable Pause Timer expires (LOCTMR[23:16]=0h), it re-asserts LHOLD. It continues to transfer when it receives LHOLDA. The PCI Bus transfer continues until the FIFO is empty for a Local-to-PCI transfer or full for a PCI-to-Local transfer.

7.1.12 DMA Unaligned Transfers

As a Local Bus Master, the IOP 480 supports unaligned transfers.

For a 32-bit bus, if a DMA transfer does not start at an address with LAD[3:2] = 00, then a single Lword transfer is first performed, followed by Lword bursts. For a 16-bit bus, if the DMA transfer does not start at an address with LAD[2:1] = 00, then a byte transfer is first performed, followed by word bursts. For an 8-bit bus, a DMA transfer may start bursting at any address.

For unaligned Local-to-PCI transfers, the IOP 480 reads a partial Lword from the Local Bus. It continues to read Lwords from the Local Bus. Lwords are assembled, aligned to the PCI Bus address, and loaded into the FIFO.

For PCI-to-Local transfers, Lwords are read from the PCI Bus and loaded into the FIFO. On the Local Bus, Lwords are assembled from the FIFO, aligned to the Local Bus address and written to the Local Bus.

On both the PCI and Local Buses, the byte enables for writes determine LAD[1:0] for the start of a transfer. For the last transfer, byte enables specify the bytes to be written. All reads are Lwords.

7.1.13 PCI Memory Write and Invalidate (MWI)

The IOP 480 can be programmed to perform Memory Write and Invalidate cycles (MWI) to the PCI Bus for DMA transfers, as well as Direct Master transfers. The IOP 480 supports Memory Write and Invalidate transfers for cache line sizes of 8 or 16 Lwords. Size is specified in the System Cache Line Size bits (PCICLSR[7:0]). If a size other than 8 or 16 is specified, the IOP 480 performs Write transfers rather than Memory Write and Invalidate transfers.

DMA Memory Write and Invalidate transfers are enabled when the DMA controller Memory Write and Invalidate Enable bit(s) (C0MODE[13] and/or C1MODE[13]) and the PCI Memory Write and Invalidate Enable bit (PCICR[4]) are set.

In Memory Write and Invalidate mode, the IOP 480 waits until the number of Lwords required for specified cache line size are read from the Local Bus before starting the PCI access. This ensures a full cache line write can complete in one PCI Bus ownership. If a target disconnects before a cache line completes, the IOP 480 completes the remainder of that cache line, using normal writes, before resuming Memory Write and Invalidate transfers. If a Memory Write and Invalidate cycle is in progress, the IOP 480 continues to burst into the next cache line if the data is in the FIFOs. Otherwise, the IOP 480 terminates the burst and waits for the next cache line to be read from the Local Bus. If the final transfer is not a full cache line, the IOP 480 completes the DMA transfer using normal writes.

7.1.14 DMA Descriptor Ring Management (Valid Mode)

In Scatter/Gather DMA mode, when the Valid Mode Enable Bit (C0MODE[18] and/or C1MODE[18]) is set to 0, the Valid bit (bit 31 of transfer count) is ignored. When the Valid Mode Enable Bit (C0MODE[18] and/or C1MODE[18]) is set to 1, the DMA descriptor proceeds only when the Valid bit is set. If the Valid bit is set, the transfer count is 0, and the descriptor is not the last descriptor, then the DMA controller moves on to the next descriptor in the chain.

When the Valid Stop Control bit (C0MODE[19] and/or C1MODE[19]) is set to 0, the DMA Scatter/Gather controller continuously polls the descriptor with the Valid bit set to 0 (invalid descriptor) until the Valid bit is read to be a 1. When the Valid Stop Control bit (C0MODE[19] and/or C1MODE[19]) is set to 1, the DMA Scatter/Gather controller pauses if a Valid bit with a value of 0 is detected. In this case, the IOP 480 CPU must restart the DMA controller by setting bit 1 of the DMA Control/Status register (C0CSR[1] and/or C1CSR[1]). When the DMA Clear Count Mode bit(s) is set (C0MODE[16]=1 and/or C1MODE[16]=1), the Valid bit is cleared at the completion of each descriptor.

7.1.15 DMA Priority

The DMA Channel Priority bits (LARBR[5:4]) can be used to specify the following priorities:

- Rotating (LARBR[5:4]=00)
- DMA Channel 0 (LARBR[5:4]=01)
- DMA Channel 1 (LARBR[5:4]=10)

7.1.16 Local Bus Arbitration

The IOP 480 DMA controller releases control of the Local Bus (de-asserts LHOLD) when one of the following conditions occurs:

- Local Bus Latency Timer is enabled and expires (LOCTMR[7:0])
- LHOLDREQ is asserted (LHOLDREQ can be enabled or disabled, or gated with a Local Bus Latency Timer before the IOP 480 releases the Local Bus)
- Direct Slave access is pending (only in Direct Slave High-Priority mode)
- EOT# input is received (if enabled)

7.1.17 PCI Bus Arbitration

The DMA controller releases control of the PCI Bus when one of the following conditions occurs:

- FIFOs are full or empty
- PCI Bus Latency Timer expires (PCILTR[7:0])—and the IOP 480 loses the PCI GNT# signal
- Target Disconnect response is received

The DMA controller de-asserts its PCI Bus request (REQ#) for a minimum of two PCI clocks.

7.1.18 DMA Local Bus Arbitration

7.1.18.1 Local Latency and Pause Timers

The Local Bus Latency and Pause Timers are programmable using the Local Bus Timers register (LOCTMR[7:0] and/or LOCTMR[23:16], respectively). The DMA controller completes the current Lword transfer and releases the bus if the Local Latency Timer expires. After its programmable Pause Timer expires, it re-requests the Local Bus. When it is granted the Local Bus, it continues with the transfer.

7.1.18.2 DRAM Refresh Timers

The IOP 480 has built-in programmable memory refresh timers. If the refresh timers expire (DRAMCTL[22:12]), the DMA controller completes the current Lword transfer and releases the bus. After its programmable Pause Timer expires, it re-requests the Local Bus. When it is granted the Local Bus, it continues with the transfer.

7.1.18.3 Local Arbiter Priority

If Direct Slave accesses are programmed to have a higher priority than DMA accesses (LARBR[3:1]), the Direct Slave access preempts DMA. When a Direct Slave access occurs, the DMA controller releases the Local Bus within two word transfers. After its programmable Pause Timer expires, it again requests the Local Bus. When it is granted the Local Bus, it continues with the transfer.

7.1.19 DMA Master Command Codes

The IOP 480 DMA controllers can assert the following Memory cycles.

Table 7-2. DMA Master Command Codes

Command Type	Code (C/BE[3:0]#)
Memory Read	0110 (6h)
Memory Write	0111 (7h)
Memory Read Multiple	1100 (Ch)
Dual Address Cycle	1101 (Dh)
Memory Read Line	1110 (Eh)
Memory Write and Invalidate	1111 (Fh)

7.2 DMA CHANNEL 2

7.2.1 Overview

The IOP 480 DMA Channel 2 Controller supports transferring data between two Local Bus Memory regions or Flyby Data transfers between Local memory and a Local I/O device. It supports an End of Transfer pin (EOT2#) and Demand mode DMA transfers. Master mode must be enabled with the Master Enable bit (PCICR[2]=1) before the IOP 480 can become a PCI Bus Master. DMA Channel 2 has a programmable 4-Lword bidirectional FIFO. Additionally, DMA Channel 2 can be programmed to:

- Operate in 8-, 16-, or 32-bit Local Bus widths
- Use from 0 to 15 internal wait states (Local Bus)
- Enable/disable internal wait states (Local Bus)
- Enable/disable Local Bus burst capability
- Local Bus burst to four limit
- Hold Local address constant (Local Target is FIFO) or increment
- Pause Local transfer with/without BLAST# (DMA Fast/Slow termination)
- Assert PCI interrupt (INTA#) or Local interrupt (INTO) when DMA transfer is complete

The Local Latency Timer determines the number of Local clocks the IOP 480 can own the Bus before relinquishing the Local Bus. The Local Pause Timer sets how soon DMA Channel 2 can request the Local Bus. The Local Pause Timer starts after LHOLDA is de-asserted.

7.2.2 DMA Register Access

7.2.2.1 Access from the Local Bus

The internal IOP 480 CPU or a Local processor can access the IOP 480 DMA registers through the base address specified in the Local Base Address for Configuration Register Access register (CFGBA).

7.2.2.2 Access from the Primary PCI Bus

A PCI Master can access the DMA registers through the address specified in the PCI Base Address for Memory Accesses to Configuration Registers and Local Space 0 register (PCIBAR0).

7.2.3 Local-to-Local Mode DMA

The initiator designates the Local source address, Local destination address, and transfer count. The initiator then sets a control bit to initiate the transfer. Once the transfer completes, the IOP 480 can generate an interrupt (programmable). The transfer terminates when one of the following occurs:

- Number of bytes specified in the Transfer Count register have been transferred
- EOT2# input is asserted
- DMA is aborted

The source and destination memory regions are independent. The bus width, type, and performance characteristics of the bus regions can be different. (Refer to Figure 7-10.)

7.2.4 Demand Mode

The Source and Destination Demand Mode bits (C2MODE[12:11]) can be used to specify that the channel operates in Demand mode. In Demand mode, the user sets up the DMA controller's configuration registers and initiates a transfer. Data is transferred when the DMA channels DREQ2# input is asserted. The IOP 480 asserts DACK2# to indicate that the current Local Bus transfer is in response to the DREQ2# input. The minimum transfer size per DREQ2# input is one Lword (32 bits). This may result in multiple transfers for an 8- or 16-bit bus.

Note: *The above transfer information is correct, unless the transfer size is less than one Lword.*

When set to 1, the Source Demand Mode bit (C2MODE[12]) causes the DMA controller to operate in Demand mode while reading source data. In Demand mode, the DMA controller transfers data when its DREQ2# input is asserted. The IOP 480 asserts DACK2# to indicate the current Local bus transfer is in response to the DREQ2# input. The DMA controller transfers Lwords (32 bits) of data. This may result in multiple transfers for an 8- or 16-bit bus.

When set to 1, the Destination Demand Mode bit (C2MODE[11]) causes the DMA controller to operate in Demand mode while writing destination data. In Demand mode, the DMA controller transfers data when its DREQ2# input is asserted. The IOP 480 asserts DACK2# to indicate the current Local bus transfer is in response to the DREQ2# input. The DMA controller transfers Lwords (32 bits) of data. This may result in multiple transfers for an 8- or 16-bit bus.

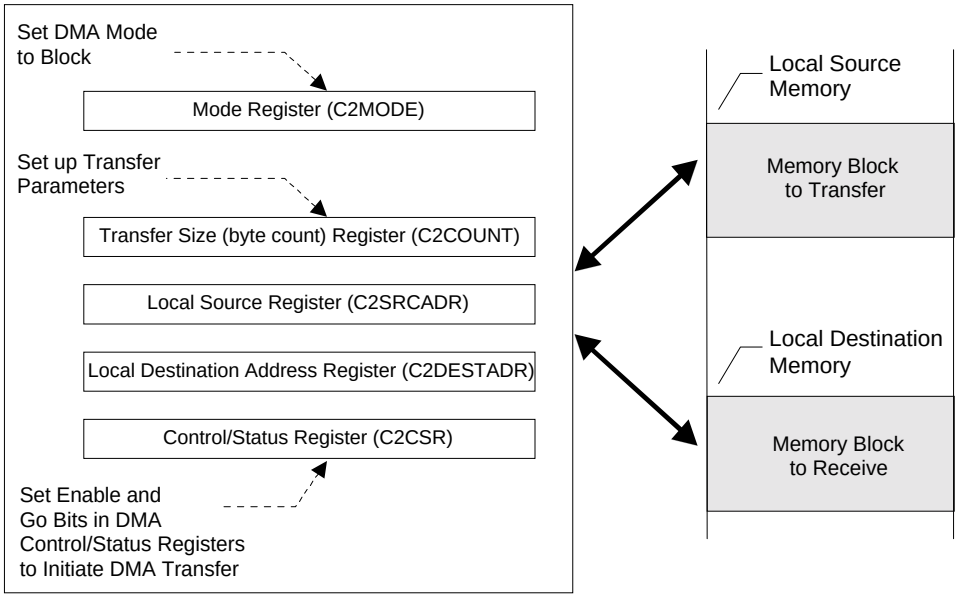


Figure 7-10. Local-to-Local DMA Initialization

7.2.5 Fast Terminate Mode

C2MODE[15] can be used to specify Fast/Slow Terminate mode. When set to 0 and EOT# is asserted, or DREQ2# or BLAST# is de-asserted, the DMA controller completes the current Lword transfer and then asserts BLAST# and completes the next Lword transfer. If BLAST# is asserted when EOT# or DREQ# are asserted, then the DMA controller transfers only one Lword.

When set to 1, the DMA controller stops transferring data after the current Lword, even if a new BLAST# is asserted. (Refer to Table 7-3.)

Table 7-3. Stop Transfer Modes

Register (C2MODE[15])	BLAST# Signal	Number of Lwords Transferred
0	1	2
0	0	1
1	x	1

7.2.6 DMA Abort

DMA transfers can be aborted, in addition to using the EOT# signal, by performing the following sequence:

1. Set the Channel Enable bit (C2CSR[0]=1).
2. Set the Channel Start bit (C2CSR[1]=1).
3. Clear the DMA Channel Enable bit (C2CSR[0]=0).
4. Abort DMA by setting the Channel Abort bit (C2CSR[2]=1).
5. Wait until the Channel Done bit is set (C2CSR[4]=1).

Note: *One to two data transfers occur after the Abort bit is set. Aborting when no DMA cycles are in progress causes the next DMA to abort.*

7.2.7 Flyby DMA

The IOP 480 Flyby mode is the most efficient DMA mode because it combines read/write into a single Bus access. Setting the Flyby Mode bit (C2MODE[10]=1) specifies the transfer direction and the transfer count. In Flyby mode, the transfer direction designates whether the destination address is read or the destination address is written (C2MODE[9]). Set the Start and Enable bits (C2CSR[1:0]=1, respectively) to initiate the transfer. The transfer starts when a Local I/O device asserts the DREQ2# input (C2MODE[11]=1).

Alternatively, Flyby mode can also work without being in Demand mode, in which case transfers start immediately (C2CSR[1:0]) without requiring DREQ2# input (C2MODE[11]=0).

The IOP 480 asserts ADS#, ALE, and DACK2#, and drives the memory address onto the Local Bus for one clock cycle. The DMA controller then floats LAD[31:0]. DACK2# indicates that the Flyby DMA is in progress. A turnaround cycle must be provided by the source before data may be driven onto the Local Address/ Data Bus. DACK2#, being asserted along with the READY# output, indicates to the I/O device that Read data is available or Write data has been accepted.

Once the transfer completes, the IOP 480 can generate an interrupt (programmable). The transfer terminates when the number of bytes specified in the transfer count register have been transferred or the EOT2 input is asserted, or the DMA is aborted.

The data continues to be transferred when the DREQ2# input is asserted. The minimum transfer size per DREQ2# input is one Lword (32 bits). This may result in multiple transfers for an 8- or 16-bit bus.

For Flyby Data transfers, the source and destination bus widths must be the same. For best performance, the Bterm and Burst Mode bits should be enabled for the destination bus region address.

7.2.8 Local Bus Arbitration

The IOP 480 DMA controller releases control of the Local Bus (de-asserts LHOLD) when one of the following conditions occurs:

- Local Bus Latency Timer is enabled and expires (LOCTMR[7:0])
- LHOLDREQ is asserted (LHOLDREQ can be enabled or disabled, or gated with a Local Bus Latency Timer before the IOP 480 releases the Local Bus)
- Direct Slave access is pending (only in Direct Slave High-Priority mode)
- EOT# input is received (if enabled)

7.2.9 DMA Unaligned Transfers

As a Local Bus Master, the IOP 480 supports unaligned transfers.

For a 32-bit bus, if a DMA transfer does not start at an address with LAD[3:2] = 00, then single Lword transfers are first performed, followed by Lword bursts. For a 16-bit bus, if the DMA transfer does not start at an address with LAD[2:1] = 00, then single word transfers are first performed, followed by word bursts. For an 8-bit bus, a DMA transfer may start bursting at any address.

On the Local Bus, the byte enables for writes determine LAD[1:0] for the start of a transfer. For the last transfer, byte enables specify the bytes to be written. All reads are Lwords.

7.2.10 DMA Local Bus Arbitration

7.2.10.1 Local Bus Latency and Pause Timers

The Local Bus Latency and Pause Timers are programmable using the Local Bus Timers register (LOCTMR[7:0] and/or LOCTMR[23:16], respectively). If the Local Bus Latency Timer is enabled and expires (LOCTMR[7:0]), the IOP 480 completes the current Lword transfer and releases LHOLD. After its programmable Pause Timer expires (LOCTMR[23:16]), it re-asserts LHOLD. It continues to transfer when it receives LHOLDA.

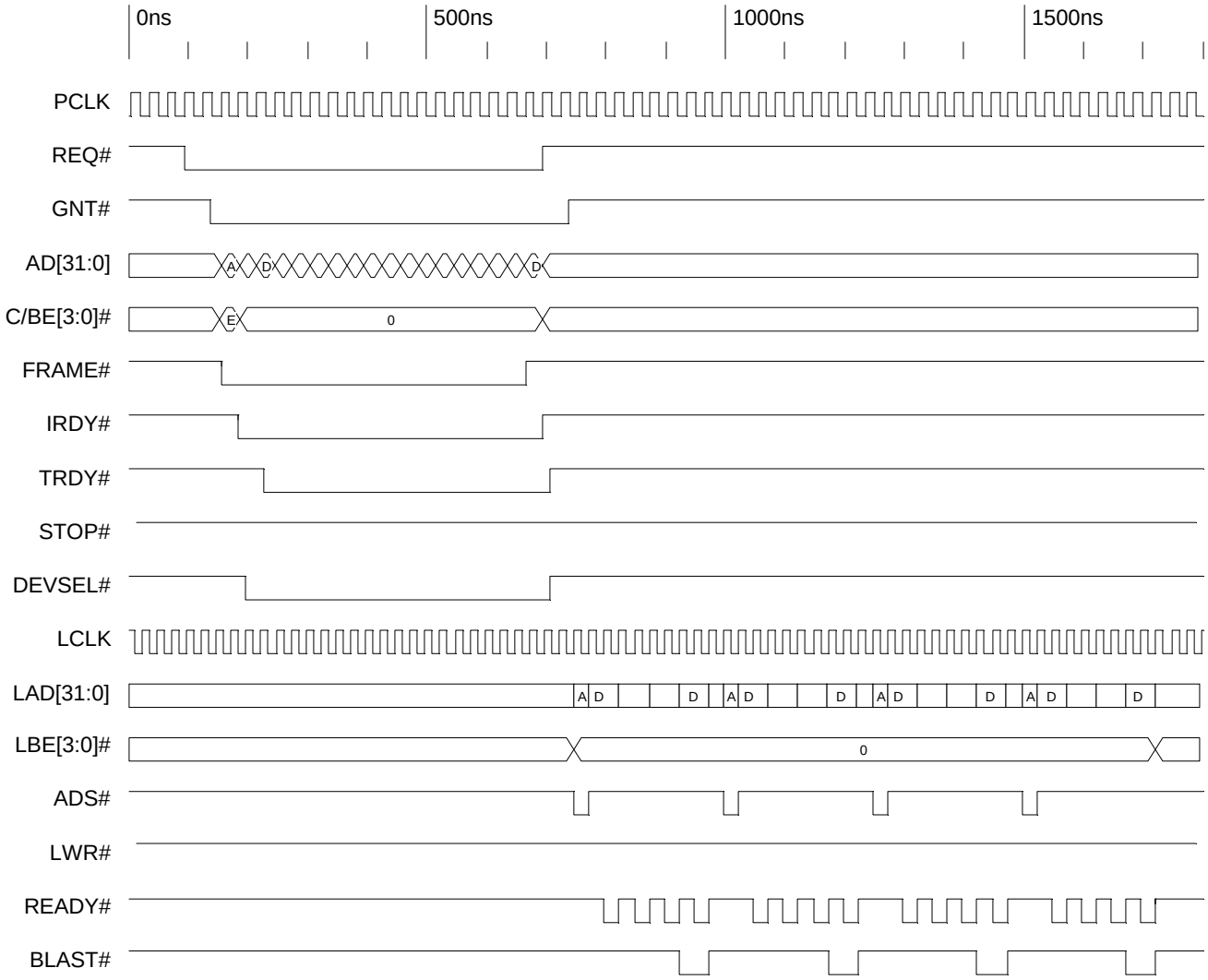
7.2.10.2 DRAM Refresh Timers

The IOP 480 has built-in programmable memory refresh timers. If the refresh timers expire (DRAMCTL[22:12]), the DMA controller completes the current word transfer and releases the bus. After its programmable Pause Timer expires, it re-requests the Local Bus. When it is granted the Local Bus, it continues with the transfer.

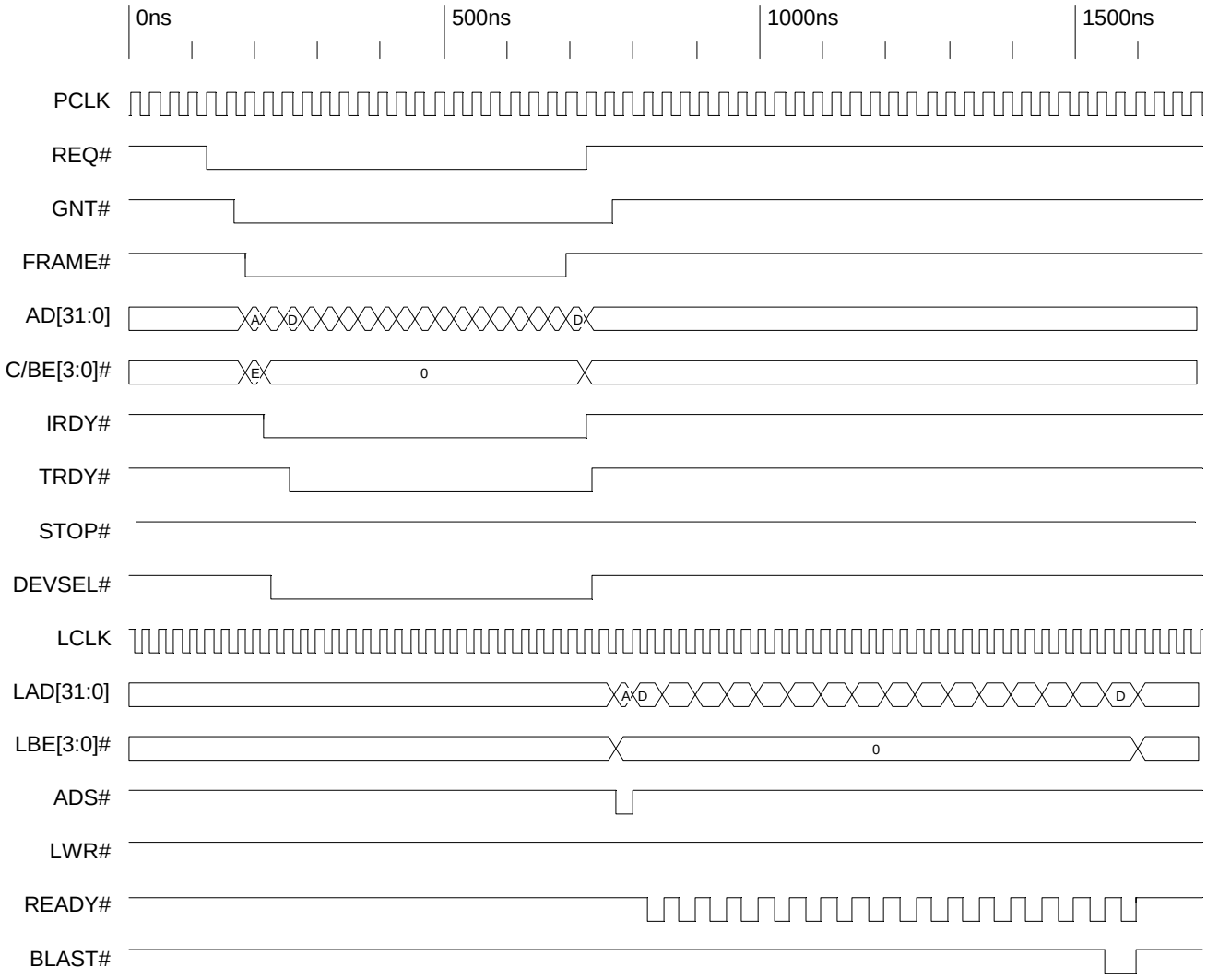
7.2.10.3 Local Arbiter Priority

If Direct Slave accesses are programmed to have a higher priority than DMA accesses (LARBR[3:1]), a Direct Slave access preempts DMA. When a Direct Slave access occurs, the DMA controller gives up the Local Bus within two word transfers. After its programmable Pause Timer expires, the DMA controller again requests the Local Bus. When it is granted the Local Bus, it continues the transfer.

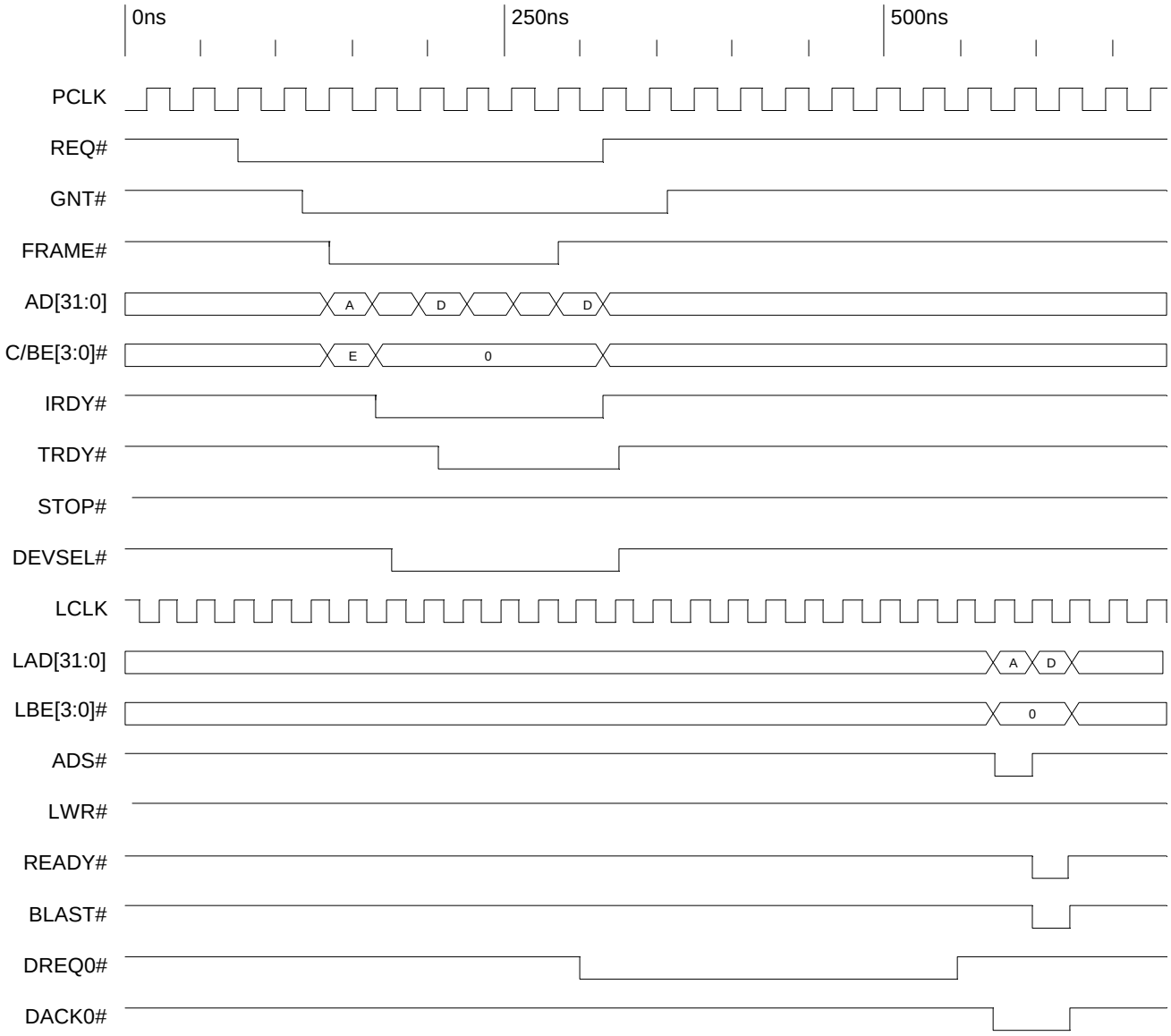
7.3 TIMING DIAGRAMS



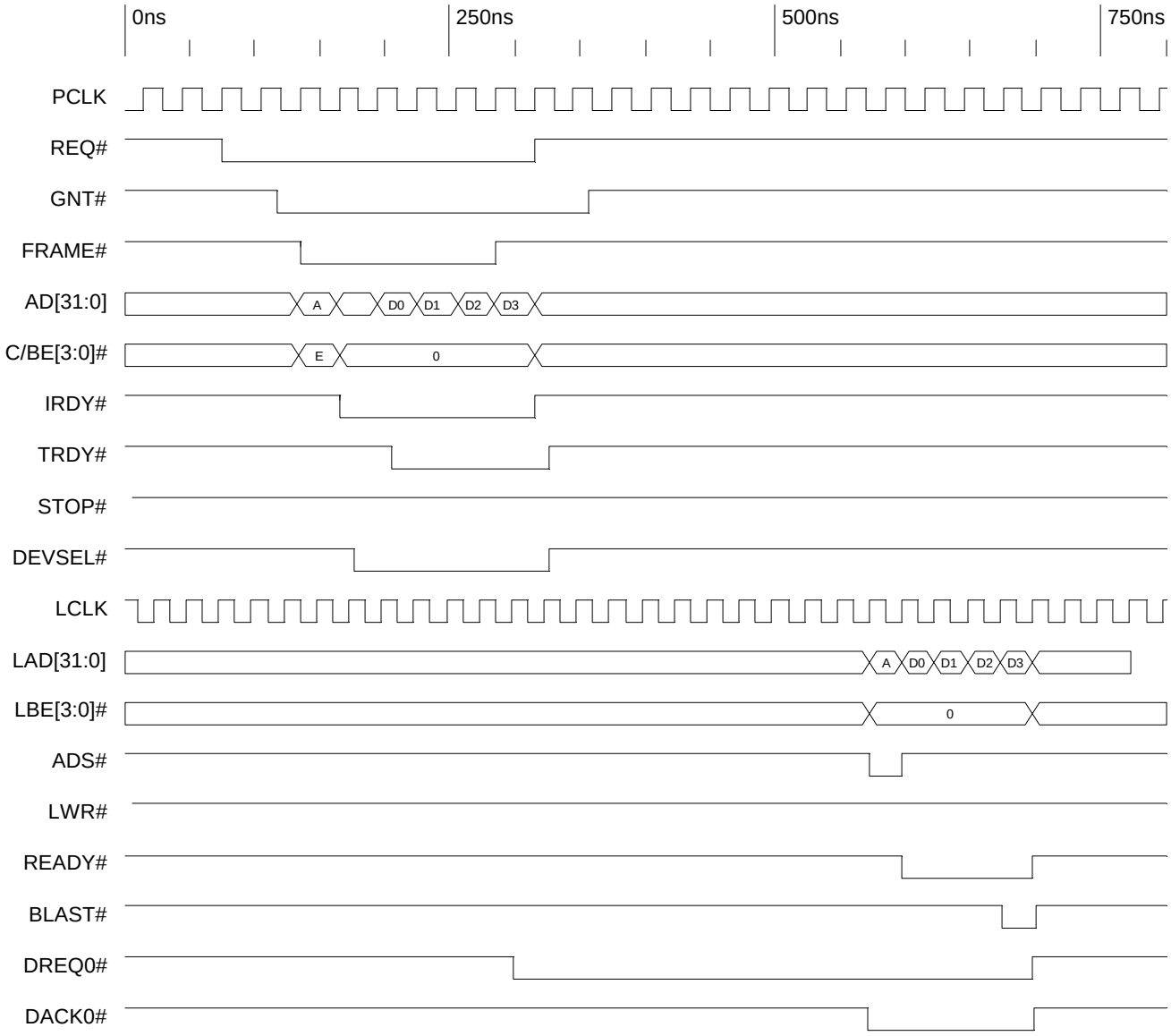
Timing Diagram 7-1. DMA from PCI-to-Local, Bterm Disabled



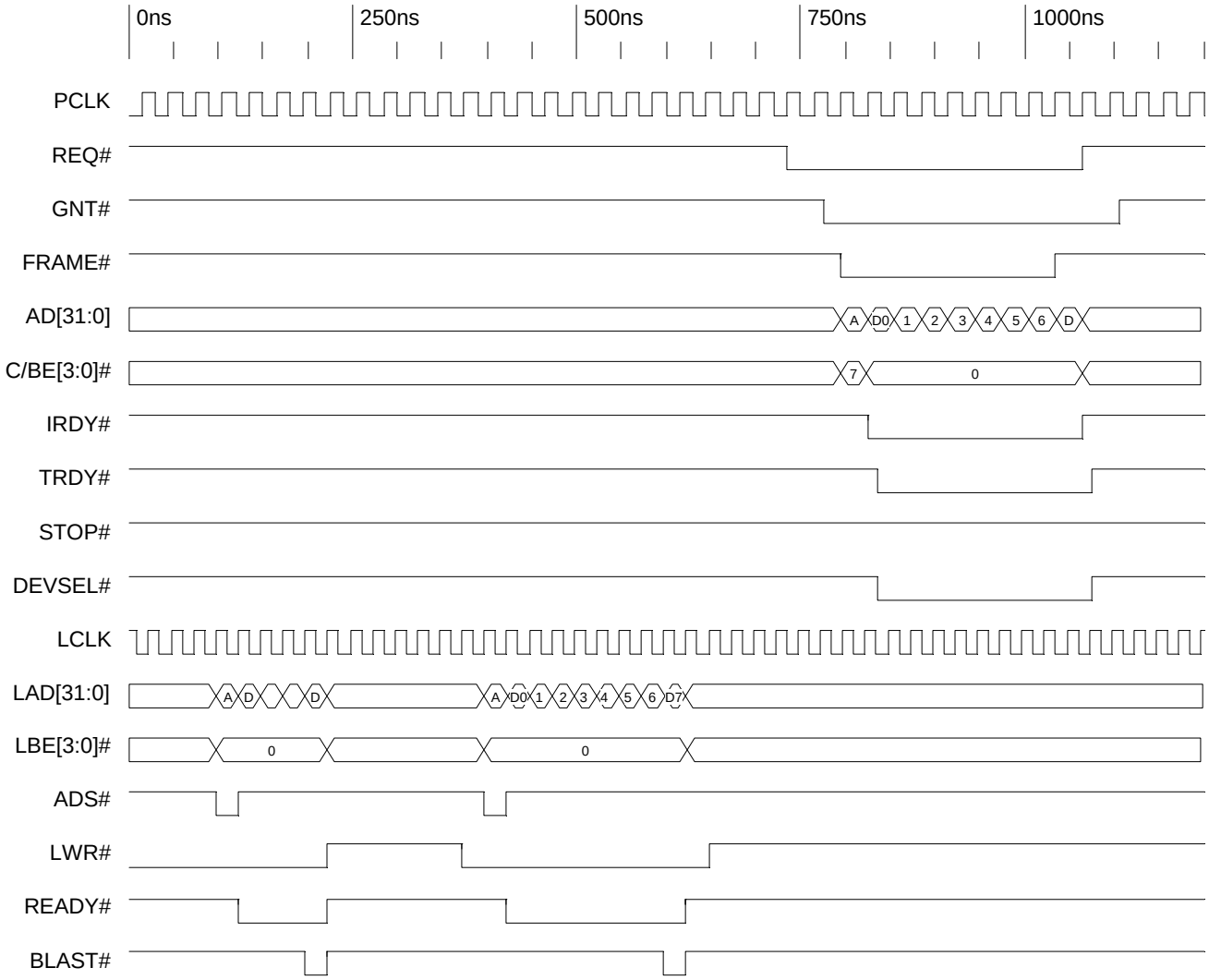
Timing Diagram 7-2. DMA from PCI-to-Local, Bterm Enabled



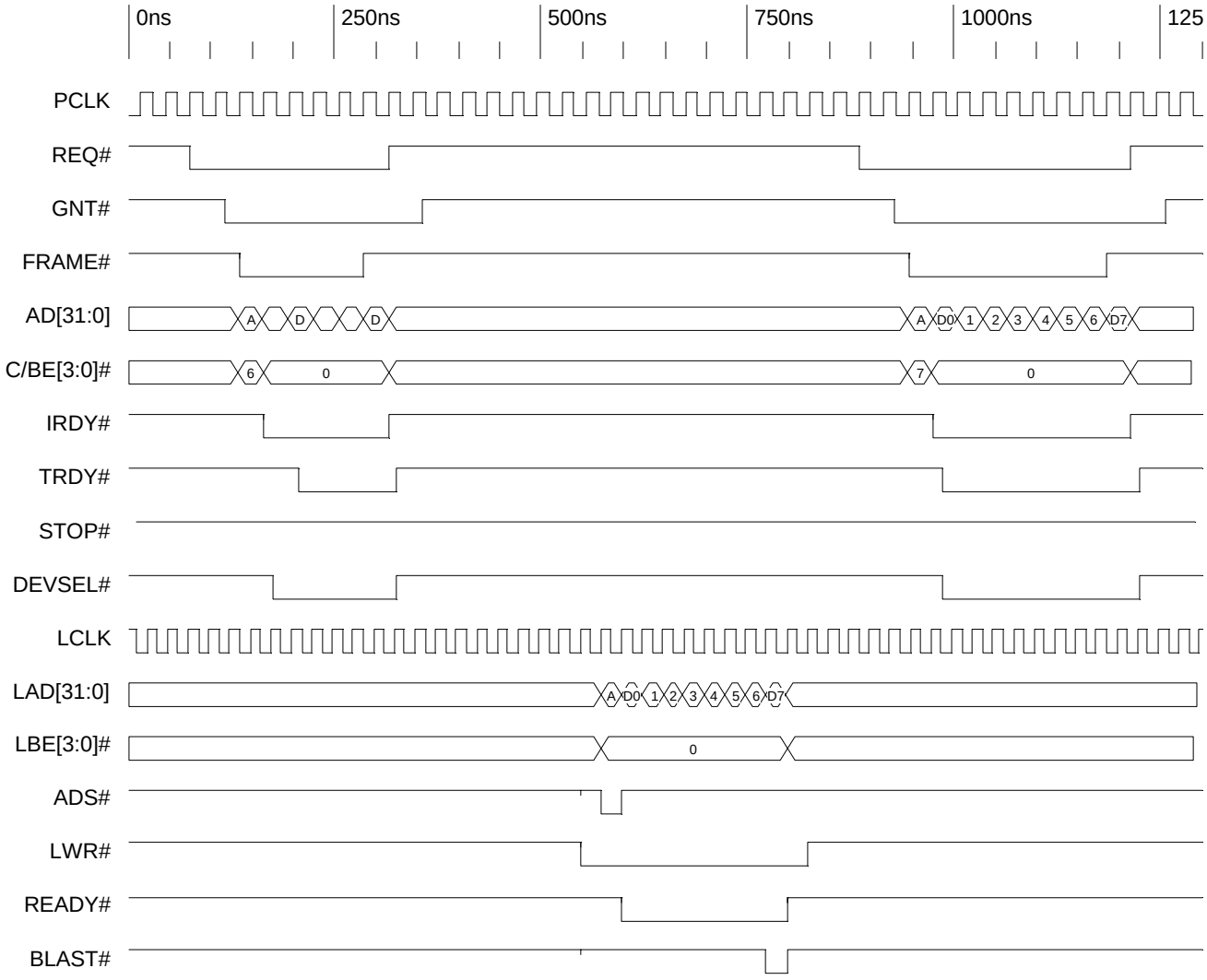
Timing Diagram 7-3. DMA Demand Mode, Write from PCI-to-Local



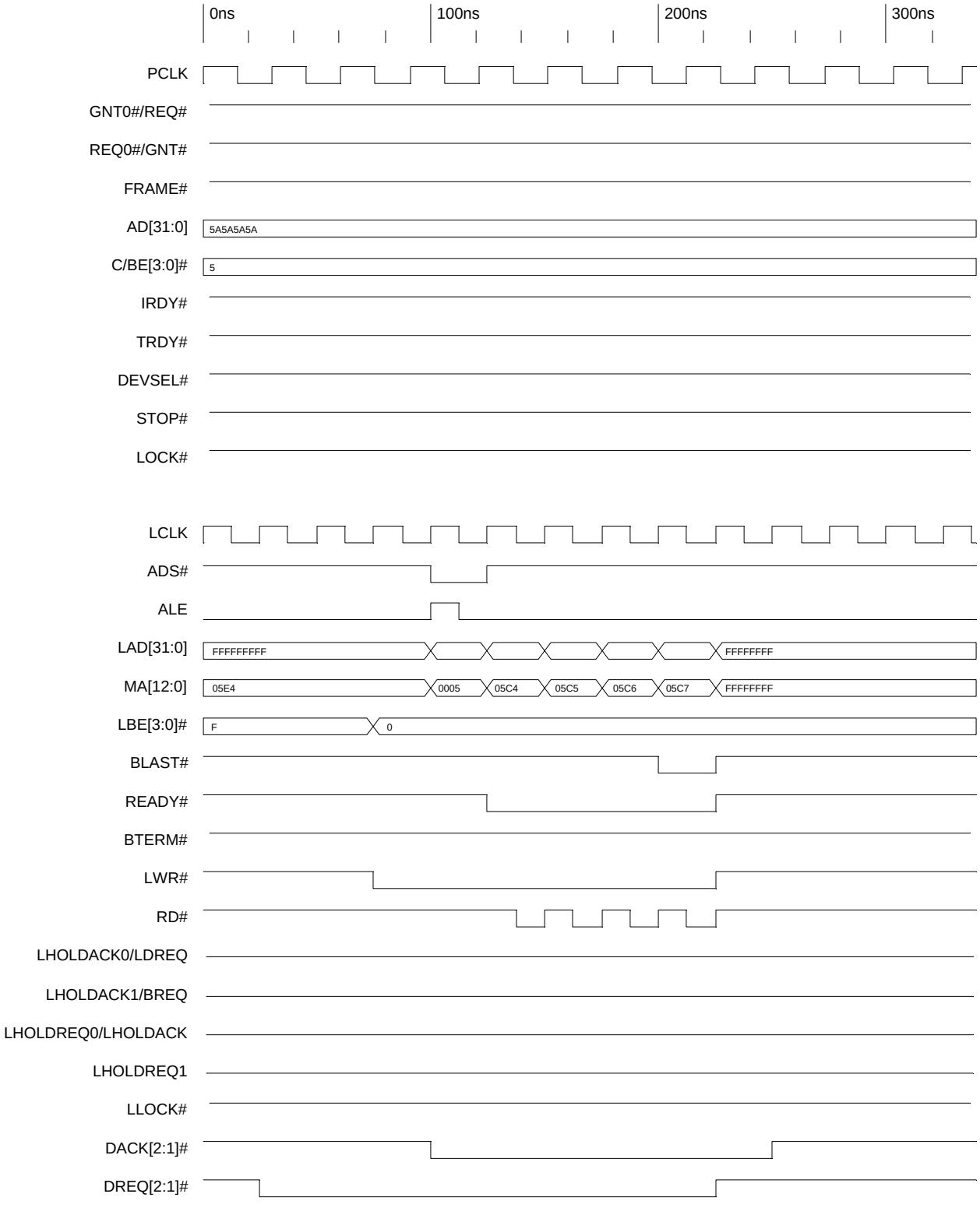
Timing Diagram 7-4. DMA Demand Mode, Write Four Lwords from PCI-to-Local



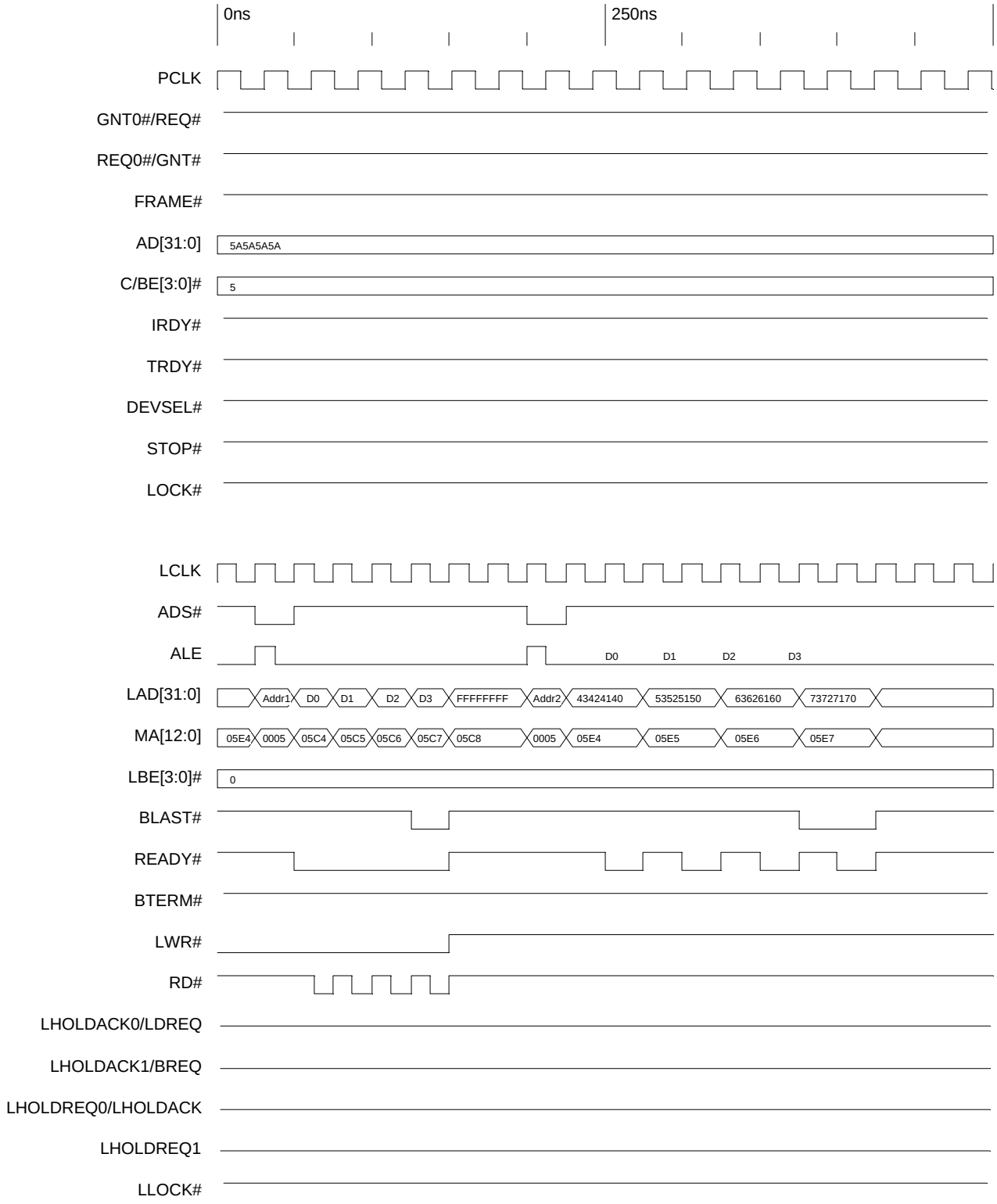
Timing Diagram 7-5. DMA Scatter/Gather with Descriptor on Local Memory



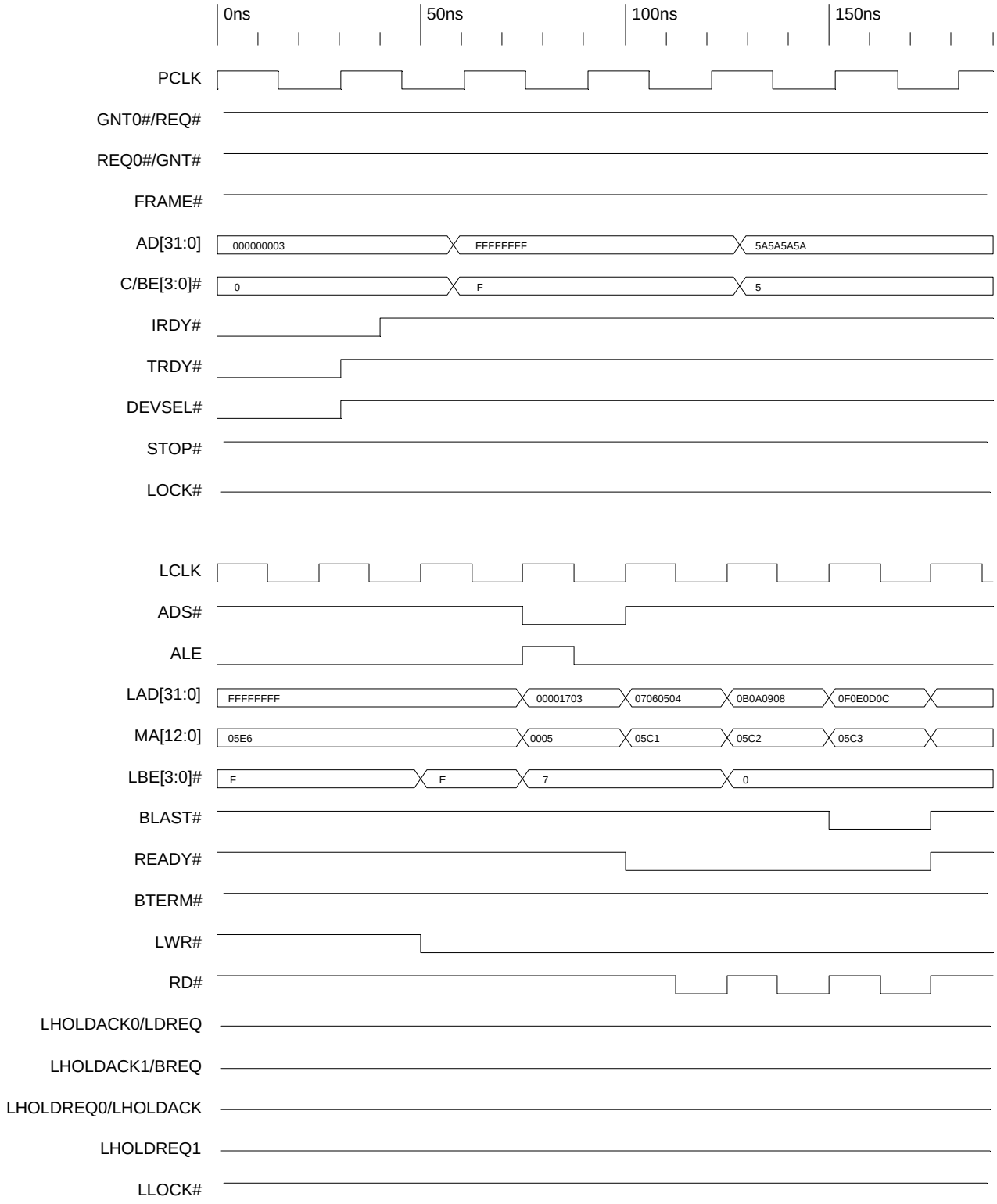
Timing Diagram 7-6. DMA Scatter/Gather with Descriptor on PCI Memory



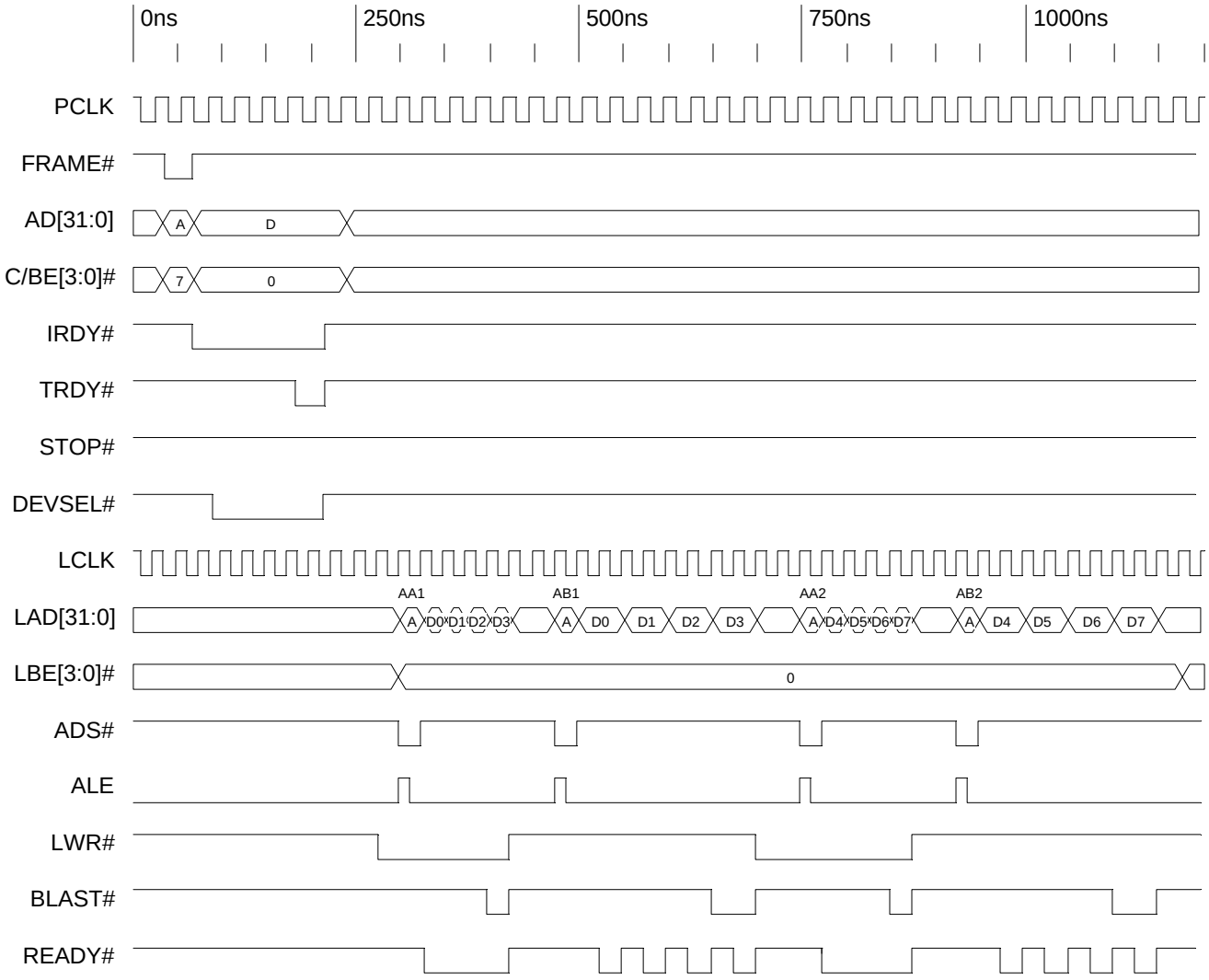
Timing Diagram 7-7. DMA2 Demand Mode Cycle



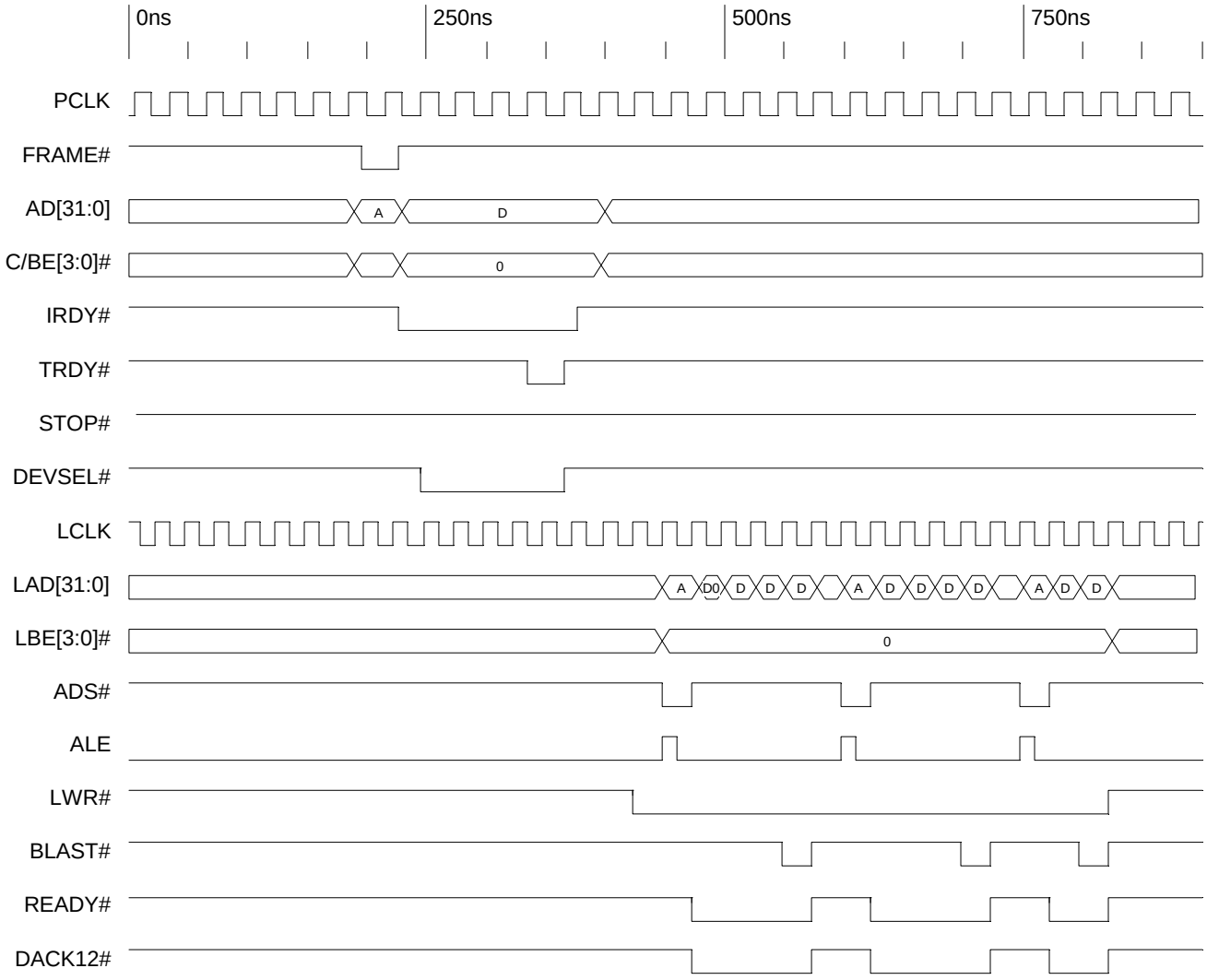
Timing Diagram 7-8. DMA2 Non-Flyby



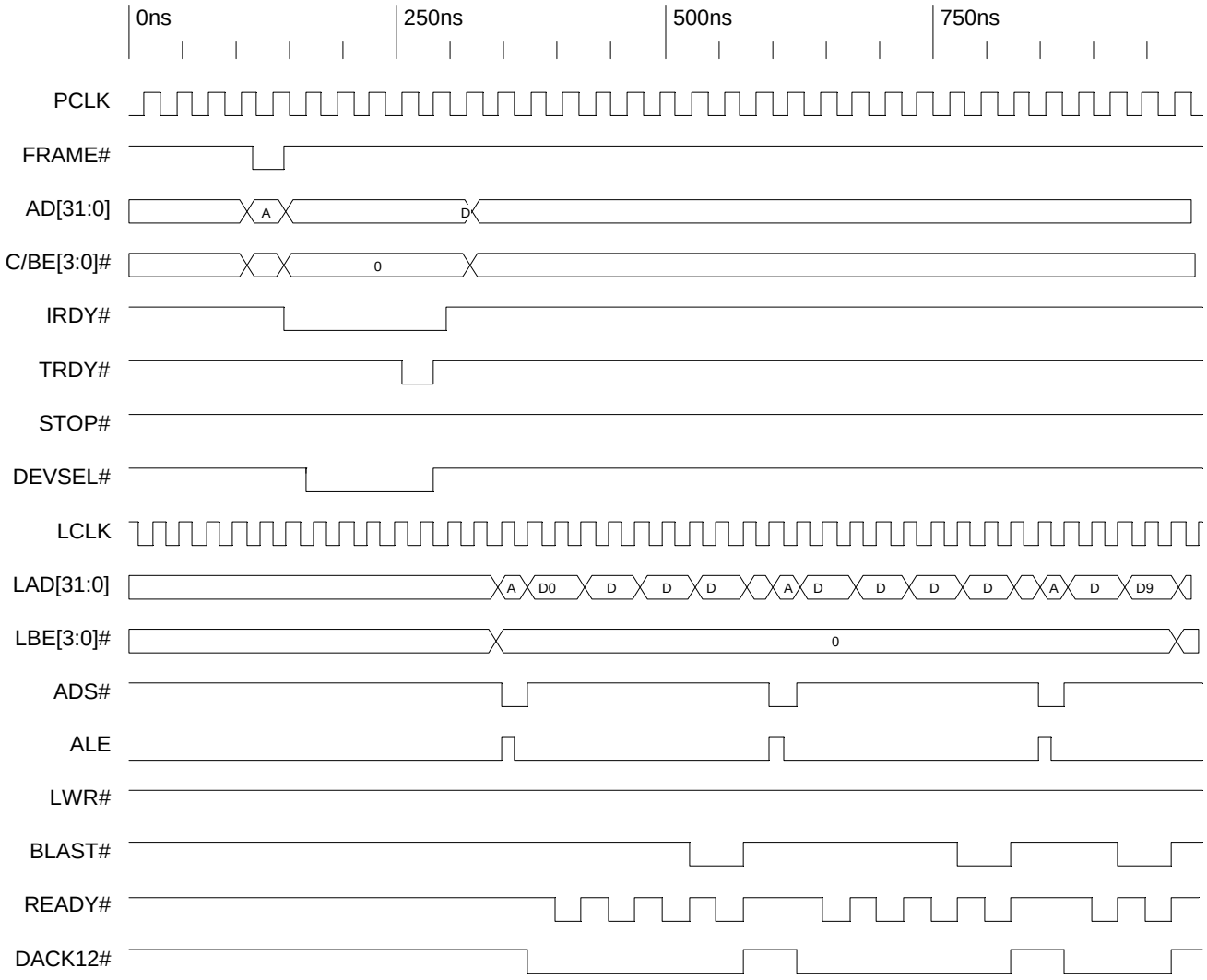
Timing Diagram 7-9. DMA2 Unaligned Transfer



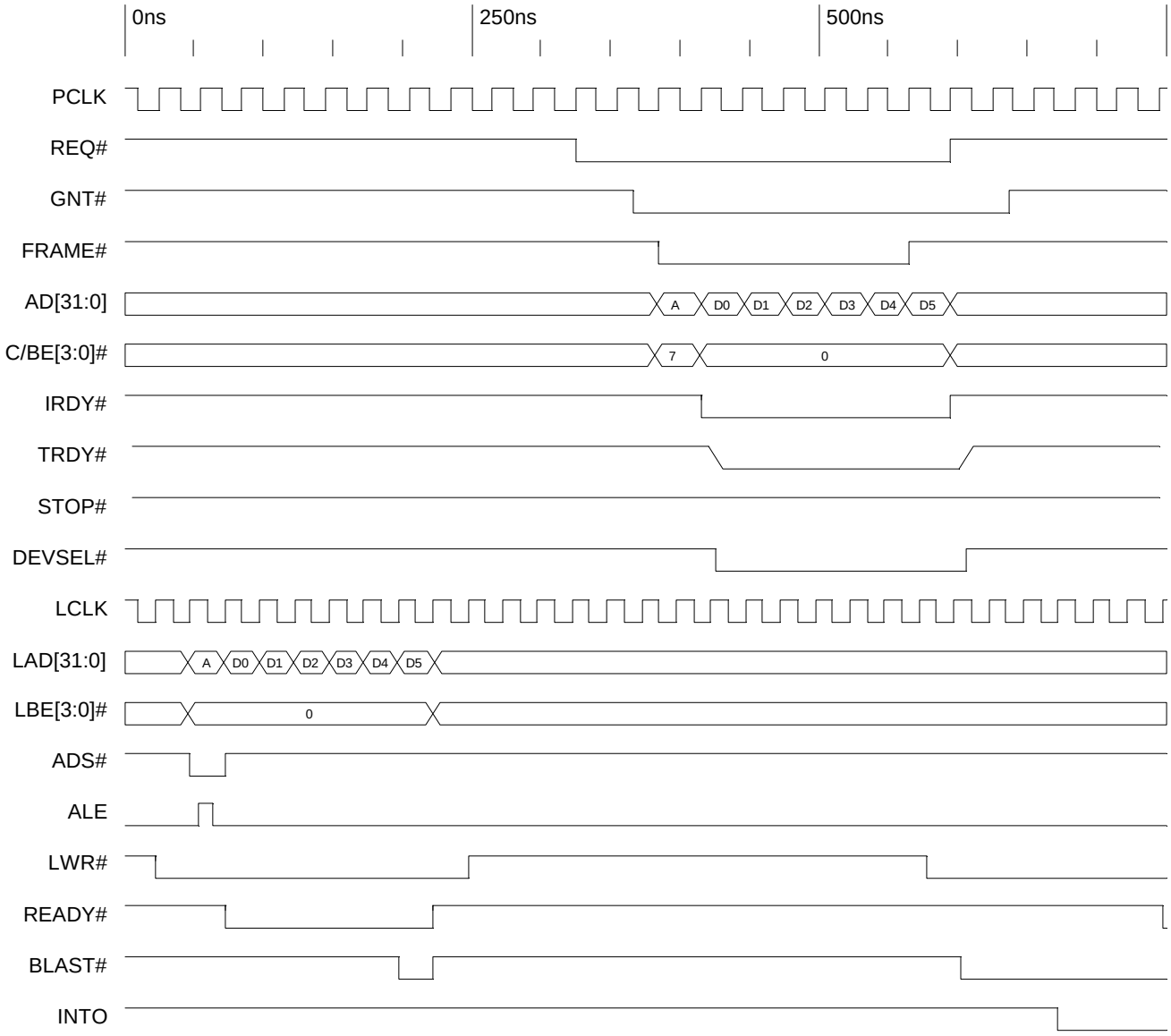
Timing Diagram 7-10. DMA2 Local-to-Local, 8 Lwords, 32-Bit RAM Transfer



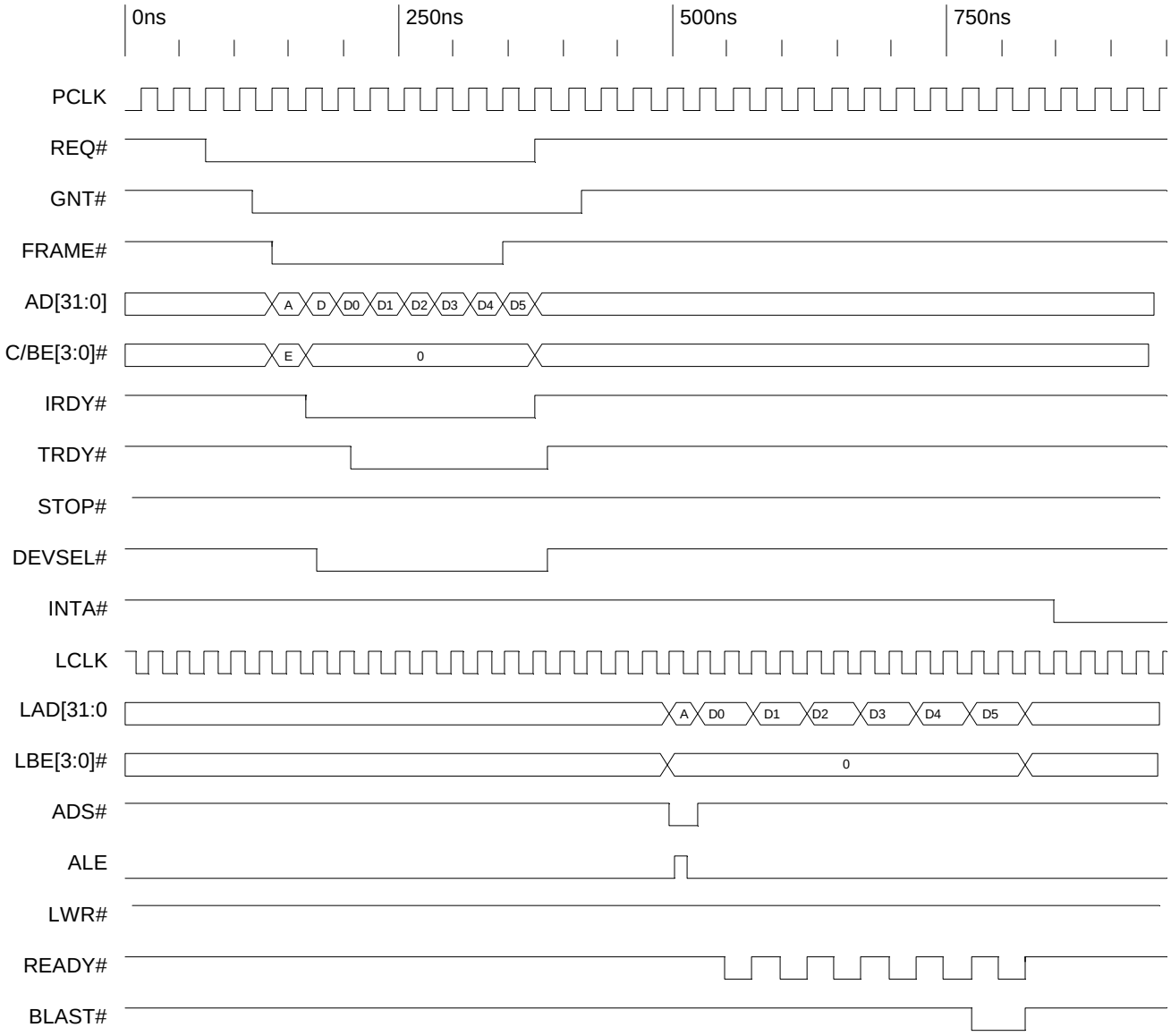
Timing Diagram 7-11. Flyby DMA2 Load Data to 32-Bit Local FIFO



Timing Diagram 7-12. Flyby DMA2 Continue–Write Data to 32-Bit Local RAM



Timing Diagram 7-13. DMA Write from Local-to-PCI, Local Interrupt Done Bit



Timing Diagram 7-14. DMA Write from PCI-to-Local, PCI Interrupt Done Bit

8 LOCAL BUS INTERNAL ARBITER

8.1 OVERVIEW

The IOP 480 supports an external or internal Local Bus arbiter. The default is for the internal Local Bus arbiter to be enabled. If an external bus arbiter is used, then a serial EEPROM must be used to configure the IOP 480.

The IOP 480 has an internal bus arbiter for the Local Bus (LARBR[0]). The following sources can request use of the Local Bus (LARBR[5:1]):

- Internal IOP 480 CPU
- Direct Slave Controller
- Internal PCI/Local DMA Controllers (CH0)
- Internal PCI/Local DMA Controllers (CH1)
- Internal Local-to-Local DMA Controller (CH2)
- LHOLDREQ0/LHOLDACK pin (external Master)
- LHOLDREQ1 pin (external Master)

8.2 INITIALIZATION

After reset, the IOP 480 performs a serial EEPROM check, and then boots the IOP 480 CPU. The IOP 480 CPU boot uses the Local Bus; therefore, the Local Bus must be up and ready after the serial EEPROM check. By default, the internal Local arbiter is enabled.

If the External Preempt Enable bit is set (LARBR[7]=1), the internal Local arbiter supports only one external Local Master, which is Master 0 because BREQ and LHOLDACK1 share the same output pin (LHOLDACK1/BREQ).

8.3 ROUND-ROBIN MODE

The seven sources of Local Bus requests are serviced in a round-robin priority, or one of the requests can be given priority. The selection is made in LARBR, a Local arbitration register. When round-robin is selected, each requester has equal priority. After a requester has been granted the bus, it is rotated to last priority, and other active requests are rotated towards top priority. (Refer to Timing Diagram 8-1.)

8.4 HIGH-PRIORITY MODE

If a single requester is given priority, then other requesters still participate in round-robin arbitration. The bus is granted alternately between the single high priority requester and the highest priority requester of the round-robin arbiter. (Refer to Timing Diagram 8-2.)

The Direct Slave Controller only preempts DMA controllers if the Arbiter mode is set in Direct Slave High-Priority mode (LARBR[3:1]=011b).

When DMA Channels 0 and 1 priority are chosen, (LARBR[3:1]=010b), LARBR[5:4] can be set to maintain a rotational priority scheme between the following:

- DMA0 and DMA1
- DMA0 has priority
- DMA1 has priority

Refresh cycle and Latency Timer expiration (if enabled) automatically preempt the internal IOP 480 CPU, Direct Slave Controller, and internal DMA (Channel 0, 1, and 2) controllers.

If the External Preempt Enable bit is disabled (LARBR[7]=0), the Refresh cycle and Local Latency Timer cannot preempt external Local Masters. If the External Preempt Enable bit is enabled (LARBR[7]=1), and there is a Refresh cycle or Local Latency Timer expiration, LHOLDACK1/BREQ output is asserted to request that an external Local Master release the Local Bus.

When the External Preempt Enable bit is enabled (LARBR[7]=1), only Master 0 should be connected to the IOP 480 Internal Local arbiter.

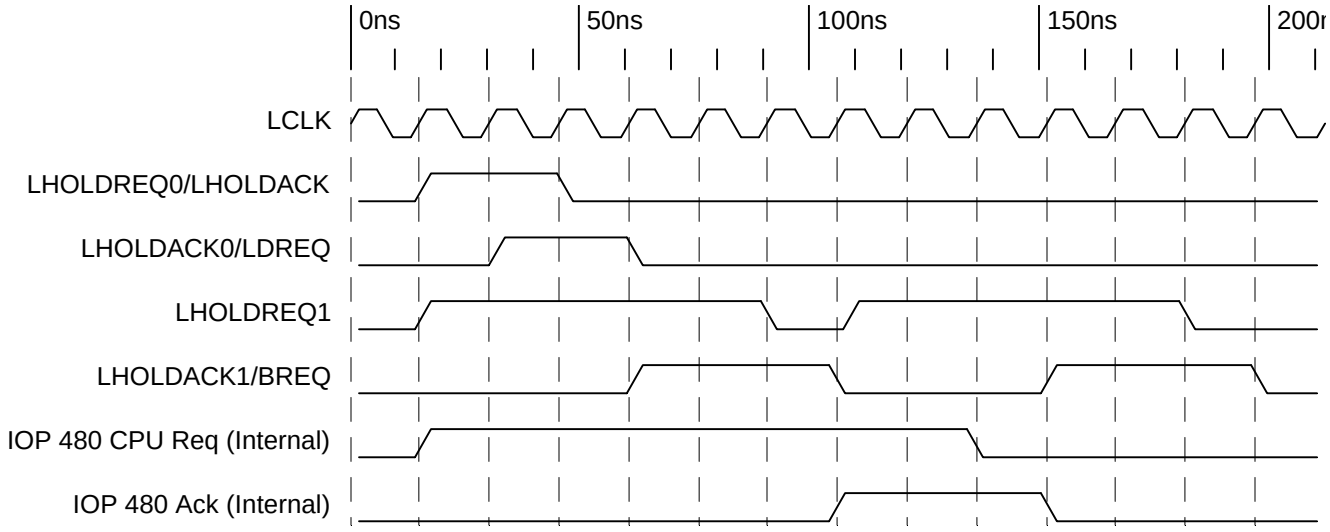
8.5 PERFORMANCE TUNING

Depending on applications, arbiter modes (Round-Robin or High-Priority), and Latency Timer value should be appropriately selected to retain a high bandwidth.

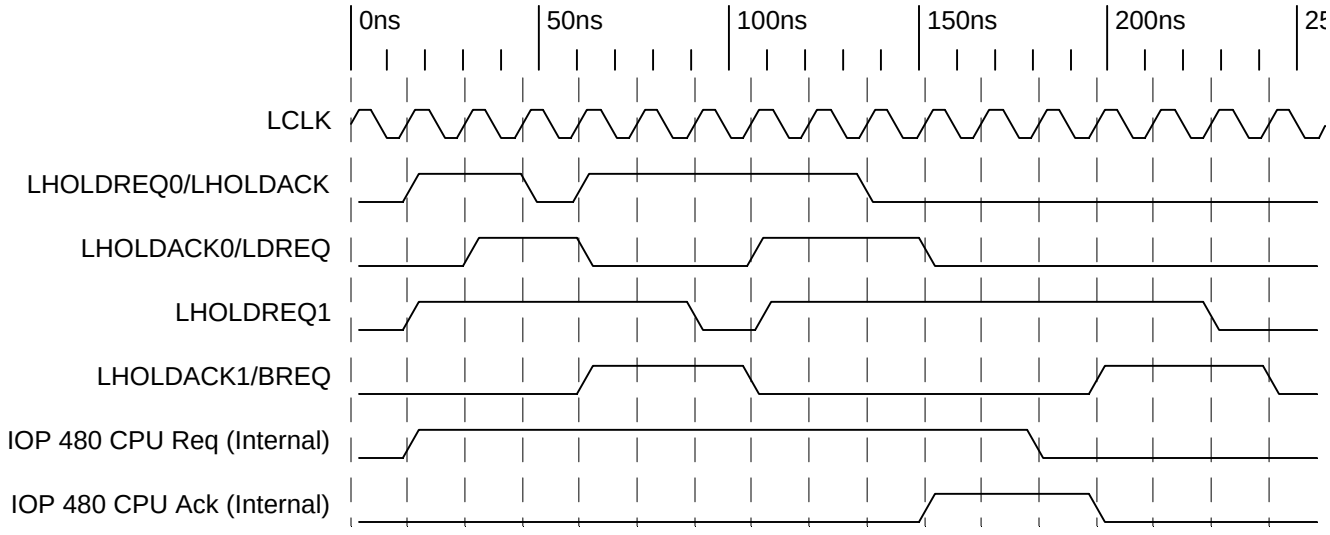
For example, to favor the DMA Channel 0 controller, select DMA 0/1 High-Priority mode (LARBR[3:1]= 010b) and set the DMA Channel Priority bits to DMA Channel 0 priority (LARBR[5:4]=01b).

The External Preempt Enable bit (LARBR[7]) and LHOLDACK1/BREQ output is used by an External Local Master to retain or release the Local Bus.

8.6 TIMING DIAGRAMS



Timing Diagram 8-1. Round-Robin Priority Arbitration (Three Active Requesters)



Timing Diagram 8-2. High-Priority Arbitration (LHOLDREQ0/LHOLDACK has Priority)

9 PCI BUS INTERNAL ARBITER

9.1 OVERVIEW

The IOP 480 can use an Internal or External PCI arbiter. The default is an External PCI arbiter. (Refer to PCICTL[16].)

The IOP 480 has an internal bus arbiter for the PCI Bus. The following sources can request use of the PCI Bus:

- IOP 480 internal PCI controller
- Three external PCI controllers

9.2 INITIALIZATION

By default, the internal PCI arbiter is disabled. It can only be enabled from the serial EEPROM, the IOP 480 CPU, or an external Local Master. The PCI Arbiter Enable bit (PCICTL[16]) should not be written from the PCI Bus.

9.3 PRIORITY MODE

The internal arbiter provides support for:

- Two operation modes—Priority or Round-Robin (PCICTL[17])
- Give grant on idle or busy (PCICTL[18])
- Park Option—Arbiter parks a grant on the current Master or on the IOP 480 when the PCI Bus is idle (PCICTL[19])

The internal arbiter can be disabled, enabling the IOP 480 to be used with an external arbiter. The selection is made in the PCI Arbiter Enable bit (PCICTL[16]).

9.3.1 Priority and Round-Robin Modes

The four sources of PCI Bus requests are serviced in Round-Robin priority, or the IOP 480 PCI controller can be given priority. The selection is made in the IOP 480 High Priority bit (PCICTL[17]). When Round-Robin is selected, each requester has equal priority. After a requester is granted the bus, the request is rotated to last priority, and the other active requests are rotated toward top priority. (Refer to Figure 9-2.)

If the IOP 480 PCI Controller is given priority, then the other requesters continue to participate in a lower level Round-Robin arbitration. The bus is alternately granted between the IOP 480 PCI controller and the winner of the Round-Robin arbitration. (Refer to Figure 9-1.)

GNT# is asserted in the PCI Idle state (FRAME# and IRDY# are both high) or in the Busy state. The internal arbiter can park a grant on the current Master or on the IOP 480 PCI controller (PCICTL[19]). If the GRANT_ON_IDLE bit is disabled in the PCI Bus Control register (PCICTL[18]=0), the current GNT# is de-asserted when FRAME# is asserted. When there are multiple requests, and a request is granted the bus, if FRAME# is not asserted on the next clock after GNT# is asserted, the arbiter passes the grant to the next request.

If there is a Reset when in Internal Arbiter mode, the IOP 480 PCI controller receives the first grant.

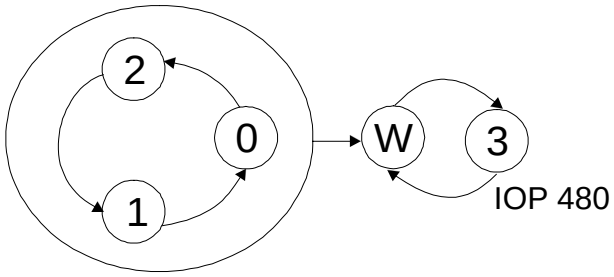


Figure 9-1. Priority Mode

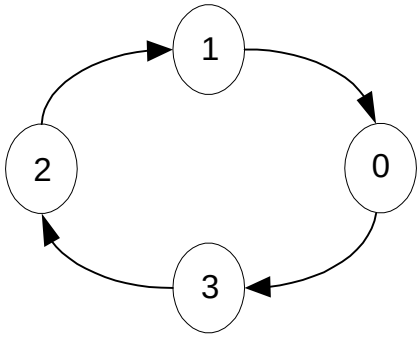


Figure 9-2. Round-Robin Mode

9.4 GRANT ON IDLE MODE

When the Early Grant Release bit is set to one (PCICTL[18]=1), the internal PCI arbiter gives a grant in the PCI Idle state. When the bit is set to zero (PCICTL[18]=0), a new grant is assigned to a new PCI Master as a current master asserts FRAME#.

9.5 PARK ON IOP 480 MODE

When the PCI Arbiter Parking on IOP 480 bit is set to one (PCICTL[19]=1), the internal PCI arbiter parks a grant on the IOP 480. When the bit is set to zero (PCICTL[19]=0), the internal PCI arbiter parks a grant on the current Master.

9.6 PERFORMANCE TUNING

Any Master that delays asserting FRAME# on the next clock after receiving a grant has the possibility of losing the grant. To minimize PCI Bus latency when the IOP 480 is a PCI Bus Master, use a combination of GRANT_ON_IDLE, PARKK_ON_IOP 480, and IOP 480 priority (PCICTL[19:17] =111b). After asserting a PCI request, in the worst case, the IOP 480 PCI controller waits only one external Master PCI latency.

10 RESET AND INITIALIZATION

10.1 OVERVIEW

The IOP 480 initialization procedure follows these steps:

1. Power-on.
2. Reset.
3. Serial EEPROM.
4. Boot the IOP 480 CPU.
5. Initialize the IOP 480 registers.
6. DRAM initialization.

10.2 POWER-ON

The typical order of events for power-on is as follows:

1. Power-on.
2. PCI and Local clocks start.
3. Master reset asserts (this can be PCI, if in Adapter mode, or Local, if in Host mode).
4. Master reset de-asserts.

Note: *It is possible for step 3 can occur prior to step 2.*

10.3 RESET

The IOP 480 reset behavior is determined by the HMODE pin. When this pin is *low*, the IOP 480 is in Adapter mode, and when *high*, in Host mode.

The HMODE pin cannot be dynamically changed.

Depending on HMODE, various reset sources have a different effect, as illustrated in the following table.

Table 10-1. Adapter and Host Mode Resets

Mode	Reset Type	Description
Adapter (Local Reset pin becomes an output only; PCI Reset pin is an input only; INTA# is an output only)	PCI	Resets all logic
	Software	Local Bus reset only
	Power Management	Resets all logic
	IOP 480 CPU for adapter	Same as reset pin
Host (Local Reset pin is an I/O only; PCI Reset pin becomes an output only; INTA# is an input only)	Local	Resets all logic
	Software	PCI Bus reset only
	IOP 480 CPU	Resets all logic
	IOP 480 CPU Power Management	No effect

10.3.1 Adapter Mode

10.3.1.1 PCI Reset

PCI RST# input causes all PCI Bus outputs to float, resets the entire IOP 480, and causes the Local RESET# pin to be asserted.

10.3.1.2 Software Reset

A Host on the PCI Bus can set the PCI Adapter Software Reset bit (DEVINIT[30]=1) to reset the IOP 480 and assert RESET# output. RESET# remains asserted for 62 clocks, or until the Software Reset bit is cleared, whichever occurs last. All Local Configuration registers are reset; however, the PCI Configuration registers, DMA registers, Shared Runtime registers, and the Local Init Status bit (DEVINIT[31]) are not reset. When the Software Reset bit (DEVINIT[30]) is set, the IOP 480 responds to PCI accesses, but not to Local Bus accesses. The IOP 480 remains in this reset condition until the PCI Host clears the bit. The serial EEPROM is reloaded if the Reload Configuration Registers bit is set (DEVINIT[29]=1).

Note: *The Local Bus cannot clear this reset bit because the Local Bus is in a reset state, even if a Local processor does not use RESET# to reset.*

10.3.1.3 Power Management Reset

When the power management reset is asserted (transition from D_3 to any other state), the IOP 480 resets as if a PCI reset was asserted. (Refer to Section 16, “Power Management.”)

10.3.1.4 Local RESET#

The Local RESET# output is driven when one of the following occurs:

- PCI RST# is asserted
- Software Reset bit is set
- IOP 480 CPU initiates an external reset
- Power management transitions from state D_3

10.3.1.5 IOP 480 CPU Chip Reset

A reset can be executed from the IOP 480 CPU processor via software or RISC watch. The chip reset causes the entire IOP 480 to be reset.

When the IOP 480 CPU executes a system-reset instruction, or detects reset input, it drives the Local RESET# pin for at least 62 clocks and then fetches its first instruction from 0xFFFFFFFC. Refer to the IOP 480 CPU description for more details on asserting chip reset.

10.3.2 Host Mode

10.3.2.1 PCI Reset

The PCI Bus RST# output is driven when the Local RESET# is asserted, the software reset bit is set, or the IOP 480 CPU initiates an external reset.

10.3.2.2 Local RESET#

When the Local RESET# pin is asserted by an external source, the Local Bus interface circuitry, the configuration registers, and the IOP 480 CPU are reset. The IOP 480 CPU drives the Local RESET# pin for 62 clocks after it detects a low input on this pin for two clocks.

10.3.2.3 Software Reset

When the Software Reset bit is set to one (DEVINIT[30]=1), the following occurs:

- PCI Master logic is held reset
- Local Bus logic is held reset
- IOP 480 CPU is held reset
- FIFOs are reset
- PCI RST# pin is asserted

The Configuration registers are not reset. A software reset can only be cleared from another Host on the Local Bus, and the IOP 480 remains in this reset condition until a Local Host clears the bit.

Note: *The PCI Bus cannot clear this reset bit because the PCI Bus is in a reset state.*

10.3.2.4 Power Management Reset

Power Management reset is not applicable for Host mode.

10.3.3 IOP 480 CPU Chip Reset

A reset can be executed from the IOP 480 CPU processor via software or RISC watch. The chip reset causes the entire IOP 480 to be reset.

When the IOP 480 CPU executes a system-reset instruction, or detects reset input, it drives the Local RESET# pin for at least 62 clocks and then fetches its first instruction from 0xFFFFFFFC. Refer to the IOP 480 CPU description for more details on asserting chip reset.

10.4 SERIAL EEPROM

After reset, the IOP 480 attempts to read the serial EEPROM to determine its presence. While the serial EEPROM is being checked, and during the serial EEPROM load, the following occurs:

- PCI accesses are Retried
- Local accesses are held off by not asserting READY#
- IOP 480 CPU is held in reset

An active start bit set to 0 indicates a serial EEPROM is present. The first Lword is then checked to verify whether the serial EEPROM is programmed. If the first Lword (32 bits) is all ones, a blank serial EEPROM is present. The IOP 480 reverts to default values. If the first Lword is not all ones, then the complete serial EEPROM load is performed.

If no serial EEPROM is present, EEDATA input should be set to 1. The 3.3V serial EEPROM clock (EESK) is derived from the PCI clock. The IOP 480 asserts the serial EEPROM clock by internally dividing the PCI clock by 132.

The IOP 480 uses the Fairchild 93CS66LEN serial EEPROM.

10.4.1 IOP 480 Initialization from the Local Bus

The internal IOP 480 CPU processor or a Local processor can access all IOP 480 internal registers through the base address specified in the Local Base Address for Configuration Register Access register (CFGBA). The access must be in a 32-bit-wide bus access.

10.4.2 Serial EEPROM Load

The registers listed in Table 10-3 are loaded from the serial EEPROM after a reset is de-asserted. The serial EEPROM is organized in words (16 bits). The IOP 480 first loads the Most Significant Word bits (MSW[31:16]), starting from the most significant bit ([31]). The IOP 480 then loads the Least Significant Word bits (LSW[15:0]), starting again from the most significant bit ([15]). For a description of each register, refer to Section 17, “Register Summary.”

10.4.3 Selectively Accessing the Serial EEPROM

The serial EEPROM can be read or written from the PCI or Local Buses. The Serial EEPROM Control register bits (DEVINIT[28:24]) control the IOP 480 pins that enable reading or writing of serial EEPROM data bits. (Refer to manufacturer’s data sheet for the particular serial EEPROM being used.) It is recommended that the Fairchild 93CS66LEN be used.

The serial EEPROM can also be read or written using the VPD function (refer to Section 15.3, “Serial EEPROM VPD Partitioning,” on page 15-1).

10.4.4 IOP 480 Initialization without a Serial EEPROM or with a Blank EEPROM

If a Boot Memory device other than a serial EEPROM is used to initialize the IOP 480, which uses the MA[16:13] as its addresses, then external pull-up resistors are required on the floated address lines during the boot-up process. These external pull-up resistors are required because, upon power-up, the data parity DP[3:0] signals of the MA[16:13]/DP[3:0] pins are disabled and the MA[16:13] signals are floated. The pulled-up MA[16:13]/DP[3:0] signals allow the IOP 480 to issue valid addresses for MA[16:13] and fetch instructions from the Boot Memory device. At boot-up, the IOP 480 must enable MA mode for the LCS0 memory region by writing a 0 to the Parity Checking bit (LCS0BRD[12]=0). The CPU instructions to accomplish this must be within address range 0xFFFF_8000 and FFFF_FFFC of the Boot Memory device because MA[16:13] are pulled high.

A blank EEPROM causes the IOP 480 to load its default values. The default is to enable the internal Local Bus arbiter. If an external Local Bus arbiter is being used in PCI Host mode, then a jumper is required to allow a blank serial EEPROM to be programmed. The jumper is used to put the IOP 480 in PCI Adapter mode (HMODE = 0, pulled low).

Table 10-2. Serial EEPROM Guidelines

Mode	Serial EEPROM	IOP 480 Register	Internal Local Arbiter	IOP 480 CPU
PCI Host	Blank	Default	Enabled	Boots
	None	Default	Enabled	Boots
	Programmed	Programmed	Programmed	Programmed
PCI Adapter	Blank	Default	Enabled	Held in Reset
	None	Default	Enabled	Boots
	Programmed	Programmed	Programmed	Programmed

Table 10-3. Serial EEPROM Load Registers

Serial EEPROM Offset	Description	Register Bits Affected
0h	MSW of Local Bus Control	LOCCTL[31:16]
2h	LSW of Local Bus Control	LOCCTL[15:0]
4h	MSW of Local Bus Timeout	LOCTMO[31:16]
6h	LSW of Local Bus Timeout	LOCTMO[15:0]
8h	MSW of Local Bus Timer	LOCTMR[31:16]
Ah	LSW of Local Bus Timer	LOCTMR[15:0]
Ch	MSW of Local/DMA Arbitration	LARBR[31:16]
Eh	LSW of Local/DMA Arbitration	LARBR[15:0]
10h	MSW of Big/Little Endian	BIGEND[31:16]
12h	LSW of Big/Little Endian	BIGEND[15:0]
14h	MSW of PCI Bus Control	PCICTL[31:16]
16h	LSW of PCI Bus Control	PCICTL[15:0]
18h	MSW of Range for PCI-to-Local Address Space 0	LAS0RR[31:16]
1Ah	LSW of Range for PCI-to-Local Address Space 0	LAS0RR[15:0]
1Ch	MSW of Local Base Address (Remap) PCI-to-Local Space 0	LAS0BA[31:16]
1Eh	LSW of Local Base Address (Remap) PCI-to-Local Space 0	LAS0BA[15:0]
20h	MSW of Range for PCI-to-Local Address Space 1	LAS1RR[31:16]
22h	LSW of Range for PCI-to-Local Address Space 1	LAS1RR[15:0]
24h	MSW of Local Base Address (Remap) PCI-to-Local Space 1	LAS1BA[31:16]
26h	LSW of Local Base Address (Remap) PCI-to-Local Space 1	LAS1BA[15:0]
28h	MSW of Range for PCI-to-Local Address Space 2	LAS2RR[31:16]
2Ah	LSW of Range for PCI-to-Local Address Space 2	LAS2RR[15:0]
2Ch	MSW of Local Base Address (Remap) PCI-to-Local Space 2	LAS2BA[31:16]
2Eh	LSW of Local Base Address (Remap) PCI-to-Local Space 2	LAS2BA[15:0]

Table 10-3. Serial EEPROM Load Registers (Continued)

Serial EEPROM Offset	Description	Register Bits Affected
30h	***Reserved***	MSW of 'h0b8 register
32h	***Reserved***	LSW of 'h0b8 register
34h	***Reserved***	MSW of 'h0bc register
36h	***Reserved***	LSW of 'h0bc register
38h	MSW of Range for Expansion ROM	EROMRR[31:16]
3Ah	LSW of Range for Expansion ROM	EROMRR[15:0]
3Ch	MSW of Local Base Address (Remap) PCI-to-Local EROM	EROMBA[31:16]
3Eh	LSW of Local Base Address (Remap) PCI-to-Local EROM	EROMBA[15:0]
40h	MSW of Range for Direct Master-to-PCI	DMRR[31:16]
42h	LSW of Range for Direct Master-to-PCI	DMRR[15:0]
44h	MSW of Local Base Address for Direct Master-to-PCI memory	DMLBAM[31:16]
46h	LSW of Local Base Address for Direct Master-to-PCI memory	DMLBAM[15:0]
48h	MSW of PCI Base Address (Remap) for Direct Master-to-PCI (lower 32 bits)	DMPBAM[31:16]
4Ah	LSW of PCI Base Address (Remap) for Direct Master-to-PCI (lower 32 bits)	DMPBAM[15:0]
4Ch	MSW of Direct Master Dual Address Cycle Upper Address	DMDAC[31:16]
4Eh	LSW of Direct Master Dual Address Cycle Upper Address	DMDAC[15:0]
50h	MSW of Local Base Address for Direct Master-to-PCI IO/CFG	DMLBAI[31:16]
52h	LSW of Local Base Address for Direct Master-to-PCI IO/CFG	DMLBAI[15:0]
54h	MSW of PCI Configuration Address for Direct Master-to-PCI IO/CFG	DMCFGA[31:16]
56h	LSW of PCI Configuration Address for Direct Master-to-PCI IO/CFG	DMCFGA[15:0]
58h	MSW of Configuration Base Address	CFGBA[31:16]
5Ah	LSW of Configuration Base Address	CFGBA[15:0]
5Ch	MSW of UART Base Address	UARTBA[31:16]
5Eh	LSW of UART Base Address	UARTBA[15:0]
60h	MSW of LCS0 Bus Region Descriptor	LCS0BRD[31:16]
62h	LSW of LCS0 Bus Region Descriptor	LCS0BRD[15:0]
64h	MSW of LCS0 Write Timing Register	LCS0WT[31:16]
66h	LSW of LCS0 Write Timing Register	LCS0WT[15:0]
68h	MSW of LCS0 Read Timing Register	LCS0RT[31:16]
6Ah	LSW of LCS0 Read Timing Register	LCS0RT[15:0]
6Ch	MSW of LCS0 Base Address	LCS0BASE[31:16]
6Eh	LSW of LCS0 Base Address	LCS0BASE[15:0]
70h	MSW of LCS0 Range	LCS0RANGE[31:16]
72h	LSW of LCS0 Range	LCS0RANGE[15:0]
74h	MSW of LCS1 Bus Region Descriptor	LCS1BRD[31:16]
76h	LSW of LCS1 Bus Region Descriptor	LCS1BRD[15:0]
78h	MSW of LCS1 Write Timing Register	LCS1WT[31:16]

Table 10-3. Serial EEPROM Load Registers (Continued)

Serial EEPROM Offset	Description	Register Bits Affected
7Ah	LSW of LCS1 Write Timing Register	LCS1WT[15:0]
7Ch	MSW of LCS1 Read Timing Register	LCS1RT[31:16]
7Eh	LSW of LCS1 Read Timing Register	LCS1RT[15:0]
80h	MSW of LCS1 Base Address	LCS1BASE[31:16]
82h	LSW of LCS1 Base Address	LCS1BASE[15:0]
84h	MSW of LCS1 Range	LCS1RANGE[31:16]
86h	LSW of LCS1 Range	LCS1RANGE[15:0]
88h	MSW of LCS2 Bus Region Descriptor	LCS2BRD[31:16]
8Ah	LSW of LCS2 Bus Region Descriptor	LCS2BRD[15:0]
8Ch	MSW of LCS2 Write Timing Register	LCS2WT[31:16]
8Eh	LSW of LCS2 Write Timing Register	LCS2WT[15:0]
90h	MSW of LCS2 Read Timing Register	LCS2RT[31:16]
92h	LSW of LCS2 Read Timing Register	LCS2RT[15:0]
94h	MSW of LCS2 Base Address	LCS2BASE[31:16]
96h	LSW of LCS2 Base Address	LCS2BASE[15:0]
98h	MSW of LCS2 Range	LCS2RANGE[31:16]
9Ah	LSW of LCS2 Range	LCS2RANGE[15:0]
9Ch	MSW of LCS3 Bus Region Descriptor	LCS3BRD[31:16]
9Eh	LSW of LCS3 Bus Region Descriptor	LCS3BRD[15:0]
A0h	MSW of LCS3 Write Timing Register	LCS3WT[31:16]
A2h	LSW of LCS3 Write Timing Register	LCS3WT[15:0]
A4h	MSW of LCS3 Read Timing Register	LCS3RT[31:16]
A6h	LSW of LCS3 Read Timing Register	LCS3RT[15:0]
A8h	MSW of LCS3 Base Address	LCS3BASE[31:16]
AAh	LSW of LCS3 Base Address	LCS3BASE[15:0]
ACh	MSW of LCS3 Range	LCS3RANGE[31:16]
AEh	LSW of LCS3 Range	LCS3RANGE[15:0]
B0h	MSW of DRAM Bus Region Descriptor	DRAMBRD[31:16]
B2h	LSW of DRAM Bus Region Descriptor	DRAMBRD[15:0]
B4h	MSW of DRAM Control	DRAMCTL[31:16]
B6h	LSW of DRAM Control	DRAMCTL[15:0]
B8h	MSW of DRAM Initialization	DRAMINIT[31:16]
BAh	LSW of DRAM Initialization	DRAMINIT[15:0]
BCh	MSW of DRAM Timing	DRAMTIM[31:16]
BEh	LSW of DRAM Timing	DRAMTIM[15:0]

Table 10-3. Serial EEPROM Load Registers (Continued)

Serial EEPROM Offset	Description	Register Bits Affected
C0h	MSW of DRAM Base Address	DRAMBASE[31:16]
C2h	LSW of DRAM Base Address	DRAMBASE[15:0]
C4h	MSW of DRAM Range	DRAMRANGE[31:16]
C6h	LSW of DRAM Range	DRAMRANGE[15:0]
C8h	MSW of Default Bus Region Descriptor	DFLTBRD[31:16]
CAh	LSW of Default Bus Region Descriptor	DFLTBRD[15:0]
CCh	MSW of Mailbox 0	MBOX0[31:16]
CEh	LSW of Mailbox 0	MBOX0[15:0]
D0h	MSW of Mailbox 1	MBOX1[31:16]
D2h	LSW of Mailbox 1	MBOX1[15:0]
D4h	Device ID	PCIDID[15:0]
D6h	Vendor ID	PCIVID[15:0]
D8h	Base Class Code, Subclass Code	PCIREV[31:16]
DAh	Register Level Programming Interface, Revision ID	PCIREV[15:0]
DCh	PCI Subsystem ID	PCISVID[31:16]
DEh	PCI Subsystem Vendor ID	PCISVID[15:0]
E0h	MSW of Capability List Pointer	CAP_PTR[31:16]
E2h	LSW of Capability List Pointer	CAP_PTR[15:0]
E4h	PCI Max_Lat, PCI Min_Gnt	PCIILR[31:16]
E6h	PCI Interrupt Pin, PCI Interrupt Line	PCIILR[15:0]
E8h	Power Management Capabilities	PMC[15:0]
EAh	Power Management Next Item Pointer, Power Management Capability ID	PMCAPID[15:0]
ECh	MSW of Power Management Data Scale Values	PMSCALE[31:6]
EEh	LSW of Power Management Data Scale Values	PMSCALE[15:0]
F0h	MSW of Power Consumed Values	PWRCON[31:16]
F2h	LSW of Power Consumed Values	PWRCON[15:0]
F4h	MSW of Power Dissipated Values	PWRDIS[31:16]
F6h	LSW of Power Dissipated Values	PWRDIS[15:0]
F8h	Hot Swap Control/Status	HSCSR[15:0]
FAh	Hot Swap Next Capability Pointer, Hot Swap Capability ID	HSNEXT[7:0] / HSCAPID[7:0][1]
FCh	MSW of Device Initialization	DEVINIT[31:16]
FEh	LSW of Device Initialization	DEVINIT[15:0]

1. Together, the two make a 16-bit serial EEPROM word.

10.5 IOP 480 CPU BOOT

This subsection describes the initial state of the IOP 480 CPU after a reset, and contains an example of the initialization code required to begin executing application code. Initialization of external system components or system-specific chip facilities may also need to be performed in addition to the basic initialization described in this section.

The IOP 480 CPU, after reset and the serial EEPROM check, begins fetching instructions from Local address 0xfffffffc. This address, by default, lies under the LCS0 Bus Region Descriptor control (LCS0BRD), and is, by default, an 8-bit-wide region. If this needs to be changed, then a serial EEPROM is needed to change the default values.

The IOP 480 CPU is a PowerPC RISC microprocessor, and should be initialized as such. In addition, there are internal IOP 480 registers outside of the CPU that may be initialized by the IOP 480 CPU. Refer to Section 4, “Direct Slave Operation,” and Section 5, “Direct Master Operation,” for more information regarding the initialization of these registers.

One particular register bit that ought to be set either by the serial EEPROM or by the IOP 480 CPU is DEVINIT[31], the Local Init. Status bit. Once DEVINIT[31] is set to 1, the IOP 480 stops Retrying all PCI accesses.

10.5.1 Processor State After Reset

After a reset, the contents of Special Purpose Registers (SPRs) control the initial processor state. Section 29, “IOP 480 CPU Register Summary,” contains descriptions of the registers.

In general, the contents of SPRs are undefined after a reset. Reset initializes the minimum number of SPR fields required for allow successful instruction fetching. System software fully configures the processor.

The MCI field of the Exception Syndrome Register (ESR) is cleared so that it can be determined if there has been a machine check during initialization, before Machine Check exceptions are enabled.

Two SPRs contain status on the type of reset that has occurred. The DBSR contains the most recent reset type. The TSR contains the most recent watchdog reset.

Table 10-4 lists the reset state of the initialized SPR fields.

10.5.2 IOP 480 CPU Initial Processor Sequencing

After any reset, the processor core fetches the word at address 0xFFFFFFFC and attempts to execute it. The instruction at 0xFFFFFFFC is typically a branch to initialization code. Because the processor is initialized in Big Endian mode, initialization code must be in Big Endian format until the SLER is configured otherwise. Unless the instruction at 0xFFFFFFFC is an unconditional branch, fetching can wrap to address 0x00000000 and attempt to execute the instruction at this location.

The system must provide non-volatile memory or memory initialized by some other mechanism, external to the processor, at location 0xFFFFFFFC and at the location of the initialization code, before a reset.

Table 10-4. Contents of Registers after Reset

Register	Field	Core Reset	Comment
MSR	APE	0	Auxiliary Processor exception disabled
	WE	0	Wait state disabled
	CE	0	Critical interrupts disabled
	EE	0	External interrupts disabled
	PR	0	Supervisor mode
	ME	0	Machine check interrupts disabled
	DE	0	Debug interrupts disabled
	IR	0	Instruction translation disabled
	LE	0	Little Endian disabled (Big Endian)
CDBCR	IOCM	Value on TIE_isocmMode	Instruction-side on-chip memory (OCM) mode
DBCR	EDM	0	Debug mode and events disabled
	RST	00	No reset action
DBSR	MRR	Most recent reset type	Most recent reset type
ESR	MCI	0	Machine check exception has not occurred
ICCR	S[0:31]	0x00000000	Instruction cache is disabled
PVR	FAM MEM CORE CHIP	0x002 0x 0x0 0xnnn	Processor Family
SGR	G[0:31]	0xFFFFFFFF	Storage is guarded
SKR	K[0:31]	0x00000000	Storage is not compressed
SLER	S[0:31]	0x00000000	Storage is Big Endian
TCR	WRC	00	Watchdog timer reset disabled
TSR	WRS	Copy of TCR[WRC]	If reset caused by watchdog timer
		Undefined	After Power-on
		Unchanged	If reset not caused by watchdog timer

10.5.3 Initialization Requirements

When any reset is performed, the processor is initialized to a minimum configuration to start executing initialization code. Initialization code is necessary to complete the processor and system configuration.

The initialization code example in this section performs the configuration tasks required to prepare the IOP 480 CPU to boot an operating system or run an application program.

Some portions of the initialization code work with system components that are beyond the scope of this data book.

Initialization code should perform the following tasks to configure the processor resources.

To improve instruction fetching performance, initialize the SGR appropriately for guarded or unguarded storage. Since all storage is initially guarded and speculative fetching is inhibited to guarded storage, reprogramming the SGR improves performance for unguarded regions.

Configure the following storage attribute control registers, if necessary:

- Initialize the SLER to configure the storage byte ordering
- Initialize the SKR to configure storage compression

Before executing instructions as cacheable:

- Invalidate the instruction cache
- Initialize the ICCR to configure instruction cacheability

Before using storage access instructions:

- Invalidate the data cache
- Initialize CDBCR to determine if a store miss results in a line fill (WOA)
- Initialize the DCWR to select copy-back or write-through caching
- Initialize the DCCR to configure data cacheability

Before allowing interrupts (synchronous or asynchronous):

- Initialize the EVPR to point to vector table
- Provide vector table with branches to interrupt handlers
- Initialize MSR(ILE) bit for Big Endian or PowerPC Little Endian mode for interrupt handlers

Before enabling asynchronous interrupts:

- Initialize timer facilities
- Initialize MSR to enable appropriate interrupts

Initialize other processor features, such as the MMU, cache line locking, debug, APU (if implemented), and trace.

Initialize non-processor resources:

- Initialize system memory as required by the operating system or application code
- Initialize off-chip system facilities
- Start the execution of operating system or application code

10.6 INITIALIZATION CODE EXAMPLE

The following initialization code example illustrates the steps that should be taken to initialize the processor before an operating system or user program begins execution. The example is presented in pseudo-code, function calls are named similarly to the IOP 480 CPU mnemonics where appropriate. Specific implementations may require different ordering of these sections to ensure proper operation. (Refer to Appendix C, “Real Code Example.”)

```
/* ———————————————————————————————————————————————*/
                /*IOP 480 CPU Initialization Pseudo Code*/
/* ———————————————————————————————————————————————*/
                @0xFFFFFFFC:/* initial instruction fetch from 0xFFFFFFFC*/
                ba(init_code);/* branch from initial address to initialization code*/

                @init_code:

  /* ———————————————————————————————————————————————*/
                /* Configure guarded attribute for performance and cacheability.*/
  /* ———————————————————————————————————————————————*/

                mtspr(SGR, guarded_attribute);

  /* ———————————————————————————————————————————————*/
                /* Configure endianness and compression.*/
  /* ———————————————————————————————————————————————*/

                mtspr(SLER, endianness);
                mtspr(SKR, compression_attribute);

  /* ———————————————————————————————————————— */
  /* Invalidate the instruction cache and enable cacheability*/
  /* ———————————————————————————————————————— */

                address=0;/* start at first line */
                for (line=0; line < n_lines; line++)/* I-cache has n_lines congruence classes
                */
                {
                iccci(address);/* invalidate congruence class */
                address += 16;/* point to the next congruence class */
                }
                mtspr(ICCR, i_cache_cacheability);/* enable I-cache*/
                isync;
```

```
/* ————————————————————————————————————————————————————————————*/
            /* Invalidate the data cache and enable cacheability */
/* ————————————————————————————————————————————————————————————*/

            address=0;/* start at first line */
            for (line=0; line <m_lines; line++)/* D-cache has m_lines congruence classes
            */
            {
            dccci(address);/* invalidate congruence class */
            address += 16;/* point to the next congruence class */
            }
            mtspr(CDBCR, store-miss_line-fill);
            mtspr(DCCR, copy-back_write-thru);
            mtspr(DCCR, d_cache_cacheability);/* enable D-cache*/
            isync;


/* ————————————————————————————————————————————————————————————*/
            /* Prepare system for synchronous interrupts*/
/* ————————————————————————————————————————————————————————————*/

            mtspr(EVPR, prefix_addr);/* initialize exception vector prefix */

            /* Initialize vector table and interrupt handlers if not already done */

            /* Initialize MSR(ILE) */

            mtmsr(MSR(ILE));

/* ————————————————————————————————————————————————————————————*/
            /* Prepare system for asynchronous interrupts*/
/* ————————————————————————————————————————————————————————————*/

            /* Initialize and configure timer facilities*/

            mtspr(PIT, 0);/* clear PIT so no PIT indication after TSR cleared*/
            mtspr(TSR, 0xFFFFFFFF);/* clear TSR*/
            mtspr(TCR, timer_enable);/* enable desired timers*/
            mtspr(TBLO, 0);/* reset time base low first to avoid ripple*/
            mtspr(TBHI, time_base_u);/* set time base, hi first to catch possible ripple*/
            mtspr(TBLO, time_base_l);/* set time base, low*/
            mtspr(PIT, pit_count);/* set desired PIT count*/
```

```
                /* Initialize the MSR */

                /* Exceptions must be enabled immediately after timer facilities to avoid
                missing a*/
                /* timer exception.*/
                /* */
                /* The MSR also controls privileged/user mode, translation, and the wait
                state.*/
                /* These must be initialized by the operating system or application code.*/
                /* If enabling translation, code must initialize the TLB.*/
/* ——————————————————————————————————————————————————————————*/

                mtmsr(machine_state);

/* ——————————————————————————————————————————————————————————*/
                /* Initialization of other processor facilities should be performed at this
                time*/
/* ——————————————————————————————————————————————————————————*/

/* ——————————————————————————————————————————————————————————*/
                /* Initialization of non-processor facilities should be performed at this time
                */
/* ——————————————————————————————————————————————————————————*/

/* ——————————————————————————————————————————————————————————*/
                /* Branch to operating system or application code can occur at this time*/
/* ——————————————————————————————————————————————————————————*/
```

10.7 DRAM INITIALIZATION

The DRAM Memory controller requires an initialization time of 13200 Local clocks (after reset) before DRAM can be accessed. Refer to Section 12.3, “DRAM,” for details.

10.7.1 IOP 480 Initialization from PCI Bus

A PCI Master can access the IOP 480 internal registers by way of one of the following:

- Memory access to the address specified in the PCIBAR0 register (PCI Base Address for Memory Accesses to Configuration Registers and Local Space 0)
- PCI Configuration access

11 INTERRUPTS

11.1 OVERVIEW

There are multiple sources of interrupts for the IOP 480, but there are only two interrupt pins, PCI INTA# and Local INTO. Each pin has a global Enable or Disable bit. In addition, each source for an interrupt has a particular enable or disable, as well as a particular Status bit indicating which interrupt source is active.

11.2 PCI INTERRUPTS

An IOP 480 PCI Interrupt (INTA#) can be asserted by one of the following:

- Local-to-PCI Doorbell Interrupt
- Local Interrupt Input (INTI)
- Master/Target Abort Interrupt
- DMA PCI Interrupts
- Messaging Outbound Post Queue Interrupt
- 256 Consecutive PCI Retries

INTA#, or individual sources of an interrupt, can be enabled or disabled with the IOP 480 Interrupt Enable register (PINTENB). The register provides the interrupt status of each interrupt source.

The IOP 480 PCI Bus interrupt is a level output. Disabling an Interrupt Enable bit or clearing the cause(s) of the interrupt can clear an interrupt.

In Adapter mode, INTA# is an output only. In Host mode, INTA# is an input only.

Notes: *For the above interrupts to activate INTA#, the PCI Interrupt Enable bit, PINTENB[0], must be enabled.*

Convention: PINTENB[0] = Active high, ~PINTENB[0] = Active low

11.2.1 Local-to-PCI Doorbell Interrupt

A Local Bus Master can assert a PCI Bus interrupt by writing to the Local-to-PCI Doorbell register bit(s) (L2PDBELL[31:0]). The PCI Host processor can read the PCI Doorbell Interrupt Active bit to determine whether a PCI Doorbell interrupt is pending (PINTSTAT[11]), and if so, read the IOP 480 Local-to-PCI Doorbell register.

Each bit in the Local-to-PCI Doorbell register is individually controlled. The Local Bus can only set bits in the Local-to-PCI Doorbell register. From Local Bus, writing 1 to any bit position sets that bit and writing 0 has no effect. Bits in the Local-to-PCI Doorbell register can only be cleared from the PCI Bus. From the PCI Bus, writing 1 to any bit position clears that bit and writing 0 has no effect.

Interrupts remain set as long as any Local-to-PCI Doorbell register bits are set and the PCI Doorbell Interrupt Enable bit (PINTENB[11]) is set.

To prevent race conditions from occurring when the PCI Bus is accessing the Local-to-PCI Doorbell register (or any Configuration register), the IOP 480 automatically de-asserts READY# output to prevent Local Bus Configuration accesses.

11.2.2 Local Interrupt Input (INTI)

Asserting the Local Interrupt input INTI can assert a PCI Bus interrupt. The PCI Host processor can read the IOP 480 Status register (PINTSTAT) to determine whether an interrupt is pending as a result of INTI being asserted (PINTSTAT[12]).

The interrupt remains asserted as long as INTI input is asserted and the Local Interrupt input is enabled. The PCI Host processor can take adapter-specific action to cause the Local Bus to release INTI.

If the PCI Interrupt Enable bit is cleared (PINTENB[12]=0), the PCI interrupt (INTA#) is de-asserted; however, the Local interrupts (INTI) and the Status bit remain active.

11.2.3 Master/Target Abort Interrupt

The IOP 480 sets the Received Master/Target Abort bit (PINTSTAT[13]=1) when it detects a Master or Target abort. This status bit causes PCI INTA# to be asserted if interrupts are enabled.

The Interrupt remains set as long as the Received Master Abort or Target Abort bit remains set and the Master/Target Abort interrupt is enabled. The Received Master/Target Abort Interrupt bit (PINTSTAT[13]=0) can be cleared by writing a 1 to the appropriate bit in the PCI Status register (PCISR[13] for Master aborts or PCISR[12] for Target aborts).

Use PCI Type 0 Configuration or Local accesses to clear the Received Master/Target Abort Interrupt bit (PINTSTAT[13]=0).

The Interrupt Control/Status register bits (PINTSTAT[18:16] are latched at the time of a Master or Target Abort interrupt. These bits provide information as to which device was the Master when the abort occurred. The IOP 480 updates these bits only when an abort occurs.

11.2.4 DMA PCI Interrupts

Each of the three DMA channels can generate a PCI INTA# interrupt. A DMA "Done" interrupt occurs at the end of a single DMA transfer if it is not in Scatter/Gather mode, or at the completion of a group of Scatter/Gather DMA transfers. The "Done" interrupt is enabled by setting C0MODE[10]/C1MODE[10], and/or C2MODE[13].

DMA Channels 0 and 1 can also be programmed to generate an interrupt after a transfer corresponding to a Scatter/Gather descriptor is completed. C0DESCPTR[2]/C1DESCPTR[2] (loaded from descriptor memory) determines whether to assert an interrupt at the end of the transfer for the current descriptor. The PCI Host processor can then read the PINTSTAT register to determine which DMA channel has a pending interrupt. The "Done" Status bit in the DMA Control/Status register can be used to determine if the interrupt is a done interrupt or is the result of a transfer for a descriptor in Scatter/Gather mode.

A DMA interrupt is cleared by writing a 1 to bit 3 of the corresponding DMA Control/Status register.

11.2.5 Messaging Unit Outbound Post Queue Interrupt

When the Outbound Post Queue is not empty, a PCI INTA# interrupt can be generated. This interrupt is enabled by clearing the Outbound Post Queue Interrupt Mask bit (OPQIM[3]=0). The interrupt is cleared when the Outbound Post Queue become empty. Also, if the Outbound Option Interrupt Mask bit is enabled (QSR[8]=0), this interrupt is asserted if the HOSTOUTIDX and IOPOUTIDX registers are not equal.

11.2.6 256 Consecutive PCI Retries

The IOP 480 can be configured to respond to 256 consecutive PCI retries as a Target abort by setting the Treat 256 PCI Retries as Abort bit to 1 (PCICTL[21]=1). When this function is enabled and 256 consecutive retries are detected, the IOP 480 generates a Target abort and sets the Target Abort Generated bit (PINTSTAT[19]=1). Writing a 1 to the Target Abort bit (PCICSR[11]=1) clears the bit (PINTSTAT[19]=0).

11.2.7 PCI Interrupt Output (INTA#)

Table 11-1 describes the PCI Interrupts (INTA#).

Table 11-1. PCI Interrupts (INTA#)

Interrupt Source Description	Register Enable Bit	Invoke Interrupt	Clear Interrupt	Interrupt Status
Local-to-PCI Doorbell Interrupt	PINTENB[11]	L2PDBELL[31:0] (write a 1 to any bit set from the Local Bus)	L2PDBELL[31:0] (write a 1 to bit set from the PCI Bus)	PINTSTAT[11]
Local Interrupt Input (INTI)	PINTENB[12]	INTI	~INTI	PINTSTAT[12]
Master/Target Abort Interrupt	PINTENB[13]	PCISR[12] / PCISR[13]	PCISR[12] / PCISR[13]	PINTSTAT[13]
DMA Ch 0 Terminal Count Reached DMA Ch 1 Terminal Count Reached	PINTENB[8] PINTENB[9]	C0DESCPTR[2] C1DESCPTR[2]	C0CSR[3] C1CSR[3]	PINTSTAT[8] PINTSTAT[9]
DMA Ch 0 Done DMA Ch 1 Done DMA Ch 2 Done	PINTENB[8] PINTENB[9] PINTENB[10]	C0MODE[10] C1MODE[10] C2MODE[12]	C0CSR[3] C1CSR[3] C2CSR[3]	PINTSTAT[8] PINTSTAT[9] PINTSTAT[10]
Messaging Outbound Post Queue Interrupt	~OPQIM[3] ~QSR[8]	Outbound Post Queue not empty HOSTOUTIDX != IOPOUTIDX	Outbound Post Queue becomes empty	OPQIS[3]
256 Consecutive PCI Retries	PCICTL[21]	PCISR[11]	PCISR[11]	PINTSTAT[19]

11.3 PCI SYSTEM ERROR OUTPUT (SERR#)

The IOP 480 generates an SERR# pulse if parity checking is enabled in the PCI Command register and it detects an address parity error or the Generate SERR# bit (PINTSTAT[0]) is changed from a 0 to a 1. The SERR# output can be enabled or disabled through the SERR# Enable bit (PCICR[8]).

11.4 LOCAL INTERRUPTS

A Local interrupt can be asserted when any one of the following events occur:

- Mailbox Interrupt
- PCI ENUM# Interrupt
- PCI PME# Interrupt
- PCI INTA# Interrupt
- PCI SERR# Interrupt
- Power Management Interrupt
- BIST Interrupt
- PCI-to-Local Doorbell Interrupt
- DMA Channel 0, 1, or 2 Interrupt
- INTI Interrupt Input Pin asserted
- Local Bus Parity Error
- Serial Port Interrupts
- Local Bus Timeout
- PCI Bus Parity Error
- Messaging Unit Outbound Free Queue Overflow Interrupt
- Messaging Unit Inbound Post Queue Not Empty Interrupt
- PCI Master or Target Abort
- CINT Critical Interrupt
- Refresh Interrupt

Individual sources of the Local interrupt can be enabled or disabled through the Local Interrupt Output Enable bit (LINTENB[0]=1). The Local Interrupt Status register (LINTSTAT) provides status for each source of the interrupt. The Local interrupt is connected to the following:

- EXTINT interrupt input of the internal IOP 480 CPU
- INTO Local Interrupt Output pin
- PCI INTA# output interrupt

The polarity of the INTO pin is controlled by LINTENB[17].

11.4.1 Mailbox Interrupt

A Local interrupt can be asserted when the PCI Host writes to one of the mailbox registers. Each Mailbox register has an associated Interrupt Enable bit in the Local Interrupt Enable register (LINTENB[31:24]). A Local Mailbox interrupt remains asserted until the mailbox is read by the internal or Local CPU. The status of the mailbox interrupts is determined by reading the Local Interrupt Status register (LINTSTAT[31:24]). All byte enables must be active to generate an interrupt.

11.4.2 PCI Enumerate Input (ENUM#)

A PCI Hot Swap Enumerate Interrupt (ENUM#) input can be serviced when the IOP 480 is operating in PCI Host mode. If ENUM# is asserted, a Local interrupt can be generated if enabled (LINTENB[16, 0]=high). The Local interrupt remains asserted as long as ENUM# is asserted and the interrupt is enabled. When operating in Host mode, the ENUM# output should be disabled by writing a zero to the ENUM# Interrupt Mask bit (HSCSR[1]=0).

11.4.3 PCI Power Management Event Input (PME#)

A PCI Power Management Event Interrupt (PME#) input can be serviced when the IOP 480 is operating in PCI Host mode. If PME# is asserted, a Local interrupt can be generated if enabled (LINTENB[15, 0]=high). The Local interrupt remains asserted as long as PME# is asserted and the interrupt is enabled. When operating in Host mode, the PME# output should be disabled by writing a zero to the PME_En bit (PMCSR[8]=0).

11.4.4 PCI Interrupt Input (INTA#)

A PCI Interrupt (INTA#) input can be serviced when the IOP 480 is operating in PCI Host mode. If INTA# is asserted, a Local interrupt can be generated if enabled (LINTENB[14, 0]=high). The Local interrupt remains asserted as long as INTA# is asserted and the interrupt is enabled. When operating in Host mode, the INTA# output should be disabled by writing a zero to the PCI Interrupt Output Enable bit (PINTENB[0]=0).

11.4.5 PCI System Error Input (SERR#)

A PCI System Error (SERR#) input can be serviced when the IOP 480 is operating in PCI Host mode. If SERR# is asserted, a Local interrupt can be generated if enabled LINTENB[20, 0]=high). The Local interrupt remains asserted as long as SERR# is asserted or LINTSTAT[20] is not clear and the interrupt is enabled. To clear the SERR# interrupt, write a 1 to LINTSTAT[20]. When operating in Host mode, the SERR# output should be disabled by writing a zero to the SERR# Enable bit (PCICR[8]=0).

11.4.6 Power Management Interrupt

A change in the Power Management State can cause a Local interrupt if enabled LINTENB[13, 0]=high). The Power Management State is changed when the host writes to the Power State bits (PMCSR[1:0]). The status of this interrupt can be read from the Power Management Interrupt bit (LINTSTAT[13]), and is cleared by writing a 1 to this bit [LINTSTAT[13]=1).

11.4.7 Built-In Self Test Interrupt (BIST)

A PCI Bus Master can generate a Local Bus interrupt by performing a PCI Type 0 Configuration write to the BIST Start bit (PCIBISTR[6]). The Local processor can then read the BIST Interrupt bit (LINTSTAT[12]) to determine that a BIST interrupt is pending. The Local interrupt remains asserted as long as PCIBISTR[6] is set and the BIST interrupt is enabled. The Local Bus should reset the bit when BIST is complete. The PCI Host software may fail the device if the bit is not reset after two seconds.

11.4.8 PCI-to-Local Doorbell Interrupt

A PCI Bus Master can generate a Local Bus interrupt by writing to the PCI-to-Local Doorbell register. The Local processor can then read the Local Doorbell Interrupt bit (LINTSTAT[11]) to determine that a doorbell interrupt is pending. It can then read the PCI-to-Local Doorbell (P2LDBELL) register. Each bit in the PCI-to-Local Doorbell register is individually controlled. Bits in the Doorbell register can only be set by the PCI Bus. From the PCI Bus, writing a 1 to any bit position sets that bit and writing a 0 to a bit position has no effect. Bits in the PCI-to-Local Doorbell register can only be cleared from the Local Bus. From the Local Bus, writing a 1 to any bit position clears that bit and writing a 0 to a bit position has no effect.

If the Local Bus cannot clear the Doorbell Interrupt, do not use the PCI-to-Local Doorbell register.

The interrupt remains asserted as long any of the PCI-to-Local Doorbell registers bits are set and the Local Doorbell interrupt is enabled.

To prevent race conditions from occurring when the Local Bus is accessing the P2LDBELL register (or any other configuration register), the IOP 480 automatically issues a Retry to the PCI Bus.

11.4.9 DMA Local Interrupts

Each of the three DMA channels can generate a Local interrupt. A DMA "Done" interrupt occurs at the end of a single DMA transfer if not in Scatter/Gather mode, or at the completion of a group of Scatter/Gather DMA transfers. The "Done" interrupt is enabled by setting the Done Interrupt Enable bit(s) (C0MODE[10]/ C1MODE[10] and/or C2MODE[13]).

DMA Channels 0 and 1 can also be programmed to generate an interrupt after a transfer corresponding to a Scatter/Gather descriptor is completed. The Interrupt after Terminal Count bit(s) (C0DESCPTR[2] and/or C1DESCPTR[2]), loaded from descriptor memory, determines whether to assert an interrupt at the end of the transfer for the current descriptor. The Local Host processor can then read the LINTSTAT register to determine which DMA channel has a pending interrupt. The "Done" Status bit in the DMA Control/Status Register can be used to determine if the interrupt is a done interrupt or is the result of a transfer for a descriptor in Scatter/Gather mode.

A DMA interrupt is cleared by writing a 1 to bit 3 of the corresponding DMA Control/Status register.

11.4.10 Local Interrupt Input

Asserting the INTI pin can cause a Local interrupt. This interrupt is enabled by setting the INTI Interrupt Enable bit (LINTENB[7]), and status is reported on the INTI Interrupt bit (LINTSTAT[7]). The interrupt remains asserted as long as the INTI pin is asserted and the Local Interrupt input is enabled. The polarity of the INTI pin is controlled by LINTENB[18].

11.4.11 Local Bus Parity Error

A data parity error detected on the Local data bus can cause a Local interrupt. This interrupt is enabled by setting the Local Parity Error Interrupt Enable bit (LINTENB[6]), and status is reported on the Local Bus Parity Error bit (LINTSTAT[6]). The interrupt is cleared by writing a 1 to LINTSTAT[6]. Parity error checking is selectively enabled in each of the six Bus Region Descriptor registers in the Memory controller.

11.4.12 Serial Port Interrupts

The internal serial port can generate two interrupts which can cause a Local interrupt. These interrupts are enabled by setting the Serial Port Interrupt 2 and 1 Enable bits (LINTENB[5:4], respectively), and status is reported on LINTSTAT[5:4].

11.4.13 Local Bus Timeout

A Timeout detected on the Local Data bus can cause a Local interrupt to be asserted. This interrupt is enabled by setting the Local Timeout Interrupt Enable bit (LINTENB[3]), and status is reported on the Local Bus Timeout Interrupt bit (LINTSTAT[3]). The interrupt is cleared by writing a 1 to LINTSTAT[3]. The Local Bus Timeout Timer is loaded with the value in LOCTMR[14:0] at the beginning of every Local access. If the counter decrements to zero before a READY# is detected, a timeout is generated.

11.4.14 PCI Bus Parity Errors

PCI Bus parity errors can be enabled to cause a Local interrupt. If either the Master Data Parity Error Detected bit (PCISR[8]) or the Parity Error Detected bit (PCISR[15]) are set in the PCI Status register, then a Local interrupt is generated if the PCI Bus Parity Error Interrupt Enable bit (LINTENB[2]) is set. Status for this interrupt is reported on the PCI Error Interrupt bit (LINTSTAT[2]). The interrupt is cleared by writing a 1 to the Master Data Parity Error Detected bit (PCISR[8]) or the Parity Error Detected bit (PCISR[15]).

11.4.15 Messaging Unit Outbound Free Queue Overflow Interrupt

An Outbound Free Queue overflow can cause a Local interrupt. This interrupt is enabled by setting the PCI Parity Error Interrupt Enable bit (LINTENB[2]) and clearing the Outbound Free Queue Overflow Interrupt Mask bit (QSR[6]). Status is reported on LINTSTAT[2]. The interrupt is cleared by writing a 1 to the Outbound Free Queue Overflow Interrupt bit (QSR[7]). Because this interrupt shares the same Enable and Status bit as PCI Parity Errors, QSR[7:4] and PCISR[15, 8] can be read to determine the source of the interrupt.

11.4.16 Messaging Unit Inbound Post Queue Interrupt

When the Inbound Post Queue is not empty, a Local interrupt can be generated. This interrupt is enabled by writing a zero to the Inbound Post Queue Interrupt Mask bit (QSR[4]). The interrupt is cleared when the Inbound Post Queue become empty. Because this interrupt shares the same Enable and Status bit as PCI Parity Errors, QSR[7:4] and PCISR[15, 8] can be read to determine the source of the interrupt.

11.4.17 Master/Target Abort Interrupt

The IOP 480 sets the Master Abort or Target Abort Status bit in the PCI Configuration register upon detection of a Master or Target abort. A Target abort is also assumed if the IOP 480 detects 256 Retry responses during a Direct Master cycle. These status bits cause a Local interrupt to be asserted if the PCI Abort Interrupt Enable bit is enabled (LINTENB[1]=1). The interrupt status can be read from the PCI Abort Interrupt bit (LINTSTAT[1]), and remains asserted as long as the Master or Target Abort bits remain set in the PCI Configuration Status register and Master/Target Abort interrupt is enabled.

The Master or Target Abort Detected bits (PINTSTAT[18:16], respectively) are latched at the time of a Target Abort interrupt or a Master Abort interrupt. They provide information as to who was Master when an abort occurred. They are updated only when an abort occurs. (Refer to Figure 11-1.)

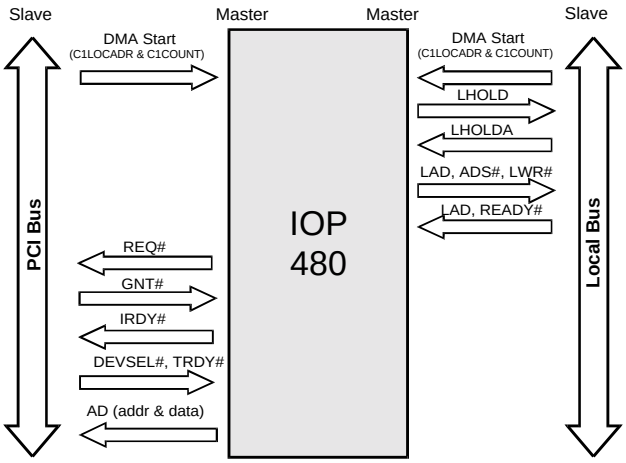


Figure 11-1. DMA, Local-to-PCI Bus

Note: *The figure represents a sequence of bus cycles.*

11.4.18 CINT Critical Interrupt

The CINT Critical Interrupt input can generate a Local Interrupt and is enabled by setting the CINT/USER2 Pin Select bit to 0 (LOCCTL[8]=0). The polarity of CINT is controlled by (LINENB[19]).

11.4.19 Refresh Interrupt

SDRAM refreshes can generate Local interrupts if two refresh requests occur without a grant occurring between the two requests. This interrupt is enabled by setting the Refresh Local Interrupt Enable bit to 1 (LINTENB[21]=1). The Refresh Local Interrupt Status can be read in the Local Interrupt Status register (LINTSTAT[21]).

11.4.20 Local Interrupt Output (INTO)

Table 11-2. Local Interrupts (INTO)

Interrupt Source Description	Register Enable Bit	Invoke Interrupt	Clear Interrupt	Interrupt Status
Abort Interrupt	LINTENB[1]	PCISR[12] / PCISR[13]	PCISR[12] / PCISR[13] (depends on which is set)	LINTSTAT[1]
BIST Interrupt	LINTENB[12]	PCIBISTR[6] by PCI	~PCIBISTR[6] by Local	LINTSTAT[12]
DMA Ch 0 Terminal Count Reached DMA Ch 1 Terminal Count Reached	LINTENB[8] LINTENB[9]	C0DESCPTR[2] C1DESCPTR[2]	C0CSR[3] C1CSR[3]	LINTSTAT[8] LINTSTAT[9]
DMA Ch 0 Done DMA Ch 1 Done DMA Ch 2 Done	LINTENB[8] LINTENB[9] LINTENB[10]	C0MODE[10] C1MODE[10] C2MODE[13]	C0CSR[3] C1CSR[3] C2CSR[3]	LINTSTAT[8] LINTSTAT[9] LINTSTAT[10]
INTI Interrupt, Input Pin Asserted	LINTENB[7]	INTI with LINTENB[18] (polarity)	LINTSTAT[7]	LINTSTAT[7]
Local Bus Parity Error	LINTENB[6]	LINTSTAT[6]	LINTSTAT[6]	LINTSTAT[6]
Local Bus Timeout	LINTENB[3]	LINTSTAT[6]	LINTSTAT[3]	LINTSTAT[3]
Mailbox Interrupt (0-7)	LINTENB[24:31]	PCI Host writes to the Mailbox register	Local Master reads the Mailbox	LINTSTAT[24:31]
Messaging Unit Inbound Post Queue Interrupt	~QSR[4]	Inbound Post Queue becomes non-empty	Inbound Post Queue becomes empty	LINTSTAT[2]
Messaging Unit Outbound Free Queue Overflow Interrupt	LINTENB[2] ~QSR[6]	QSR[7]	QSR[7] (write a message frame address 1)	LINTSTAT[2]
PCI Bus Parity Error	LINTENB[2]	PCISR[8] / PCISR[15]	PCISR[8] / PCISR[15] (depends on which is set)	LINTSTAT[2]
PCI ENUM# Interrupt	LINTENB[16]	ENUM# or LINTSTAT[16]	LINTSTAT[16]	LINTSTAT[16]
PCI INTA# Interrupt	LINTENB[14]	INTA# or LINTSTAT[14]	LINTSTAT[14]	LINTSTAT[14]
PCI PME Interrupt	LINTENB[15]	PME# or LINTSTAT[15]	LINTSTAT[15]	LINTSTAT[15]
PCI SERR# Interrupt	LINTENB[20]	SERR# or LINTSTAT[20]	LINTSTAT[20]	LINTSTAT[20]
PCI-to-Local Doorbell Interrupt	LINTENB[11]	P2LDBELL[31:0] (write a 1 to any bit from the PCI Bus)	P2LDBELL[31:0] (write a 1 to bit set from the Local Bus)	LINTSTAT[11]
Power Management Interrupt	LINTENB[13]	Change power state to PMCSR[1:0]	LINTSTAT[13]	LINTSTAT[13]
Refresh Interrupt	LINTENB[21]	LINTSTAT[21]	LINTSTAT[21]	LINTSTAT[21]
Serial Port Interrupt 1	LINTENB[4] (L_UART+'20h[31] / L_UART+'20h[4])	L_UART[5] / L_UART[6]	Write a 1 to the bit that invokes the interrupt (L_UART[6:5])	LINTSTAT[4]
Serial Port Interrupt 2	LINTENB[5] (L_UART+'1Ch[3] / L_UART+'1Ch[4])	L_UART[0] / L_UART[1] / L_UART[2] / L_UART[3] / L_UART[4]	Write a 1 to the bit that invokes the interrupt (L_UART[0-4])	LINTSTAT[5]

Notes: *For the above local interrupts to activate INTO, the Local Interrupt Enable bit, LINTENB[0], must be enabled.*

Convention: LINTENB[0] = Active high, ~LINTENB[0] = Active low

INTO polarity is determined by LINTENB[17]:
LINTENB[17] = 1 ⟹ INTO active high
LINTENB[17] = 0 ⟹ INTO active low

11.5 DOORBELL REGISTERS

The IOP 480 has two 32-bit doorbell interrupt/status registers. One is assigned to the PCI Bus interface. The other is assigned to the Local Bus interface.

A Local processor can assert a PCI Bus interrupt by writing any number other than all zeroes to the Local-to-PCI Doorbell register bits (L2PDBELL[31:0]=1 and PINTENB[0]=1).

A PCI Host can assert a Local Bus interrupt by writing any number other than all zeroes to the PCI-to-Local Doorbell register bits (P2LDBELL[31:0]=1 and LINTENB[0]=1).

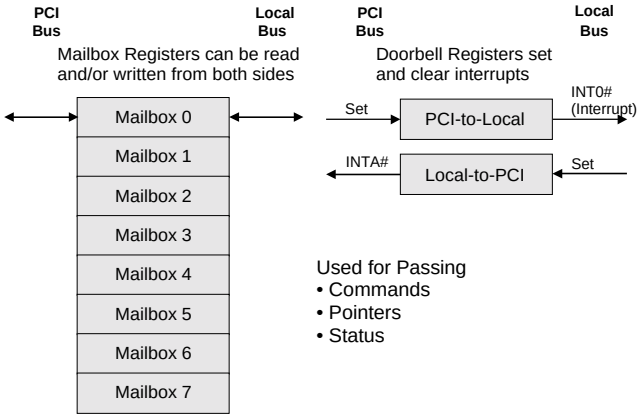


Figure 11-2. Mailbox and Doorbell Message Passing

11.6 MAILBOX REGISTERS

The IOP 480 has eight 32-bit Mailbox registers that can be written to and read from both buses. These registers can be used to pass command and status information directly between the PCI Bus and Local Bus devices.

A Local interrupt can be asserted, if the Local Interrupt Output Enable and Mailbox Interrupt Enable bits are enabled (LINTENB[0] and LINTENB[31:24], respectively), when the PCI Host writes to one of the eight Mailbox registers. All byte enables must be active to generate an interrupt.

11.7 IOP 480 EXCEPTIONS, INTERRUPTS, AND TIMERS

Exceptions in the IOP 480 CPU are generated by signals from external peripherals, instructions, the internal timer facility, debug events, and error conditions. Two external interrupt signals are provided in the IOP 480 CPU, one critical and one non-critical. Both external interrupts are maskable.

This chapter begins by defining the terminology of exceptions and interrupts in Section 11.7.1, “Interrupts and Exceptions.”

Table 11-4 on page 11-13 lists the exceptions which are handled by the in the order of exception vector offsets. Detailed descriptions of each exception follow, in the same order. Table 11-4 on page 11-13 also provides an index to the descriptions.

Several registers support exception handling and control. Section 11.7.3, “General Exception Handling Registers,” on page 11-13, describes the general exception handling registers:

- Data Exception Address Register (DEAR)
- Exception Syndrome Register (ESR)
- Exception Vector Prefix Register (EVPR)
- Machine State Register (MSR)
- Save/Restore Registers (SRR0–SRR3)

Critical and non-critical external interrupt signals are enabled by the MSR.

Section 11.7.5, “Machine Check Exceptions,” on page 11-20, describes machine checks.

Section 11.7.18, “Timer Facilities,” on page 11-28, describes the timer facilities of the IOP 480 CPU, including the time base and the registers Time Base Low Register (TBLO), Time Base Low User-mode (TBLU), Time Base High Register (TBHI), and Time Base High User-mode (TBHU). This section also describes the timer-related registers: the Timer Status Register (TSR) and the Timer Control Register (TCR), and the Programmable Interval Timer (PIT) register.

11.7.1 Interrupts and Exceptions

An *interrupt* is the action in which the processor saves its old context (MSR and instruction pointer) and begins execution at a pre-determined interrupt-handler address, with a modified MSR. *Exceptions* are events which, if enabled, cause the processor to take an interrupt.

11.7.1.1 Architectural Definitions and Behavior

Precise interrupts are those for which the instruction pointer saved by the interrupt must be either the address of the excepting instruction or the address of the next sequential instruction. *Imprecise* interrupts are those for which it is possible (not required, just possible) for the saved instruction pointer to be something else, possibly prohibiting guaranteed software recovery.

Note: *"Precise" and "imprecise" are defined assuming that the interrupts are unmasked (enabled to occur) when the associated exception occurs. Consider an exception that would cause a precise interrupt, were the interrupt enabled at the time of the exception, but that occurs while the interrupt is masked. Some exceptions of this type can cause the interrupt to occur later, immediately upon its enabling. In such a case, the interrupt is not considered precise with respect to the enabling instruction, but imprecise ("delayed precise") with respect to the cause of the exception.*

Asynchronous interrupts are caused by events which are independent of instruction execution. All asynchronous interrupts are precise, and the following rules apply:

1. All instructions prior to the one whose address is reported to the exception handling routine (in the save/restore register) have completed execution. However, some storage accesses generated by these preceding instructions may not have completed.

2. No subsequent instruction has begun execution, including the instruction whose address is reported to the exception handling routine.

3. The instruction having its address reported to the exception handler may appear not to have begun execution, or may have partially completed.

Synchronous interrupts are caused directly by the execution (or attempted execution) of instructions. Synchronous interrupts can be either precise or imprecise.

For synchronous precise interrupts, the following rules apply:

1. The save/restore register addresses either the instruction causing the exception or the next sequential instruction. Which instruction is addressed is determined by the interrupt type and status bits.

2. All instructions preceding the instruction causing the exception have completed execution. However, some storage accesses generated by these preceding instructions may not have completed.

3. The instruction causing the exception may appear not to have begun execution (except for causing the exception), may have partially completed, or may have completed, depending on the interrupt type.

4. No subsequent instruction has begun execution.

The IOP 480 CPU does not implement any imprecise interrupts. Refer to The IBM PowerPC Embedded environment for an architectural description of imprecise interrupts.

Machine check interrupts are a special case typically caused by some kind of hardware or storage subsystem failure, or by an attempt to access an invalid address. A machine check can be indirectly caused by the execution of an instruction, but not recognized or reported until long after the processor has executed past the instruction that caused the machine check. As such, machine check interrupts cannot properly be thought as synchronous, nor as precise or imprecise. However, in the IOP 480 CPU, machine checks are handled precisely as critical interrupts (refer to Section 11.7.2, "Critical and Non-Critical Exceptions," on page 11-10). For machine checks, the following general rules apply:

1. No instruction following the one whose address is reported to the machine check handler in the save/restore register has begun execution.

2. The instruction whose address is reported to the machine check handler in the save/restore register, and all previous instructions, may or may not have completed successfully. All previous instructions that would ever complete have completed, within the context existing before the machine check interrupt. No further interrupt (other than possible additional machine checks) can occur as a result of those instructions.

11.7.1.2 IOP 480 CPU Implementation Behavior

All exceptions are handled precisely. Precise handling implies that the address of the excepting instruction (for synchronous exceptions other than the system call exception), or the address of the next instruction to be executed (asynchronous exceptions and the system call exception), is passed to an exception handling routine. Precise handling also implies that all instructions that precede the instruction whose address is reported to the exception-handling routine have executed and that no subsequent instruction has begun execution. The specific instruction whose address is reported may not have begun execution or may have partially completed, as specified for each precise exception type.

Synchronous precise exceptions include most debug exceptions, program exceptions, instruction and data storage exceptions, TLB miss exceptions, system call, and alignment exceptions.

Asynchronous precise exceptions include the critical interrupt exception, external interrupts, internal peripherals, internal timer facility exceptions, and some debug events.

The machine check exceptions, which are neither synchronous or asynchronous, are handled precisely.

The synchronism of instruction-side machine checks (errors that occur while attempting to fetch an instruction from external memory) require further explanation. Fetch requests to cacheable memory that miss in the instruction cache (ICU) cause an instruction cache line fill (four Lwords). If any Lwords in the fetched line are associated with an error, an exception occurs upon attempted execution and the cache line is invalidated.

It is improper to declare an exception when an erroneous Lword is passed to the fetcher; the address could be the result of an incorrect speculative access. It is quite likely that no attempt will be made to execute an instruction from the erroneous address. An n instruction-side machine check exception occurs only when execution is attempted. If the exception occurs, execution is suppressed, SRR2 contains the erroneous address, and the ESR indicates that instruction-side machine check occurred. Although such an exception is clearly asynchronous to the erroneous memory access, it is handled synchronously with respect to the attempted execution from the erroneous address.

Except for machine checks, all IOP 480 CPU exceptions are handled precisely:

- The address of the excepting instruction (for synchronous exceptions, other than the system call exception) or the address of the next sequential instruction (for asynchronous exceptions and the system call exception) is passed to the exception-handling routine.
- All instructions that precede the instruction whose address is reported to the exception-handling routine have completed execution and that no subsequent instruction has begun execution. The specific instruction whose address is reported might not have begun execution or might have partially completed, as specified for each exception type.

11.7.1.3 Exception-Handling Priorities

In the IOP 480 CPU, only one exception is handled at a time. Multiple simultaneous exceptions are handled in the priority order shown in Table 11-3 on page 11-12.

11.7.2 Critical and Non-Critical Exceptions

The IOP 480 CPU processes exceptions as non-critical and critical. Six exceptions are defined as *non-critical*: program exception, system call exception, alignment exception, an active external interrupt input, fixed interval timer (FIT) exception, and PIT exception. Five exceptions are defined as *critical*: machine check exceptions (instruction- and data-side), debug exceptions (any of the three types), exceptions caused by an active critical interrupt input, and the first timeout from the watchdog timer.

When a *non-critical* exception is taken, Save/Restore Register 0 (SRR0) is written with the address of the excepting instruction (most synchronous exceptions) or the next sequential instruction to be processed (asynchronous exceptions and system call).

If the IOP 480 CPU was executing a multi-cycle instruction (load/store, multiply, divide, or cache operation), the instruction is terminated and its address is written in SRR0. When load instructions terminate, the addressing registers are not updated. This ensures that the instructions can be restarted; if the addressing registers were in the range of registers

to be loaded, this would be an invalid form in any event. Some target registers of a load instruction may have been written by the time of the exception; when the instruction restarts, the registers are simply written again. Similarly, some of the target memory of a store instruction may have been written, and is again written when the instruction restarts.

Save/Restore Register 1 (SRR1) is written with the contents of the MSR; the MSR is then updated to reflect the new machine context. The new MSR contents take effect beginning with the first instruction of the exception handling routine.

Exception handling routine instructions are fetched at an address determined by the exception type. The address of the exception handling routine is formed by concatenating the 16 high-order bits of the EVPR and the exception vector offset. (A user must initialize the EVPR contents at power-up using an **mtspr** instruction.)

Table 11-4 on page 11-13 shows the exception vector offsets for the exception types. Note that there may be multiple sources of the same exception type; exceptions of the same type are mapped to the same exception vector, regardless of source. In such cases, the exception handling routine must examine status registers to determine the exact source of the exception.

At the end of the exception handling routine, execution of an **rfi** instruction forces the contents of SRR0 and SRR1 to be written to the program counter and the MSR, respectively. Execution then begins at the address in the program counter.

Critical exceptions are processed similarly. When a critical exception is taken, Save/Restore Register 2 (SRR2) and Save/Restore Register 3 (SRR3) hold the next sequential address to be processed when returning from the exception and the contents of the MSR, respectively. At the end of the critical exception handling routine, execution of an **rfci** instruction writes the contents of SRR2 and SRR3 into the program counter and the MSR, respectively.

Table 11-3. Exception-Handling Priorities

Priority	Exception Type	Critical or Non-Critical	Causing Conditions
1	Machine check—data	Critical	External bus error during data-side access.
2	Debug—IAC	Critical	IAC debug event while in internal debug mode.
3	Machine check—instruction	Critical	Attempted execution of instruction for which an external bus error occurred during fetch.
4	Debug—UDE, EXC	Critical	UDE or EXC debug event while in internal debug mode.
5	Critical interrupt input	Critical	Active level on the critical interrupt input.
6	Watchdog timer—first timeout	Critical	Posting of an enabled first timeout of the watchdog timer in the TSR.
7	Instruction TLB Miss	Non-critical	Attempted execution of an instruction at an address and process ID for which a valid matching entry was not found in the TLB.
8	Instruction storage exception—ZPR[Zn]=00	Non-critical	Instruction translation is active, execution access to the translated address is not permitted because ZPR[Zn]=00 in user mode, and an attempt is made to execute the instruction.
9	Instruction storage exception—TLB_entry[EX]=0	Non-critical	Instruction translation is active, execution access to the translated address is not permitted because TLB_entry[EX]=0, and an attempt is made to execute the instruction.
	Instruction storage exception—TLB_entry[G]=1	Non-critical	Instruction translation is active, the page is marked Guarded, and an attempt is made to execute the instruction.
10	Program	Non-critical	Attempted execution of illegal instructions, TRAP instruction, or privileged instruction in problem state.
	System call	Non-critical	Execution of the **sc** instruction.
11	Data TLB miss	Non-critical	Valid matching entry for the effective address and process ID of an attempted data access is not found in the TLB.
12	Data storage exception—ZPR[Zn]=00	Non-critical	Data translation is active and data-side access to the translated address is not permitted because ZPR[Zn]=00 in user mode.
13	Data storage exception—TLB_entry[WR]=0	Non-critical	Data translation is active and write access to the translated address is not permitted because TLB_entry[WR]=0.
14	Data storage exception—Cache line locking	Non-critical	MSR[PR]=1, and: **dcbt** and CDBCR[DUXE]=1; **dcbz** and CDBCR[DLXE]=1); **icbi** and CDBCR[IUXE]=1.
15	Alignment	Non-critical	Misaligned data accesses in PowerPC Little Endian mode; **dcbz** to non-cacheable address or write-through storage; Any string or multiple instruction in PowerPC Little Endian mode; Non-Lword-aligned **dcread**, **lwarx**, and **stwcx** as described in Table 11-12 on page 11-24.
16	Debug—IC, BT, TIE, DAC	Critical	IC, BT, TIE, or DAC debug event while in internal debug mode.
17	External interrupt input	Non-critical	Interrupts from the ExtInt pin.
18	Fixed Interval Timer (FIT)	Non-critical	Posting of an enabled FIT interrupt in the TSR.
19	Programmable Interval Timer (PIT)	Non-critical	Posting of an enabled PIT interrupt in the TSR.

Table 11-4. Exception Vector Offsets

Offset	Exception Type	Exception Class	Category	Page
0x0100	Critical interrupt	Asynchronous precise	Critical	11-19
0x0200	Machine check—data	—	Critical	11-21
	Machine check—instruction	—	Critical	11-20
0x0300	Data storage exception— MSR[DR]=1 and ZPR[Zn]=0 or TLB_entry[WR]=0	Synchronous precise	Non-critical	11-21
0x0400	Instruction storage exception	Synchronous precise	Non-critical	11-22
0x0500	External interrupt	Asynchronous precise	Non-critical	11-23
0x0600	Alignment	Synchronous precise	Non-critical	11-24
0x0700	Program	Synchronous precise	Non-critical	11-24
0x0C00	System Call	Synchronous precise	Non-critical	11-25
0x1000	PIT	Asynchronous precise	Non-critical	11-26
0x1010	FIT	Asynchronous precise	Non-critical	11-26
0x1020	Watchdog timer	Asynchronous precise	Critical	11-27
0x1100	Data TLB miss	Synchronous precise	Non-critical	11-27
0x1200	Instruction TLB miss	Synchronous precise	Non-critical	11-27
0x2000	Debug exception— IC, BT, TIE, IA1, DR1, DW1	Synchronous precise	Critical	11-28
	Debug exception—UDE, EXC	Asynchronous precise	Critical	

11.7.3 General Exception Handling Registers

The general exception handling registers are the Machine State Register (MSR), SRR0–SRR3, the Exception Vector Prefix Register (EVPR), the Exception Syndrome Register (ESR), and the Data Exception Address Register (DEAR).

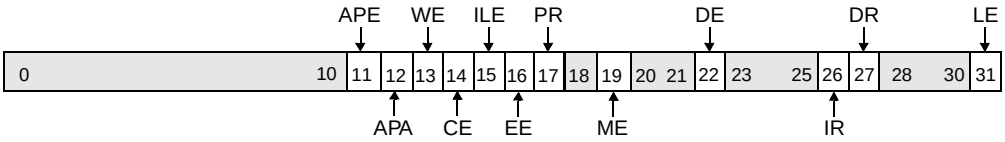
11.7.3.1 Machine State Register (MSR)

The MSR is a 32-bit register that holds the current context of IOP 480 CPU. When a non-critical interrupt is taken, the MSR contents are written to SRR1; when a critical interrupt is taken, the MSR contents are written to SRR3. When an **rfi** or **rfci** instruction executes, the contents of the MSR are read from SRR1 or SRR3, respectively.

The MSR contents can be read into general purpose registers (GPRs) using an **mfmsr** instruction. The contents of a GPR can be written to the MSR using an **mtmsr** instruction. The MSR[EE] bit may be set/cleared atomically using the **wrtee** or **wrteei** instructions.

Register 11-1 illustrates the MSR bits.

Register 11-1. Machine State Register (MSR)



0:10		***Reserved***	
11	APE	Auxiliary Processor Exception Enable 0 Auxiliary processor exception disabled. 1 Auxiliary processor exception enabled.	
12	APA	Auxiliary Processor Available 0 Auxiliary processor not available. 1 Auxiliary processor available.	
13	WE	Wait State Enable 0 The processor is not in the wait state. 1 The processor is in the wait state.	If MSR[WE]=1, the processor remains in the wait state until an exception is taken, a reset occurs, or an external debug tool clears WE.
14	CE	Critical Interrupt Enable 0 Critical interrupts are disabled. 1 Critical interrupts are enabled.	Controls the critical interrupt input and watchdog timer first timeout interrupts.
15	ILE	Interrupt Little Endian 0 Interrupt handlers execute in Big Endian mode. 1 Interrupt handlers execute in PowerPC Little Endian mode.	Copied to MSR(LE) when an interrupt is taken.
16	EE	External Interrupt Enable 0 Asynchronous exceptions are disabled. 1 Asynchronous exceptions are enabled.	Controls the non-critical external interrupt input, Programmable Interval Timer, and Fixed Interval Timer interrupts.
17	PR	Problem State 0 Supervisor State (all instructions allowed). 1 Problem State (some instructions not allowed).	
18		***Reserved***	
19	ME	Machine Check Enable 0 Machine check exceptions are disabled. 1 Machine check exceptions are enabled.	
20:21		***Reserved***	
22	DE	Debug Exception Enable 0 Debug exceptions are disabled. 1 Debug exceptions are enabled.	
23:25		***Reserved***	
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.	If TIE_cpuMmuEn is 0, reading or writing this bit has no effect.
27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.	If TIE_cpuMmuEn is 0, reading or writing this bit has no effect.
28:30		***Reserved***	
31	LE	Little Endian 0 Processor executes in Big Endian mode. 1 Processor executes in PowerPC Little Endian mode.	

11.7.3.2 Save/Restore Registers 0 and 1 (SRR0–SRR1)

SRR0 and SRR1 are 32-bit registers that hold the interrupted machine context when a non-critical interrupt is processed. On interrupt, SRR0 is set to the current or next instruction address and the contents of the MSR are written to SRR1. When an **rfi** instruction is executed at the end of the interrupt handler, the program counter and the MSR are restored from SRR0 and SRR1, respectively.

The contents of SRR0 and SRR1 can be written into GPRs using the **mfspr** instruction. The contents of GPRs can be written to SRR0 and SRR1 using the **mtspr** instruction.

Register 11-2 illustrates the SRR0 bits.

Register 11-3 illustrates the SRR1 bits.

11.7.3.3 Save/Restore Registers 2 and 3 (SRR2–SRR3)

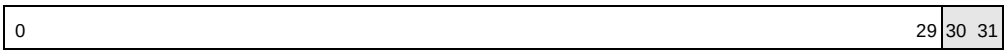
SRR2 and SRR3 are 32-bit registers that hold the interrupted machine context when a critical interrupt is processed. On interrupt, SRR2 is set to the current or next instruction address and the contents of the MSR are written to SRR3. When an **rfci** instruction is executed at the end of the interrupt handler, the program counter and the MSR are restored from SRR2 and SRR3, respectively.

The contents of SRR2 and SRR3 can be written to GPRs using the **mfspr** instruction. The contents of GPRs can be written to SRR2 and SRR3 using the **mtspr** instruction.

Register 11-4 illustrates the SRR2 bits.

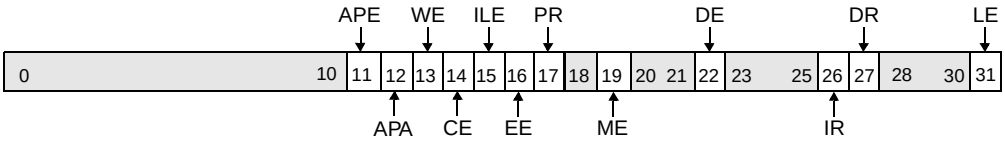
Register 11-5 illustrates the SRR3 bits.

Register 11-2. Save/Restore Register 0 (SRR0)



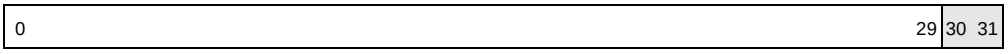
0:29		SRR0 receives an instruction address when a non-critical interrupt is taken; the Program Counter is restored from SRR0 when **rfi** executes.	
30:31		***Reserved***	

Register 11-3. Save/Restore Register 1 (SRR1)



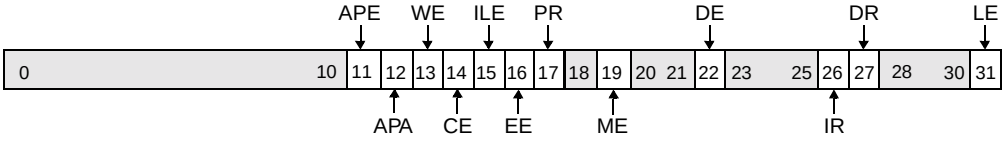
0:31		SRR1 receives a copy of the MSR when a critical interrupt is taken; the MSR is restored from SRR1 when **rfi** executes.

Register 11-4. Save/Restore Register 2 (SRR2)



0:29		SRR2 receives an instruction address when a critical interrupt is taken; the Program Counter is restored from SRR2 when **rfci** executes.
30:31		***Reserved***

Register 11-5. Save/Restore Register 3 (SRR3)



0:31		SRR3 receives a copy of the MSR when a critical interrupt is taken; the MSR is restored from SRR3 when **rfci** executes.

11.7.3.4 Exception Vector Prefix Register (EVPR)

The EVPR is a 32-bit register whose high-order 16 bits contain the prefix for the address of an exception processing routines. The 16-bit exception vector offsets (shown in Table 11-4, “Exception Vector Offsets,” on page 11-13) are concatenated to the right of the high-order 16 bits of the EVPR to form the 32-bit address of the exception processing routine.

The contents of the EVPR can be written to a GPR using the **mfspr** instruction. The contents of a GPR can be written to EVPR using the **mtspr** instruction.

Register 11-6 illustrates the EVPR bits.

11.7.3.5 Exception Syndrome Register (ESR)

The ESR is a 32-bit register whose bits help to specify the exact cause of various synchronous exceptions. These exceptions include instruction and data side machine checks, data storage exceptions, program exceptions, instruction storage exceptions, and data TLB miss exceptions.

Section 11.7.5.1, “Instruction Machine Check Handling,” on page 11-20, describes instruction machine checks. Section 11.7.6, “Data Storage Exceptions,” on page 11-21, describes data storage exceptions. Section 11.7.10, “Program Exceptions,” on page 11-24, describes program exceptions.

Although exception-handling routines are not required to reset the ESR, it is recommended that instruction machine check handlers reset the ESR; Section 11.7.5.1, “Instruction Machine Check Handling,” on page 11-20 describes why such resets are recommended.

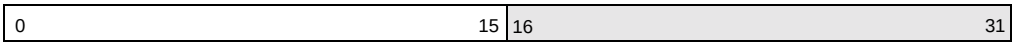
The contents of the ESR can be written to a GPR using the **mfspr** instruction. The contents of a GPR can be written to the ESR using the **mtspr** instruction

Register 11-7 illustrates the ESR bits.

In general, ESR bits are set to indicate the kind of precise interrupt that occurred; other bits are cleared. The machine check—instruction (ESR[MCI]) bit behaves differently, however. Because instruction-side machine checks can occur without an interrupt being taken (if MSR[ME]=0), this bit is set even when other ESR-setting exceptions (data storage, program, DTLB-miss) are occurring. Thus, data storage and program exceptions leave ESR[MCI] alone, but clear the data storage and program exception bits that are not associated with the specific data storage or program exception that is occurring. Enabled instruction-side machine checks (MSR[ME]=1) set ESR[MCI] and clear the data storage and program exception bits.

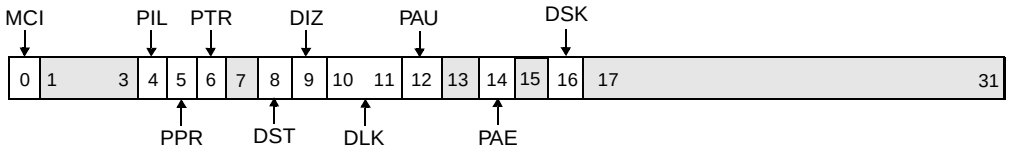
If a machine check—instruction exception occurs but is disabled (MSR[ME]=0), it sets ESR[MCI] but leaves the data storage and program exception bits alone. If a machine check—instruction exception occurs while MSR[ME]=0, *and* the instruction upon which the machine check—instruction exception is occurring also is some other kind of ESR-setting instruction (program, DTLB-miss, or Instruction Storage exception), ESR[MCI] is set to indicate that a machine check—instruction exception occurred; the other ESR bits are set or cleared to indicate the other exception. These scenarios are summarized in Table 11-5 on page 11-19.

Register 11-6. Exception Vector Prefix Register (EVPR)



0:15		Exception Vector Prefix	
16:31		***Reserved***	

Register 11-7. Exception Syndrome Register (ESR)



0	MCI	Machine check—instruction 0 Instruction machine check did not occur. 1 Instruction machine check occurred.	
1:3		***Reserved***	
4	PIL	Program exception—illegal 0 Illegal Instruction error did not occur. 1 Illegal Instruction error occurred.	
5	PPR	Program exception—privileged 0 Privileged instruction error did not occur. 1 Privileged instruction error occurred.	
6	PTR	Program exception—trap 0 Trap with successful compare did not occur. 1 Trap with successful compare occurred.	
7		***Reserved***	
8	DST	Data storage exception—store fault 0 Excepting instruction was not a store. 1 Excepting instruction was a store (includes **dcbi**, **dcbz**, and **dccci**).	
9	DIZ	Data/instruction storage exception—zone fault 0 Excepting condition was not a zone fault. 1 Excepting condition was a zone fault.	
10:11	DLK	Data Storage exception— lock fault 00 No lock exception 01 **dcbf** unlock exception 10 **icbi** unlock exception 11 **dcbz** lock-out exception	
12	PAU	Program exception—auxiliary processor unavailable 0 Auxiliary processor unavailable exception did not occur. 1 Auxiliary processor unavailable exception occurred.	
13		***Reserved***	
14	PAE	Program exception—auxiliary processor enabled 0 Auxiliary processor enabled exception did not occur. 1 Auxiliary processor enabled exception occurred.	
15		***Reserved***	
16	DSK	Data storage exception—compressed 0 Excepting instruction did not access compressed storage. 1 Excepting instruction accessed compressed storage.	
17:31		***Reserved***	

Table 11-5. ESR Alteration by Various Exceptions

Scenario	$ESR_{4:6}$	$ESR_{8:9}$
Program exception w/o MCI	Set to type	Cleared
Enabled MCI	Cleared	Cleared
Disabled MCI, no others	Unchanged	Unchanged
Disabled MCI+ program exception	Set to type	Cleared

11.7.3.6 Data Exception Address Register (DEAR)

The DEAR is a 32-bit register that contains the address of the access for which one of the following synchronous precise errors occurred—alignment error, data TLB miss, or data storage exception.

The contents of the DEAR can be written to a GPR using the **mfspr** instruction. The contents of a GPR can be written to the DEAR using the **mtspr** instruction.

Register 11-8 illustrates the DEAR bits.

11.7.4 Critical Interrupt Exception

An external source requests a critical interrupt by driving the critical interrupt input active. The critical exception is recognized if enabled by MSR[CE].

MSR[CE] also enables the watchdog timer first-timeout exception. However, the watchdog interrupt has a different exception vector than the critical pin interrupt. See Section 11.7.14, “Watchdog Timer Exception,” on page 11-27.

After detecting a critical interrupt, if no synchronous precise exceptions are outstanding, the IOP 480 CPU immediately takes the critical interrupt exception and writes the address of the next instruction to be executed in SRR2. Simultaneously, the contents of the MSR are saved in SRR3. MSR[CE] is reset to 0 to prevent another critical interrupt or the watchdog timer first timeout exception from interrupting the critical interrupt exception handler before SRR2 and SRR3 get saved. MSR[DE] is reset to 0 to disable debug exceptions during the critical interrupt exception handler.

The MSR is also written with the values shown in Table 11-6. The high-order 16 bits of the program counter are then loaded with the contents of the EVPR and the low-order 16 bits of the program counter are loaded with 0x0100. Exception processing begins at the address in the program counter.

Inside the exception handling routine, after the contents of SRR2/SRR3 are saved, critical interrupts can be enabled again by setting MSR[CE]=1.

Executing an **rfci** instruction restores the program counter from SRR2 and the MSR from SRR3, and execution resumes at the address in the program counter.

Table 11-6. Register Settings during Critical Interrupt Exceptions

SRR2	Written with the address of the next instruction to be executed
SRR3	Written with the contents of the MSR
MSR	APE, APA, WE, PR, CE, EE, DE ← 0 ME ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0100

Register 11-8. Data Exception Address Register (DEAR)

0:31		Address of Data Error (synchronous)	

11.7.5 Machine Check Exceptions

When an external bus error occurs on an instruction fetch, and execution of that instruction is subsequently attempted, a machine check—instruction exception occurs.

When an external bus error occurs while attempting data accesses, a machine check—data exception occurs.

When an instruction-side machine check interrupt occurs, the IOP 480 CPU stores the address of the excepting instruction in SRR2. When a data-side machine check occurs, the IOP 480 CPU stores the address of the next sequential instruction in SRR2. Simultaneously, for all machine check exceptions, the contents of the MSR are loaded into SRR3.

The MSR Machine Check Enable bit (MSR[ME]) is reset to 0 to disable another machine check from interrupting the machine check exception handling routine. The other MSR bits are loaded with the values shown in Table 11-7 and Table 11-8. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0200. Exception processing begins at the new address in the program counter.

Executing an **rfci** instruction restores the program counter from SRR2 and the MSR from SRR3, and execution resumes at the address in the program counter.

11.7.5.1 Instruction Machine Check Handling

When a machine check occurs on an instruction fetch, *and execution of that instruction is subsequently attempted*, a machine check—instruction exception occurs. If enabled by MSR[ME], the processor reports the machine check—instruction exception by vectoring to the machine check handler (EVPR[0:15] || 0x0200), setting ESR[MCI]. Note that only a bus error can cause a machine check—instruction exception. Taking the vector automatically clears MSR[ME] and the other MSR fields.

It is improper to declare a machine check—instruction exception when the instruction is fetched, because the address is possibly the result of an incorrect speculation by the fetcher. It is quite likely that no attempt will be made to execute an instruction from the erroneous address. The exception occurs only if execution of the instruction is subsequently attempted.

When a machine check occurs on an instruction fetch, the erroneous instruction is never written to the instruction cache unit (ICU). Fetch requests to cacheable memory that miss in the ICU cause an instruction cache line fill (four Lwords). If any Lwords in the fetched line are associated with an error, an exception occurs upon attempted execution and the cache line is invalidated. If any Lword in the line is in error, the cache line is invalidated after the line fill.

ESR[MCI] is set, even if MSR[ME]=0. This means that if a machine check—instruction exception occurs while running in code in which MSR[ME] is disabled, the machine check—instruction exception is recorded in the ESR, but no interrupt occurs. Software running with MSR[ME] disabled can sample ESR[MCI] to determine whether at least one machine check—instruction exception occurred during the disabled execution.

After MSR[ME] is enabled again, if a new machine check—instruction exception occurs, it is recorded in ESR[MCI] and the machine check—instruction exception handler is invoked. However, enabling MSR[ME] again does *not* cause a Machine Check interrupt to occur simply due to the presence of ESR[MCI] indicating that a machine check—instruction exception occurred while MSR[ME] was disabled. The machine check—instruction exception must occur while MSR[ME] is enabled for the machine check interrupt to be taken. Software should, in general, clear the ESR bits before returning from a machine check interrupt, to avoid any ambiguity when handling subsequent machine check exceptions.

Table 11-7. Register Settings during Machine Check—Instruction Exceptions

SRR2	Written with the address that caused the machine check.
SRR3	Written with the contents of the MSR
MSR	APE, APA, WE, PR, CE, EE, ME, DE ← 0 DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0200
ESR	MCI ← 1

11.7.5.2 Data Machine Check Handling

When a machine check occurs on an data access, a machine check—data exception occurs. The handling of machine check—data exceptions is implementation-specific.

11.7.6 Data Storage Exceptions

The data storage exception occurs on a cache line locking error or is generated when the desired access to the effective address is not permitted for any of the following reasons:

1. In Problem State with data translation enabled

- A zone fault, which is any user-mode storage access (data load, store, **icbi**, **dcbz**, **dcbst**, or **dcbf**) with an effective address with (ZPR field)=00. (**dcbt** and **dcbtst** no-op in this situation, rather than cause an exception. The instructions **dcbi**, **dccci**, **icbt**, and **iccci**, being privileged, cannot cause zone fault Data Storage exceptions.)
  - Data store or **dcbz** to an effective address with the WR bit clear and (ZPR field) ≠ 11. (The privileged instructions **dcbi** and **dccci** are treated as “stores”, but cause privileged Program exceptions, rather than Data Storage exceptions.)

2. In Supervisor State with data translation enabled:

- Data store, **dcbi**, **dcbz**, or **dccci** to an effective address with the WR bit clear and (ZPR field) other than 11 or 10.

Note: *The **icbi**, **icbt**, and **iccci** instructions are treated as loads from the addressed byte with respect to address translation and protection. Instruction cache ops use MSR[DR], not MSR[IR], to determine translation of their operands. Instruction Storage Exceptions and Instruction-side TLB Miss Exceptions are associated with the fetching of instructions, not with the execution of instructions. Data Storage Exceptions and Data TLB Miss Exceptions are associated with the execution of instruction cache ops.*

Table 11-8. Register Settings during Machine Check—Data Exceptions

SRR2	Written with the address of the next sequential instruction.
SRR3	Written with the contents of the MSR
MSR	APE, APA, WE, PR, CE, EE, ME, DE ← 0 DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0200
ESR	MCI ← 0

When a data storage exception is detected, the IOP 480 CPU suppresses the instruction causing the exception and writes the instruction address in SRR0. The Data Exception Address Register (DEAR) is loaded with the data address that caused the access violation. Exception Syndrome Register (ESR) bits are loaded as shown in Table 11-9 to provide further information about the error. The current contents of the MSR are loaded into SRR1, and MSR bits are then loaded with the values shown in Table 11-9.

The high-order 16 bits of the program counter are then loaded with the contents of the EVPR and the low-order 16 bits of the program counter are loaded with 0x0300. Exception processing begins at the new address in the program counter. Executing the return from interrupt instruction (**rfi**) restores the contents of the program counter and the MSR from SRR0 and SRR1, respectively. The IOP 480 CPU resumes execution at the new program counter address.

For instructions that can simultaneously generate Program Exceptions (privileged instructions executed in Problem State) and Data Storage Exceptions, the Program Exception has priority.

The following registers will be modified to the specified values:

The cache line locking errors occur when certain cache control instructions attempt to access a cache line, *in user mode*, when the lock exception bits are enabled in the Cache Debug Control Register (CDBCR). The data storage exception occurs regardless of whether the cache line is locked. Section 25.5, “Cache Line Locking,” on page 25-11, describes cache line locking, including resulting exceptions.

The following cache line locking errors occur in user mode:

- If **dcbf** attempts to access a cache line while the DCU unlock exception is enabled (CDBCR[DUXE]=1).
- If **dcbz** references a cacheable address while the DCU lockout exception is enabled (CDBCR[DLXE]=1). An alignment error also occurs; however, in this case, the data storage exception takes priority.
- If **dcbz** references a non-cacheable address while the DCU unlock exception is enabled (CDBCR[DUXE]=1). An alignment error, which takes priority, also occurs.
- If **icbi** attempts to invalidate a locked cache line, while the ICU unlock exception is enabled (CDBCR[IUXE]=1).

When a data storage exception occurs, the IOP 480 CPU suppresses execution of the instruction causing the exception and writes the EA of the instruction address in SRR0. The current contents of the MSR are saved into SRR1. The DEAR is written with the EA of the failed access. ESR[DLK] is set to indicate the cause of the exception.

The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0300. Exception processing begins at the new address in the program counter.

Table 11-9. Register Settings during Data Storage Exceptions

SRR0	Written with the EA of the instruction causing the data storage exception
SRR1	Written with the value of the MSR at the time of the exception
MSR	APE, APA, WE, PR, EE ← 0 CE, ME, DE ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0300
DEAR	Written with the EA of the failed access
ESR	The DLK, DST, DIZ, and DSK fields are set to indicate the cause of the exception. See Figure 11-7 on page 11-18 for details of the DLK field.

11.7.7 Instruction Storage Exception

The Instruction Storage Exception is generated when instruction translation is active and execution is attempted for an instruction whose fetch access to the effective address is not permitted for any of the following reasons:

- In Problem State
  - Instruction fetch from an effective address with (ZPR field)=00.
  - Instruction fetch from an effective address with the EX bit clear and (ZPR field) ≠ 11.
  - Instruction fetch from an effective address contained within a Guarded region (G=1).
- In Supervisor State
  - Instruction fetch from an effective address with the EX bit clear and (ZPR field) other than 11 or 10.
  - Instruction fetch from an effective address contained within a Guarded region (G=1).

SRR0 saves the address of the instruction causing the Instruction Storage exception.

ESR is set to indicate the following conditions:

- If ESR[DIZ]=1, the excepting condition was a zone fault: the attempted execution of an instruction address fetched in user-mode with (ZPR field)=00.
- If ESR[DIZ]=0, then the excepting condition was either EX=0 or G=1.

The exception is precise with respect to the attempted execution of the instruction. Program flow vectors to EVPR[0:15] || 0x0400.The following registers are modified to the specified values.

Table 11-10. Register Settings during Instruction Storage Exceptions

SRR0	Set to the EA of the instruction for which execute access was not permitted
SRR1	Set to the value of the MSR at the time of the exception
MSR	APE, APA, WE, PR, EE ← 0 CE, ME, DE ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0400
ESR	***Notes:****DIZ ← 1If access failure due to a zone protection fault (ZPR[Zn]=00 in user mode)* *If the DIZ bit is not set, the exception occurred because TBL_entry[EX] was clear in an otherwise accessible zone or instruction fetch from a storage region marked as guarded.* *See Section 11.7.3.5, "Exception Syndrome Register (ESR)," on page 11-17 for details of ESR operation.*

11.7.8 External Interrupt Exception

External interrupt exceptions are triggered by active levels on the external interrupt inputs. All external interrupting events are presented to the processor as a single external interrupt. External interrupts are enabled or disabled by the MSR[EE] bit.

Note: *The MSR[EE] bit also enables the occurrence of PIT and FIT interrupts. However, after timer interrupts, control passes to different exception vectors than for the interrupts discussed in the preceding paragraph. Therefore, these timer exceptions are described in Section 11.7.12, "Programmable Interval Timer (PIT) Exception," on page 11-26 and Section 11.7.13, "Fixed Interval Timer (FIT) Exception," on page 11-26.*

11.7.8.1 External Interrupt Exception Handling

When MSR[EE]=1 (external interrupts are enabled), and a non-critical external interrupt exception occurs, and this exception is the highest priority exception condition, the processor immediately writes the address of the next sequential instruction into SRR0. Simultaneously, the contents of the MSR are saved in SRR1.

When the processor takes a non-critical external interrupt, MSR[EE] is reset to 0. This disables other external interrupts from interrupting the exception handler before SRR0 and SRR1 are saved. The MSR is also written with the other values shown in Table 11-11. The high-order 16 bits of the program counter are written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0500. Exception processing begins at the address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 11-11. Register Settings during External Interrupt Exceptions

SRR0	Written with the address of the next sequential instruction
SRR1	Written with the contents of the MSR
MSR	APE, APA, WE, PR, EE ← 0 CE, ME, DE ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0500

Table 11-12. Alignment Exception Summary

IOP 480 CPU MSR	Instructions Causing Alignment Exceptions	Conditions
MSR[LE]=0	dcbz	EA in non-cacheable or write-through storage
	dcread, lwarx, stwcx.	EA not Lword-aligned
MSR[LE]=1	dcbz	EA in non-cacheable or write-through storage
	lha, lhau, lhaux, lhax, lhbrx, lhz, lhzu, lhzux, lhzx, sth, sthbrx, sthu, sthux, sthx	EA not word-aligned
	dcread, lwarx, lwbrx, lwz, lwzu, lwzux, lwzx, stw, stwbrx, stwcx., stwu, stwux, stwx	EA not Lword-aligned
	lmw, lswi, lswx, stmw, stswi, stswx	Always

11.7.9 Alignment Exception

Alignment exceptions are caused by misaligned data accesses by storage reference instructions, a **dcbz** instruction to non-cacheable or write through storage, or load/store multiple and string instructions when MSR[LE]=1 (in PowerPC Little Endian mode). Table 11-12 summarizes the instructions and conditions causing alignment exceptions.

Execution of an instruction causing an alignment exception is prohibited from completing. SRR0 is written with the address of that instruction and the current contents of the MSR are saved into SRR1. The DEAR is written with the address that caused the alignment error. The MSR bits are written with the values shown in Table 11-13. The high-order 16 bits of the program counter are written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0600. Exception processing begins at the new address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter

Note: *Alignment exceptions cannot be disabled. To avoid overwrites of SRR0 and SRR1 by alignment exceptions that occur within a handler, exception handlers should save these registers as soon as possible.*

Table 11-13. Register Settings during Alignment Error Exceptions

SRR0	Written with the address of the instruction causing the alignment exception
SRR1	Written with the contents of the MSR
MSR	APE, APA, WE, PR, EE ← 0 CE, ME, DE, PX ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0600
DEAR	Written with the address that caused the alignment violation

11.7.10 Program Exceptions

Program exceptions are caused by attempting to execute an illegal operation, by executing a trap instruction with conditions satisfied, or by attempting to execute a privileged instruction while in the problem state.

The ESR bits that differentiate these situations (refer to Table 11-14) are mutually exclusive—when a program exception occurs, the appropriate bit is set and the others are cleared. These exceptions are not maskable.

Table 11-14. ESR Usage for Program Exceptions

Bit	Exception Cause
ESR[PIL]	Illegal
ESR[PPR]	Privileged
ESR[PTR]	Trap
ESR[PAU]	Auxiliary processor unavailable
ESR[PAE]	Auxiliary processor exception

The program exception interrupt handler does not need to reset the ESR.

When execution of an illegal instruction is attempted (including memory management instructions when TIE_cpuMmuEn=0 or the APU_cpuException signal is asserted), or when execution of a privileged instruction is attempted in problem state, the IOP 480 CPU does not execute the instruction, and it writes the address of the excepting instruction into SRR0.

Trap instructions can be used as a program exception or a debug event, or both (refer to Section 26, “IOP 480 CPU Debugging and JTAG Facilities,” for information about debug events). When a trap instruction is detected as a program exception, the IOP 480 CPU writes the address of the trap instruction into SRR0. See **tw** on page 28-163 and **twi** on page 28-166 for a detailed discussion of the behavior of trap instructions with various exceptions enabled.

If MSR[APA]=0, an attempt to execute an instruction intended for an auxiliary processor unit (APU) causes a program exception if MSR[APE]=0 or the APU_cpuException signal is asserted.

After any program exception, the contents of the MSR are written into SRR1 and the MSR bits are written with the values shown in Table 11-15. The high-order 16 bits of the program counter are written with the contents of the EVPR; the low-order 16 bits of the program counter are written with 0x0700. Exception processing begins at the new address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 11-15. Register Settings during Program Exceptions

SRR0	Written with the address of the excepting instruction
SRR1	Written with the contents of the MSR
MSR	WE, PR, EE ← 0 CE, ME, DE ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0700
ESR	Written with the type of program exception (refer to Table 11-5 on page 11-19 and Table 11-14 on page 11-25)

11.7.11 System Call Exception

System call exceptions occur when a **sc** instruction is executed. The IOP 480 CPU writes the address of the instruction following the **sc** into SRR0. The contents of the MSR are written into SRR1 and the MSR bits are written with the values shown in Table 11-16. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x0C00. Exception processing begins at the new address in the program counter.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 11-16. Register Settings during System Call Exceptions

SRR0	Written with the address of the instruction following the **sc** instruction
SRR1	Written with the contents of the MSR
MSR	APE, APA, WE, PR, EE ← 0 CE, ME, DE ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x0C00

11.7.12 Programmable Interval Timer (PIT) Exception

For a discussion of the IOP 480 CPU timer facilities, see Section 11.7.18, “Timer Facilities,” on page 11-28. The PIT is described in Section 11.7.18.2, “Programmable Interval Timer (PIT),” on page 11-32.

If the PIT exception is enabled by TCR[PIE] and MSR[EE], the IOP 480 CPU initiates a PIT interrupt after detecting a timeout from the PIT. Timeout is detected when, at the beginning of a clock cycle, TSR[PIS]=1. (This occurs on the cycle after the PIT decrements on a PIT count of 1.) The IOP 480 CPU immediately takes the interrupt. The address of the next sequential instruction is saved in SRR0; simultaneously, the contents of the MSR are written into SRR1 and the MSR is written with the values shown in Table 11-17. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x1000. Exception processing begins at the address in the program counter.

To clear a PIT interrupt, the exception handling routine must clear the PIT interrupt bit, TSR[PIS]. Clearing is performed by writing an Lword to TSR, using an **mtspr** instruction, that has 1 in bit positions to be cleared and 0 in all other bit positions. The data written to the TSR is not direct data, but a mask; a 1 clears the bit and 0 has no effect.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 11-17. Register Settings during Programmable Interval Timer Exceptions

SRR0	Written with the address of the next instruction to be executed
SRR1	Written with the contents of the MSR
MSR	APE, APA, WE, PR, EE ← 0 CE, ME, DE ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x1000
TSR	PIS ← 1

11.7.13 Fixed Interval Timer (FIT) Exception

For a discussion of the IOP 480 CPU timer facilities, see Section 11.7.18, “Timer Facilities,” on page 11-28. The FIT is described in Section 11.7.18.3, “Fixed Interval Timer (FIT),” on page 11-33.

If the FIT exception is enabled by TCR[FIE] and MSR[EE], the IOP 480 CPU initiates a FIT interrupt after detecting a timeout from the FIT. Timeout is detected when, at the beginning of a clock cycle, TSR[FIS]=1. (This occurs on the second cycle after the 0→1 transition of the appropriate time-base bit.) The IOP 480 CPU immediately takes the interrupt. The address of the next sequential instruction is written into SRR0; simultaneously, the contents of the MSR are written into SRR1 and the MSR is written with the values shown in Table 11-18. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x1010. Exception processing begins at the address in the program counter.

To clear a FIT interrupt, the exception handling routine must clear the FIT interrupt bit, TSR[FIS]. Clearing is performed by writing an Lword to TSR, using an **mtspr** instruction, that has 1 in any bit positions to be cleared and 0 in all other bit positions. The data written to the TSR is not direct data, but a mask; a 1 clears a bit and 0 has no effect.

Executing an **rfi** instruction restores the program counter from SRR0 and the MSR from SRR1, and execution resumes at the address in the program counter.

Table 11-18. Register Settings during Fixed Interval Timer Exceptions

SRR0	Written with the address of the next sequential instruction
SRR1	Written with the contents of the MSR
MSR	APE, APA, WE, PR, EE ← 0 CE, ME, DE ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x1010
TSR	FIS ← 1

11.7.14 Watchdog Timer Exception

For a general description of the IOP 480 CPU timer facilities, see Section 11.7.18, “Timer Facilities,” on page 11-28. The watchdog timer (WDT) is described in Section 11.7.18.4, “Watchdog Timer,” on page 11-33.

If the WDT exception is enabled by TCR[WIE] and MSR[CE], the IOP 480 CPU initiates a WDT interrupt after detecting the first WDT timeout. First timeout is detected when, at the beginning of a clock cycle, TSR[WIS]=1. (This occurs on the second cycle after the 0→1 transition of the appropriate time-base bit while TSR[ENW]=1 and TSR[WIS]=0.) The IOP 480 CPU immediately takes the interrupt. The address of the next sequential instruction is saved in SRR2; simultaneously, the contents of the MSR are written into SRR3 and the MSR is written with the values shown in Table 11-19. The high-order 16 bits of the program counter are then written with the contents of the EVPR and the low-order 16 bits of the program counter are written with 0x1020. Exception processing begins at the address in the program counter.

To clear the WDT interrupt, the exception handling routine must clear the WDT interrupt bit TSR[WIS]. Clearing is done by writing an Lword to TSR (using **mtspr**), with a 1 in any bit position that is to be cleared and 0 in all other bit positions. The data written to the status register is not direct data, but a mask; a 1 causes the bit to be cleared, and a 0 has no effect.

Executing the return from critical interrupt instruction (**rfci**) restores the contents of the program counter and the MSR from SRR2 and SRR3, respectively, and the IOP 480 CPU resumes execution at the contents of the program counter.

Table 11-19. Register Settings during Watchdog Timer Exceptions

SRR2	Written with the address of the next sequential instruction
SRR3	Written with the contents of the MSR
MSR	APE, APA, WE, PR, CE, EE, DE ← 0 ME ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x1020
TSR	WIS ← 1

11.7.15 Data TLB Miss Exception

The data TLB miss exception is generated if data translation is enabled and a valid TLB entry matching the EA and PID is not present. The address of the instruction generating the untranslatable effective data address is saved in SRR0. In addition, the hardware also saves the data address (that missed in the TLB) in the DEAR.

The ESR is set to indicate whether the excepting operation was a store (includes **dcbz**, **dcbi**, **dccci**).

The exception is precise. Program flow vectors to EVPR[0:15] || 0x1100.

The following registers are modified to the specified values.

Table 11-20. Register Settings during Data TLB Miss Exceptions

SRR0	Set to the address of the instruction generating the effective address for which no valid translation exists.
SRR1	Set to the value of the MSR at the time of the exception
MSR	APE, APA, WE, PR, EE, PE ← 0 CE, ME, DE, PX ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x1100
DEAR	Set to the effective address of the failed access
ESR	DST ← 1 if excepting operation is a store operation (includes **dcbi**, **dcbz**, and **dccci**). See Section 11.7.3.5, “Exception Syndrome Register (ESR),” on page 11-17 for details of ESR operation.

Note: *Data TLB miss exceptions can happen whenever data translation is active. Therefore, ensure that SRR0 and SRR1 are saved before enabling translation in an exception handler.*

11.7.16 Instruction TLB Miss Exception

The instruction TLB miss exception is generated if instruction translation is enabled and execution is attempted for an instruction for which a valid TLB entry matching the EA and PID for the instruction fetch is not present. The instruction whose fetch caused the TLB miss is saved in SRR0.

The exception is precise with respect to the attempted execution of the instruction. Program flow vectors to EVPR[0:15 || 0x1200.

The following registers are modified to the specified values.

Table 11-21. Register Settings during Instruction TLB Miss Exceptions

SRR0	Set to the address of the instruction for which no valid translation exists.
SRR1	Set to the value of the MSR at the time of the exception
MSR	APE, APA, WE, PR, EE, PE ← 0 CE, ME, DE, PX ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x1200

Note: *Instruction TLB miss exceptions can happen any time instruction translation is active. Therefore, insure that SRR0 and SRR1 are saved before enabling translation in an exception handler.*

11.7.17 Debug Exception Handling

Debug exceptions can be either *synchronous* or *asynchronous*. The following debug events generate synchronous exceptions: instruction address compare (also referred to as IAC), data address compare (also referred to as DAC), trap with condition satisfied (TIE), branch taken (BT), and instruction completion (IC). The following debug events generate asynchronous exceptions: unconditional debug event (UDE) and exceptions (EXC). See Section 26, “IOP 480 CPU Debugging and JTAG Facilities,” for more information about debug events.

For debug events, SRR2 is written with an address, which varies with the type of debug event, as shown in Table 11-22.

Table 11-22. SRR2 during Debug Exceptions

Debug Event	Address Saved in SRR2
IAC DAC TDE BT	Address of the instruction that caused the event
IC	Address of the instruction *following* the instruction that caused the event
UDE	Address of next instruction to be executed at time of UDE
EDE	Interrupt vector address of the initial exception that caused the exception debug event

SRR3 is written with the contents of the MSR and the MSR is written with the values shown in Table 11-23. The high-order 16 bits of the program counter are then written with the contents of the EVPR; the low-order 16 bits of the program counter are written with 0x2000. Exception processing begins at the address in the program counter.

Executing an **rfci** instruction restores the program counter from SRR2 and the MSR from SRR3, and execution resumes at the address in the program counter.

Table 11-23. Register Settings during Debug Exceptions

SRR2	Written with an address as described in Table 11-22
SRR3	Written with the contents of the MSR
MSR	APE, APA, WE, PR, CE, EE, DE ← 0 ME ← unchanged DR, IR ← 0 ILE ← unchanged LE ← ILE
PC	EVPR[0:15] || 0x2000
DBSR	Set to indicate type of debug event.

11.7.18 Timer Facilities

The IOP 480 CPU provides four timer facilities: a time base, a Programmable Interval Timer (PIT), a Fixed Interval Timer (FIT), and a Watchdog Timer (WDT). These facilities, which share the same base clock frequency, can support:

- Time-of-day functions
- Data logging functions
- Peripherals requiring periodic schedule service
- General system maintenance

Additionally, the timer facilities can help a system recover from faulty hardware or software.

Figure 11-3 on page 11-30 shows the relationship of these facilities and base clock.

The time base of the IOP 480 CPU is functionally similar to the time base defined in PowerPC Architecture, but the two are not the same. Descriptions of the difference, as an aid to porting code, follow.

For PowerPC Architecture:

- The PowerPC Architecture defines a 64-bit time base that can be accessed as a pair of 32-bit registers. Different register numbers are used for read access (user or privileged mode) and write access (privileged mode).
- The PowerPC Architecture provides a **mftb** (move from time base) instruction for user-mode read access to the time base. The register numbers (0x10C and 0x10D) used to specify the time base registers are not SPR numbers. However, the opcode of **mftb** differs from the opcode of **mfspr** by only one bit. The PowerPC Architecture allows an implementation to ignore this bit and handle **mftb** as **mfspr**. Therefore, the time base register numbers cannot specify other SPRs. Further, PowerPC compilers can use the **mftb** opcode only with the register numbers specified in the PowerPC Architecture as read-access time base registers (0x10C and 0x10D).
- The PowerPC Architecture does not provide privileged-mode-only read access to the time base (by definition, the user-mode read access mechanism is also available in privileged mode).
- The PowerPC Architecture provides privileged-mode write access to the time base using **mtspr** with SPR numbers 0x11C and 0x11D.

For the IOP 480 CPU:

- IOP 480 CPU defines a 64-bit time base that can be accessed as a pair of 32-bit registers. Different register numbers are used for read access (user or privileged mode) and write access (privileged mode). The IOP 480 CPU also allows read access (in privileged mode only) using the same register number as a write access.
- IOP 480 CPU provides user-mode and privileged-mode read access to the time base using **mfspr** with SPR numbers 0x3CD and 0x3CC. The IOP 480 CPU provides privileged-mode-only read access via **mfspr** instructions with SPR numbers 0x3DD and 0x3DC. The IOP 480 CPU does not implement **mftb**; an attempt to use it results in a program exception caused by the “illegal” opcode.
- IOP 480 CPU provides privileged-mode write access to the time base using **mtspr** instructions with SPR numbers 0x3DD and 0x3DC.

Table 11-24 on page 11-31 summarizes the differences between the time bases.

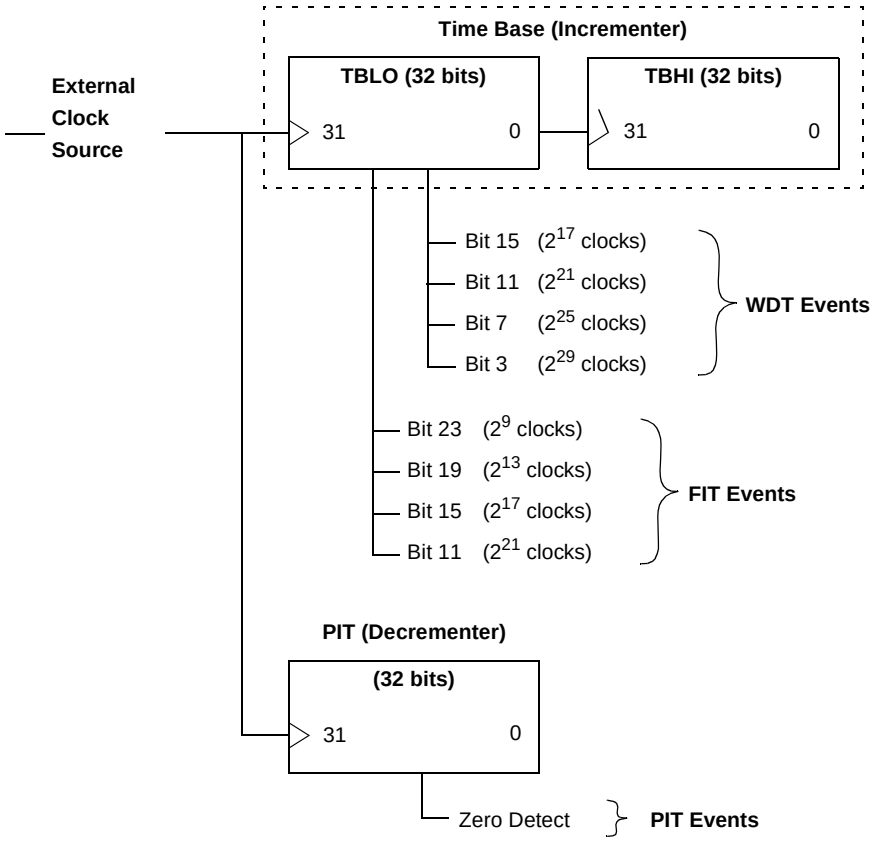


Figure 11-3. Relationship of Timer Facilities to the Base Clock

11.7.18.1 Time Base

The IOP 480 CPU implements a 64-bit time base. The time base, which increments once during each period of the time base clock, provides a time reference. The time base is accessed using the 32-bit registers TBLO and TBHI. Software access to the time base is through the **mfspr** and **mtspr** instructions.

Access to the time base registers TBHI and TBLO is privileged.

User-mode read-only access to the Time Base is provided by reading from different SPR numbers. Specifically, read-only access to TBHI is accomplished by reading TBHU, and read-only access to TBLO is accomplished by reading TBLU. Both TBHU and TBLU are read using **mfspr** instructions. An **mtspr** to these registers is boundedly undefined.

The period of the 64-bit Time Base is approximately 7794 years for a 75 MHz time base clock. The time base does not generate interrupts, even when it wraps. For most applications, the time base is set at system reset and only read thereafter. Note that the FIT and the watchdog timer (discussed below) are driven by 0→1 transitions of selected bits of TBLO. Transitions caused by software alteration of TBLO, using **mtspr,** have the same effect as transitions caused by normal incrementing of the time base.

Register 11-9 illustrates TBHI(TBHU); Register 11-10 illustrates TBLO(TBLU).

Register 11-9. Time Base Register (TBHI, TBHU)

0:31		Time High	Current count, high-order. ***Note:*** *TBHU provides read-only access to the time base register TBHI. The contents of TBHU and TBHI never differ).*

Register 11-10. Time Base Register (TBLO, TBLU)

0:31		Time Low	Current count, low-order. ***Note:*** *TBLU provides read-only access to the time base register TBLO. The contents of TBLU and TBLO never differ).*

Table 11-24. Time Base Comparison

	PowerPC Architecture			IOP 480 CPU		
	Access Instructions	**Register Number**	**Access Restrictions**	**Access Instructions**	**Register Number**	**Access Restrictions**
Upper 32 bits	mftbu RT *Extended mnemonic for* mftb RT,TBU	0x10D	Read-Only	mftbhu RT *Extended mnemonic for* mfspr RT TBHU	0x3CC	Read-Only
	mttbu RS *Extended mnemonic for* *mtspr TBU,RS*	0x11D	Privileged; Write-Only	mftbhi RT *Extended mnemonic for* mfspr RT,TBHI mttbhi RS *Extended mnemonic for* mtspr TBHI,RS	0x3DC	Privileged; Read Privileged; Write
Lower 32 bits	mftb RT *Extended mnemonic for* *mftb RT,TBL*	0x10C	Read-Only	mftblu RT *Extended mnemonic for* mfspr RT,TBLU	0x3CD	Read-Only
	mttbl *Extended mnemonic for* *mtspr TBL,RS*	0x11C	Privileged; Write-Only	mftblo RT *Extended mnemonic for* mfspr RT,TBLO mttblo RS *Extended mnemonic for* mtspr TBLO,RS	0x3DD	Privileged; Read Privileged; Write

11.7.18.2 Programmable Interval Timer (PIT)

The PIT is a 32-bit register that decrements at the same rate as the time base. The PIT is read or written using **mfspr** or **mtspr**. Writing to the PIT, using **mtspr,** simultaneously writes to a hidden reload register. Reading the PIT using **mfspr** returns the current PIT contents; the hidden reload register cannot be read. When a non-zero value is written to the PIT, it begins to decrement. A PIT event occurs when a decrement occurs on a PIT count of 1. When a PIT event occurs, the following occur:

1. If the PIT is in auto-reload mode (TCR[ARE]=1), the PIT is loaded with the last value an **mtspr** wrote to the PIT. A decrement from a PIT count of 1 immediately causes a reload; no intermediate PIT content of 0 occurs.

   If the PIT is not in auto-reload mode (TCR[ARE]=0), a decrement from a PIT count of simply causes a PIT content of 0.
2. TSR[PIS] is set to 1.
3. If enabled (TCR[PIE]=1 and MSR[EE]=1), a PIT interrupt is taken. See Section 11.7.12, “Programmable Interval Timer (PIT) Exception,” on page 11-26, for details of register behavior during a PIT interrupt.

The interrupt handler should use software to reset the TSR[PIS] bit. This is done by using **mtspr** to write an Lword to the TSR having a 1 in TSR[PIS] and any other bits to be cleared, and a 0 in all other bits. The data written to the TSR is not direct data, but a mask. A 1 clears a bit; a 0 has no effect.

Using **mtspr** to force the PIT to 0 *does not* cause a PIT interrupt. However, decrementing that was ongoing at the instant of the **mtspr** instruction can cause the appearance of an interrupt. To eliminate the PIT as a source of interrupts, write a 0 to TCR[PIE], the PIT interrupt enable bit.

To eliminate all PIT activity:

1. Write a 0 to TCR[PIE]. This prevents PIT activity from causing interrupts.
2. Write a 0 to TCR[ARE]. This disables the PIT auto-reload feature.
3. Write zeroes to the PIT to halt PIT decrementing. Although this action does not cause a PIT interrupt to become pending, a near-simultaneous decrement might have done so.
4. Write a 1 to TSR[PIS] (PIT Interrupt Status bit). This clears TSR[PIS] to 0 (refer to Section 11.7.18.5, “Timer Status Register (TSR),” on page 11-35). This also clears any pending PIT interrupt. Because the PIT freezes, no further PIT events are possible.

If the auto-reload feature is disabled (TCR[ARE]=0) after the PIT decrements to 0, the PIT remains 0 until software uses **mtspr** to reload it.

On all resets, TCR[ARE]=0, which disables the auto-reload feature.

Register 11-11 lists the PIT bits.

Register 11-11. Programmable Interval Timer (PIT)

0:31		Programmed Interval Remaining	The number of clocks remaining until the PIT event

11.7.18.3 Fixed Interval Timer (FIT)

The FIT provides timer interrupts having a repeatable period, facilitating system maintenance. The FIT is functionally similar to an auto-reload PIT, except that fewer selections of interrupt periods are available.

The FIT exception occurs on 0→1 transitions of selected bits from the time base, as shown in Table 11-25.

Table 11-25. FIT Controls

TCR[FP]	TBLO Bit	Period (Time Base Clocks)	Period (66 MHz Clock)
0, 0	23	2^9 clocks	7.68 µs
0, 1	19	2^{13} clocks	122.9 µs
1, 0	15	2^{17} clocks	1.966 ms
1, 1	11	2^{21} clocks	31.46 ms

The TSR[FIS] bit logs a FIT exception as a pending interrupt. A FIT interrupt occurs if TCR[FIE] and MSR[EE] are enabled at the time of the FIT exception. Section 11.7.13, “Fixed Interval Timer (FIT) Exception,” on page 11-26, describes register behavior during a FIT interrupt.

The interrupt handler must use software to reset the TSR[FIS] bit. This is done by using **mtspr** to write an Lword to the TSR having a 1 in TSR[FIS] and any other bits to be cleared, and a 0 in all other bits. The data written to the TSR is not direct data, but a mask. A 1 clears a bit and a 0 has no effect.

11.7.18.4 Watchdog Timer

The watchdog timer (WDT) aids system recovery from software or hardware faults.

A WDT timeout occurs on 0→1 transitions of selected bits from the time base, as shown in Table 11-26.

Table 11-26. Watchdog Timer Controls

TCR[WP]	TBLO Bit	Period (Time Base Clocks)	Period (66 MHz Clock)
0,0	15	2^{17} clocks	1.966 ms
0,1	11	2^{21} clocks	31.46 ms
1,0	7	2^{25} clocks	0.503s
1,1	3	2^{29} clocks	8.053s

If a WDT timeout occurs while TSR[WIS]=0 and TSR[ENW]=1, a WDT interrupt occurs if the interrupt is enabled by TCR[WIE] and MSR[CE]. Section 11.7.14, “Watchdog Timer Exception,” on page 11-27 describes register behavior during a WDT interrupt.

The interrupt handler must use software to reset the TSR[WIS] bit. This is done by using **mtspr** to write an Lword to the TSR having a 1 in TSR[WIS] and any other bits to be cleared, and a 0 in all other bits. The data written to the TSR is not direct data, but a mask. A 1 clears a bit and a 0 has no effect.

If a WDT timeout occurs while TSR[WIS]=1 and TSR[ENW]=1, a hardware reset occurs if enabled by a non-zero value of TCR[WRC]. The assumption is that TSR[WIS] was not cleared because the processor could not execute the watchdog handler, leaving reset as the only way to restart the system. Note that after TCR[WRC] is set to a non-zero value, it cannot be reset by software. This prevents errant software from disabling the WDT reset capability.

Figure 11-4 and Table 11-27 describe the watchdog state machine. In the figure, numbers in parentheses refer to descriptions of operating modes that follow the table.

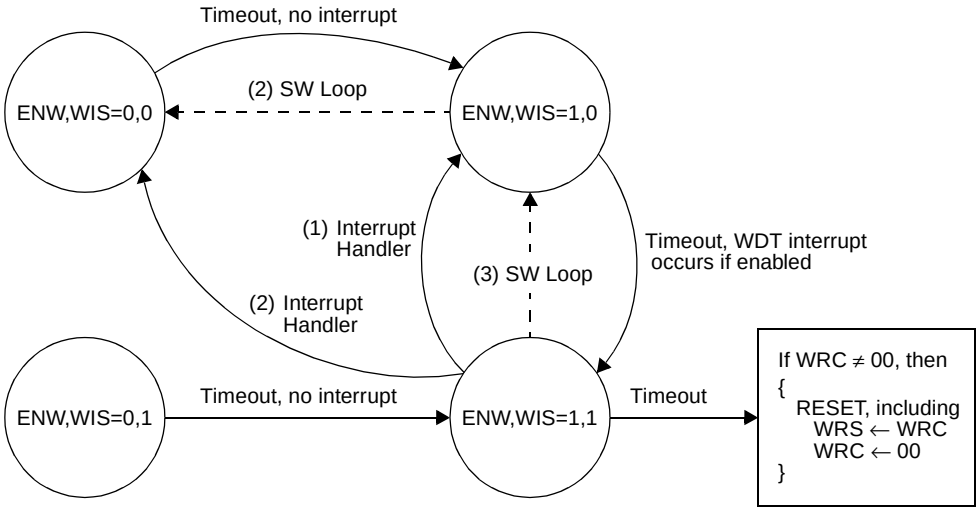


Figure 11-4. Watchdog Timer State Machine

Table 11-27. Watchdog Timer State Machine

Enable Next WDT TSR[ENW]	WDT Status TSR[WIS]	Action when Timer Interval Expires
0	0	Set enable next watchdog (TSR[ENW]=1).
0	1	Set TSR[ENW]=1.
1	0	Set the watchdog interrupt status (TSR[WIS]=1). If TCR[WIE]=1 and MSR[CE]=1, then interrupt.
1	1	Cause the watchdog reset action specified by TCR[WRC]. On reset, copy pre-reset TCR[WRC] into TSR[WRS] and clear TCR[WRC].

The controls described in the above table imply three different operating modes that a programmer can select for the WDT. The modes assume that TCR[WRC] was set to allow processor reset by the WDT:

1. Always take a pending WDT interrupt, and never attempt to prevent its occurrence. (This mode is described in the preceding text.)

- Clear TSR[WIS] in the WDT handler.
- Never use TSR[ENW].

2. Always take a pending WDT interrupt, but avoid it whenever possible.

   This assumes that a recurring code loop of reliable duration exists outside the interrupt handlers, or that a FIT interrupt handler is operational. One of these mechanisms clears TSR[ENW] more frequently than the watchdog period.

- Clear TSR[ENW] to 0 in loop or in FIT handler.
- To clear TSR[ENW], use **mtspr** to write a 1 to TSR[ENW] (and to any other bits that are to be cleared), with 0 in all other bit locations.
- Clear TSR[WIS] in WDT handler. (This is an unexpected event.)

3. Never take a WDT interrupt.

   This assumes that a recurring code loop of reliable duration exists outside the interrupt handlers, or that a FIT interrupt handler is operational. This method guarantees only one WDT period before a reset occurs.

- Clear TSR[WIS] in the loop or in FIT handler.
- Never use TSR[ENW].

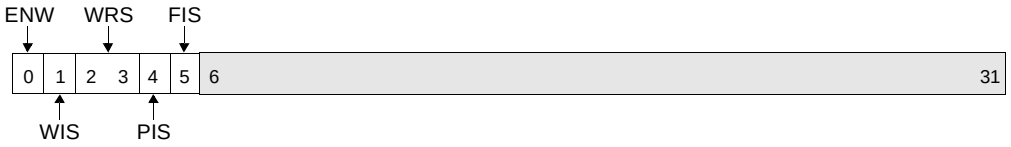
11.7.18.5 Timer Status Register (TSR)

The TSR can be accessed for read or write-to-clear.

Status registers are generally set by hardware and read and cleared by software. The **mfspr** instruction reads the TSR. Clearing the TSR is performed by writing an Lword to the TSR, using **mtspr,** having a 1 in all bit positions to be cleared and a 0 in all other bit positions. The data written to the TSR is not direct data, but a mask. A 1 clears the bit and a 0 has no effect.

Register 11-12 illustrates the TSR bits.

Register 11-12. Timer Status Register (TSR)



0	ENW	Enable Next Watchdog 0 Action on next Watchdog event is to set TSR[0]. 1 Action on next Watchdog event is governed by TSR[1].	See Section 11.7.18.4, “Watchdog Timer,” on page 11-33.
1	WIS	Watchdog Interrupt Status 0 No Watchdog interrupt is pending. 1 Watchdog interrupt is pending.	
2:3	WRS	Watchdog Reset Status 00 No Watchdog reset has occurred. 01 Core reset was forced by the Watchdog. 10 Chip reset was forced by the Watchdog. 11 System reset was forced by the Watchdog.	
4	PIS	PIT Interrupt Status 0 No PIT interrupt is pending. 1 PIT interrupt is pending.	
5	FIS	FIT Interrupt Status 0 No FIT interrupt is pending. 1 FIT interrupt is pending.	
6:31		***Reserved***	

11.7.18.6 Timer Control Register (TCR)

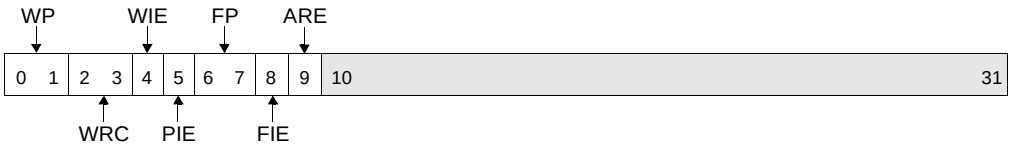
The TCR controls PIT, FIT, and WDT operation.

The TCR[WRC] field is cleared to 0 by all processor resets. (Section , “This subsection describes the initial state of the IOP 480 CPU after a reset, and contains an example of the initialization code required to begin executing application code. Initialization of external system components or system-specific chip facilities may also need to be performed in addition to the basic initialization described in this section.,” on page 10-8, describes the types of processor reset.) This field is set only by software. However, hardware does not allow software to clear the field after it is set. After software writes a 1 to a bit in the field, that bit remains a 1 until any reset occurs. This prevents errant code from disabling the WDT reset function.

All processor resets clear TCR[ARE] to 0, disabling the auto-reload feature of the PIT.

Register 11-13 illustrates the TCR bits.

Register 11-13. Timer Control Register (TCR)



0:1	WP	Watchdog Period 00 2^{17} clocks 01 2^{21} clocks 10 2^{25} clocks 11 2^{29} clocks	
2:3	WRC	Watchdog Reset Control 00 No Watchdog reset occurs. 01 Core reset is forced by the Watchdog. 10 Chip reset is forced by the Watchdog. 11 System reset is forced by the Watchdog.	TCR[WRC] resets to 00. This field can be set by software, but cannot be cleared by software, except by a software-induced reset.
4	WIE	Watchdog Interrupt Enable 0 Disable WDT interrupt. 1 Enable WDT interrupt.	
5	PIE	PIT Interrupt Enable 0 Disable PIT interrupt. 1 Enable PIT interrupt.	
6:7	FP	FIT Period 00 2^{9} clocks 01 2^{13} clocks 10 2^{17} clocks 11 2^{21} clocks	
8	FIE	FIT Interrupt Enable 0 Disable FIT interrupt. 1 Enable FIT interrupt.	
9	ARE	Auto Reload Enable 0 Disable auto reload. 1 Enable auto reload.	Disables on reset.
10:31		***Reserved***	

12 MEMORY CONTROLLER

12.1 OVERVIEW

The IOP 480 Memory Controller supports up to four banks of SRAM by using LCS[3:0]# signals and up to four banks of either Extended Data Out (EDO) DRAM or Synchronous DRAM (SDRAM) by using MCS[3:0]# signals. The SRAM banks (LCS[3:0]# signals) can also be used to access PROMs, I/O peripheral chips, Flash memory, parallel EEPROMs, and FIFO. Four different Masters can access these banks of memory and peripherals. Each of these Masters have 32-bit-wide data buses:

- Direct Slave Controller
- Internal DMA Channels 0, 1, and 2
- Internal IOP 480 CPU
- Local Bus Master

When an address on the Local Bus falls into the programmed range of one of the SRAM or DRAM banks, the Memory controller accesses the appropriate device. Either single or burst accesses are supported. If a Local address does not match any of the programmed ranges, nor does it fall into the Direct Master or Configuration register ranges, then a default bus descriptor register is used to define the characteristics of the bus. This register is used when the IOP 480 is accessing a general-purpose device on the Local Bus that is not controlled by an SRAM or DRAM register.

SRAM banks have individual sets of configuration registers that define base address, size, and bus characteristics. All four DRAM banks use one additional set of configuration registers. The Bus width of the SRAM banks can be 8, 16, or 32 bits, while the bus width of all four DRAM banks must be 32 bits.

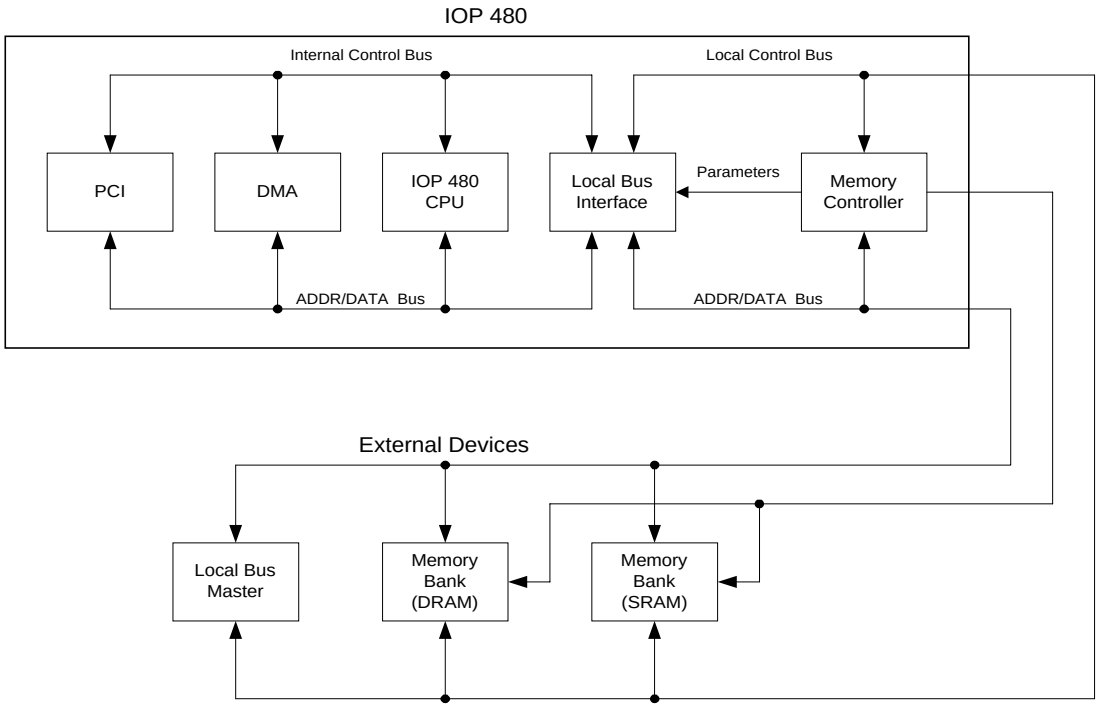


Figure 12-1. Memory Block Diagram

Table 12-1. SRAM Signals

Source	Signal	Description
Memory Controller	MA[12:0]	De-multiplexed address bits [14:2]
	DP[3:0]/MA[16:13]	De-multiplexed address bits [18:15] or parity bits
	LCS3#/MA17	De-multiplexed address bit 19 or LCS3#
	MOE#	Memory output enable
	LCS[2:0]#	Local chip selects
	MDQM[3:0]#	Byte write signals
	RD#	Read strobe
	MWE#	Write strobe
Local Bus	LAD[31:0]	Address/Data bus (LAD0 is least significant)
	LBE[3:0]#	Byte enables; also used as address bits [1:0]
	ALE	Address latch enable
	LWR#	Write/Read status

Table 12-2. Local Burst Address

Device Width	Device Address		
	LA[19:2]	LA1	LA0
8 bits	MA[17:0]	LBE1#	LBE0#
16 bits	MA[17:0]	LBE1#	Not Used
32 bits	MA[17:0]	Not Used	Not Used

12.2 SRAM

SRAM banks can be used for accesses to SRAM, PROM, external asynchronous FIFOs, or I/O peripheral devices.

12.2.1 Control Signals

Minimum size of each SRAM bank is 256 bytes. The following address, data, and control signals are used to access an SRAM device. Each unique application does not necessarily use all of the following signals, but only those required by the application.

12.2.2 Address

The Local Address Bus is de-multiplexed by the Memory controller to produce address bits [19:2] to memory. For devices which use more than 20 address bits (1 MB), LAD[31:20] must be de-multiplexed in an external latch controlled by ALE. The address connections to various widths of memory devices are shown below. A[19:0] represent address pins on the memory device. The local byte enables are used to provide the least significant de-multiplexed address bits.

12.2.3 Parity Checking

Even or odd parity is generated for each byte on the Local Data Bus. While data is being read from SRAM, parity is checked if it is enabled in the corresponding configuration register. When the IOP 480 is the Local Bus Master, the bus width determines which bytes are checked for parity. When an external Master controls the Local Bus, the LBE[3:0]# signals determine which bytes are checked for parity. Parity errors can be programmed to cause an interrupt.

12.2.4 Boot PROM

After a reset, the registers for LCS0# default to settings which allow the internal IOP 480 CPU to boot from an external 8-bit EPROM on the Local Bus. The EPROM must use LCS0# as its chip select.

12.2.5 SRAM Examples

Figure 12-2 and Figure 12-3 illustrate examples of SRAM devices (two 64K x 16 and four 256K x 8 devices, respectively).

12.2.6 SRAM Write Access

An SRAM write access is started when ADS# is asserted, LWR# is high, and the address falls into the appropriate range. If the IOP 480 is controlling the access, the multiplexed address/data bus, LAD[19:2], is loaded into the MA[17:0] counter at the end of the period in which ADS# is asserted. If an external Local Bus Master is controlling the access, the multiplexed address/data bus, LAD[19:2], is loaded into the MA[17:0] counter at the end of the period in which ADS# is asserted.

After each transfer, MA[17:0] is incremented, thus pointing to the next 32-bit word. The MOE# signal is connected to the OE# input of the SRAMs, and disables the output drivers during a Write access. Burst accesses are not permitted to cross a 32 KB page boundary if the internal burst address is being used. If an external address counter is used, the BTERM# input can be used to terminate a burst when a page boundary is reached.

Five types of programmable timing parameters are associated with SRAM Write accesses, as illustrated in Table 12-3 on page 12-5.

As each word is written to memory, the Memory Controller generates an internal Ready signal, MEMRDY#, which signals the internal Local Bus controller that the transfer has occurred. MEMRDY# is based on the internal wait state generator, and is driven out onto the READY# pin when the READY# input pin is disabled in the Configuration registers.

If the Ready/Recover Mode bit is set in the Bus Region Descriptor Register, then the READY# pin is also driven during the recovery states. This can be used by an external Local Bus Master to delay the start of the next access until the recovery period expires.

If DRAM and Page Mode are enabled, the Memory controller does not assert CAS_MDQM[3:0]# during SRAM accesses. The WE# inputs in Figure 12-3 would need to be implemented in external logic.

If DRAM is also being used on the Local Bus, refresh accesses can cause an SRAM access to be delayed. This is a result of the sharing of Memory controller pins between SRAM and DRAM.

The timing parameter values shown at the top of each timing diagram refer to delay clock periods. This is not necessarily the value written to the corresponding field in the DRAMTIM register. The DRAMTIM register description details the mapping between the value written and resulting number of delay clock periods.

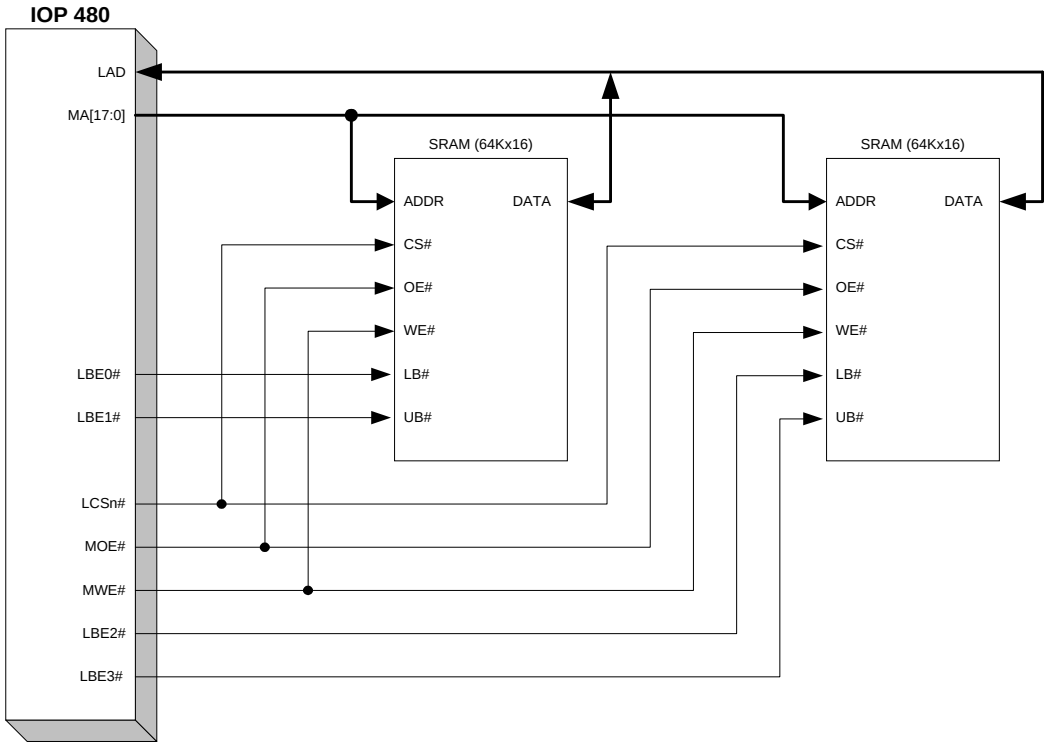


Figure 12-2. SRAM (Two 64K x 16 Devices)

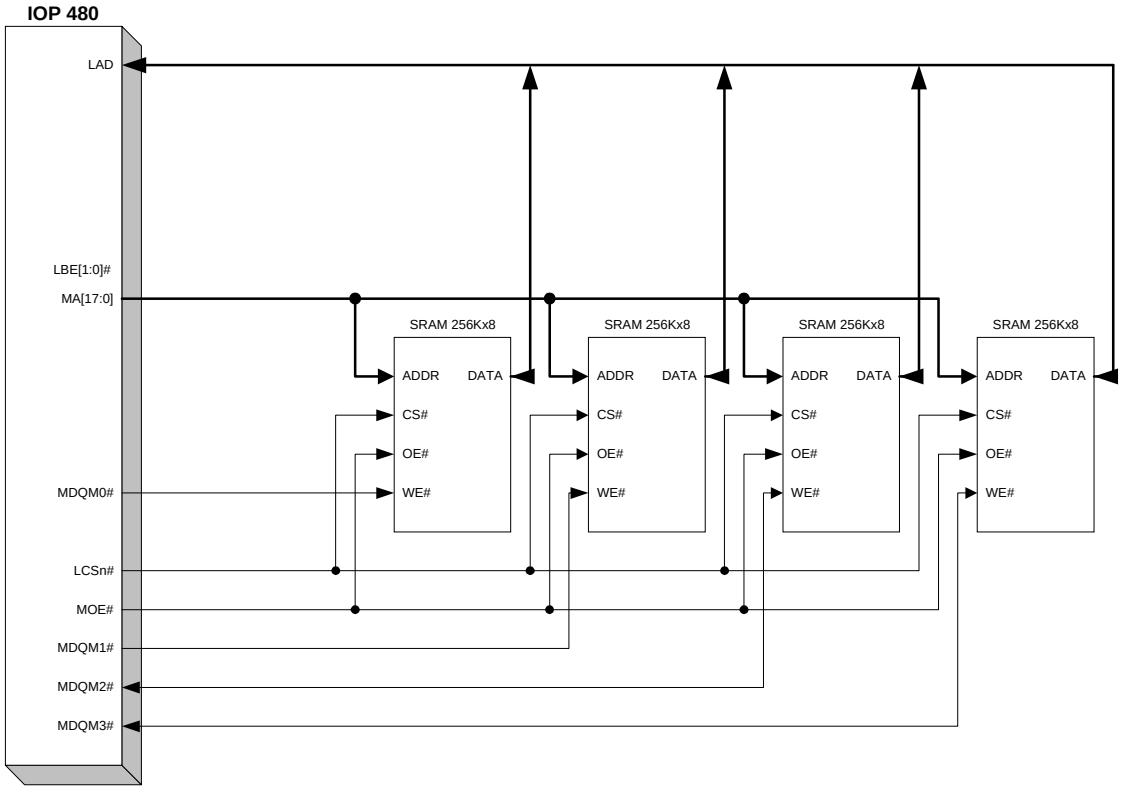


Figure 12-3. SRAM (Four 256K x 8 Devices)

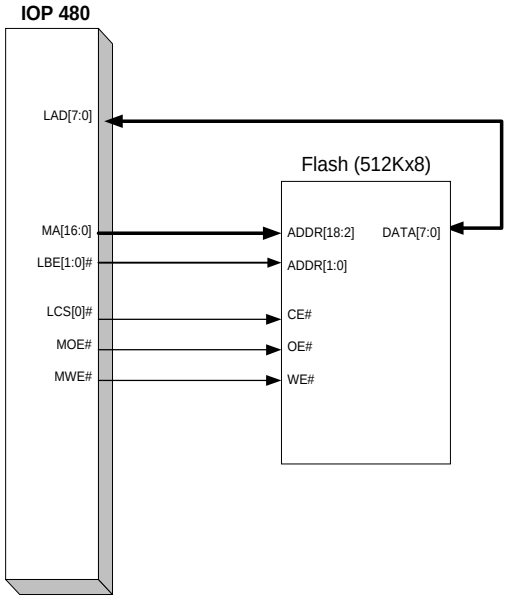
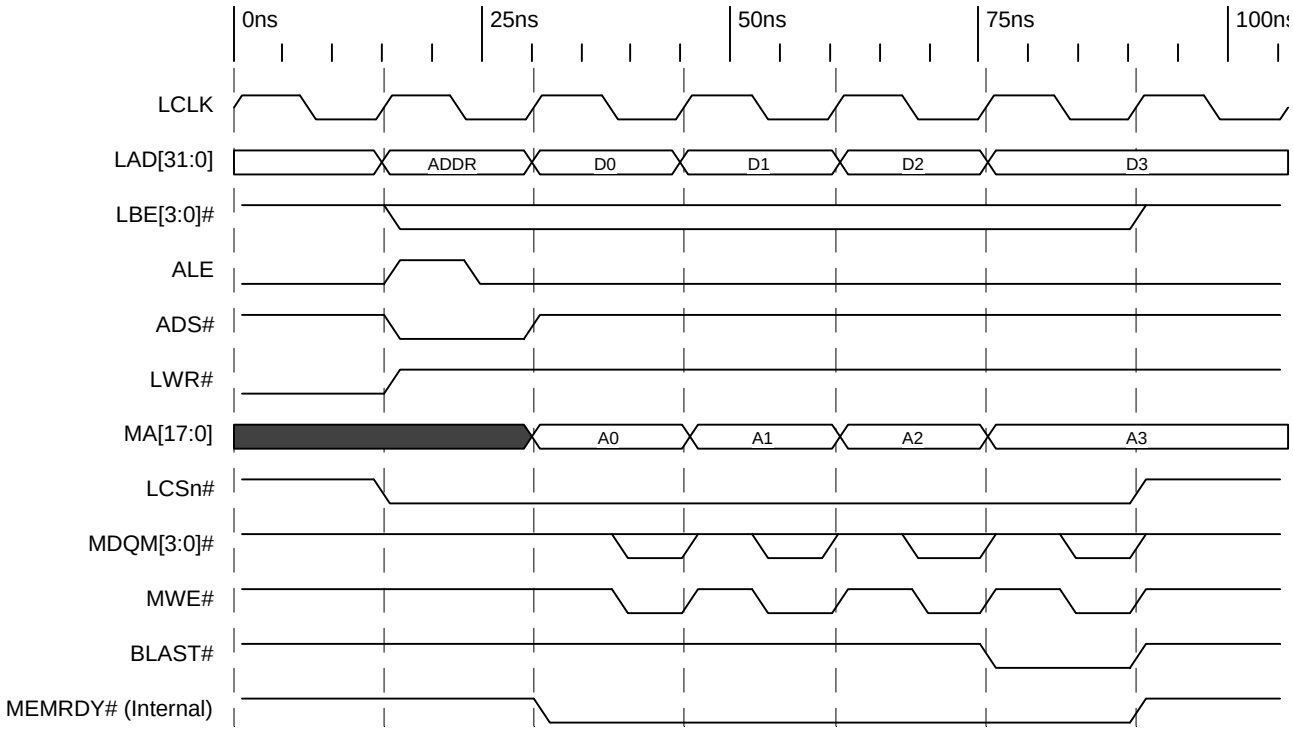


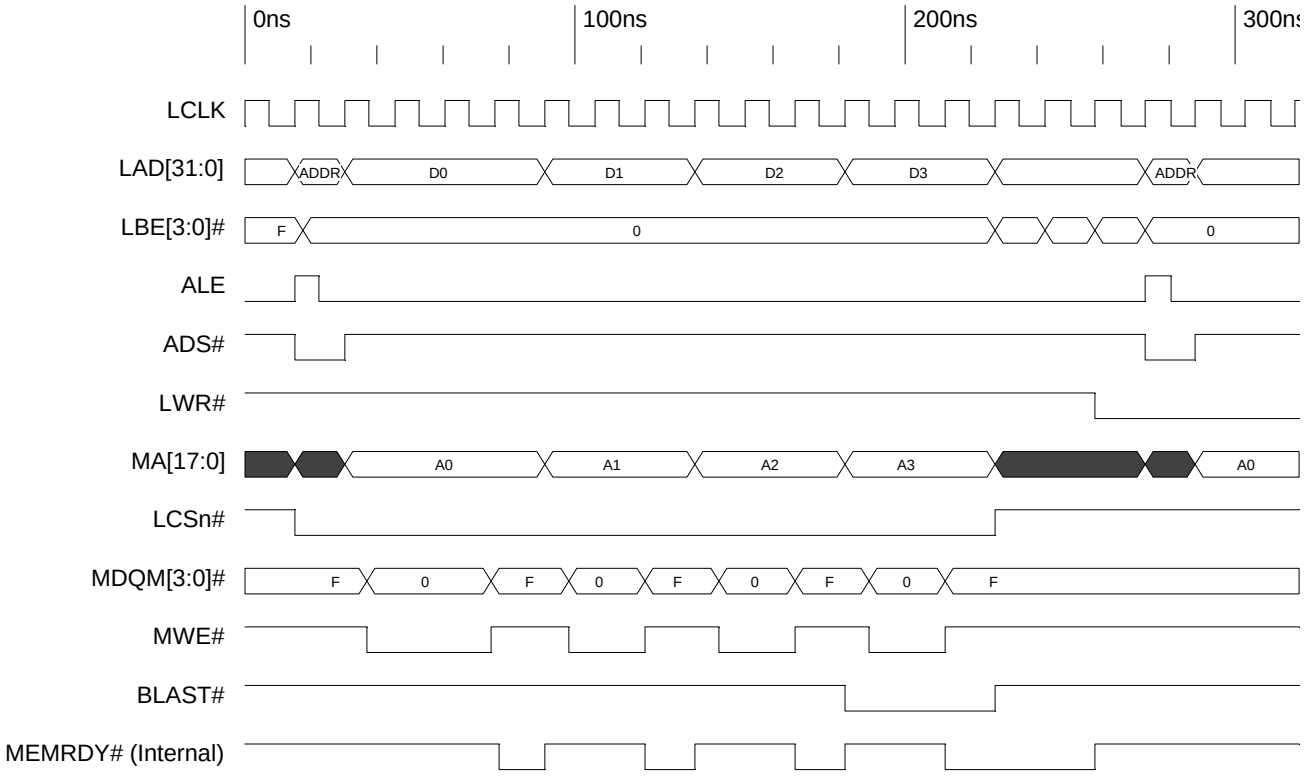
Figure 12-4. Flash (One 8-Bit Device)

Table 12-3. SRAM Write Access Programmable Timing Parameters

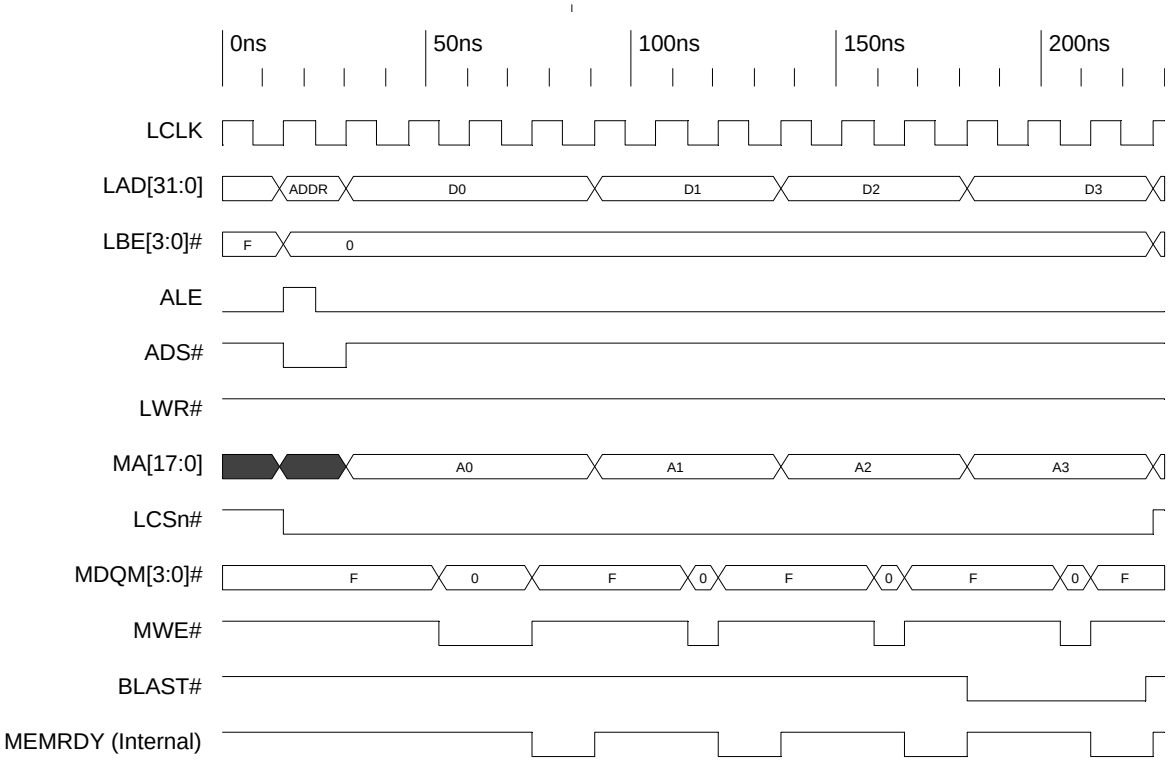
Field	Description
WAD	Number of write address-to-data wait states (0-15). Determines number of wait states for a single transfer, or for the first word of a burst transfer. In cases where the memory being accessed has some start-up latency associated with it, this number is set larger than data-to-data wait states.
WDD	Number of Write data-to-data wait states (0-15). Determines the number of wait states between burst data transfers. When this value is set to 0, data is transferred on every clock access if WDLY and WHLD are set to 0.
WDLY	Number of write enable delay states (0-7). Determines the delay from the assertion of the chip select until the write enables are first asserted. It also determines the delay from READY# to the re-assertion of the write enables during a burst access. When this value is set to 0, write strobes are asserted one-half clock period after chip select for external Bus Masters, and 1.5 clocks after the chip select when the IOP 480 is the Local Bus Master. If this field is set to a non-zero value, then WAD and WDD values must be set larger to accommodate the write strobe delay. This field does not cause wait states to be automatically added.
WHLD	Number of write hold states (0-7). Determines the number of hold states after the write enables have been de-asserted. During this period, address, data, and read/write signals are held stable. This field causes wait states to be added automatically, as well as those specified in WAD and WDD fields.
WRCV	Number of write recovery states (1-8). Determines the number of recovery states after a single write transfer or last data transfer of a burst.



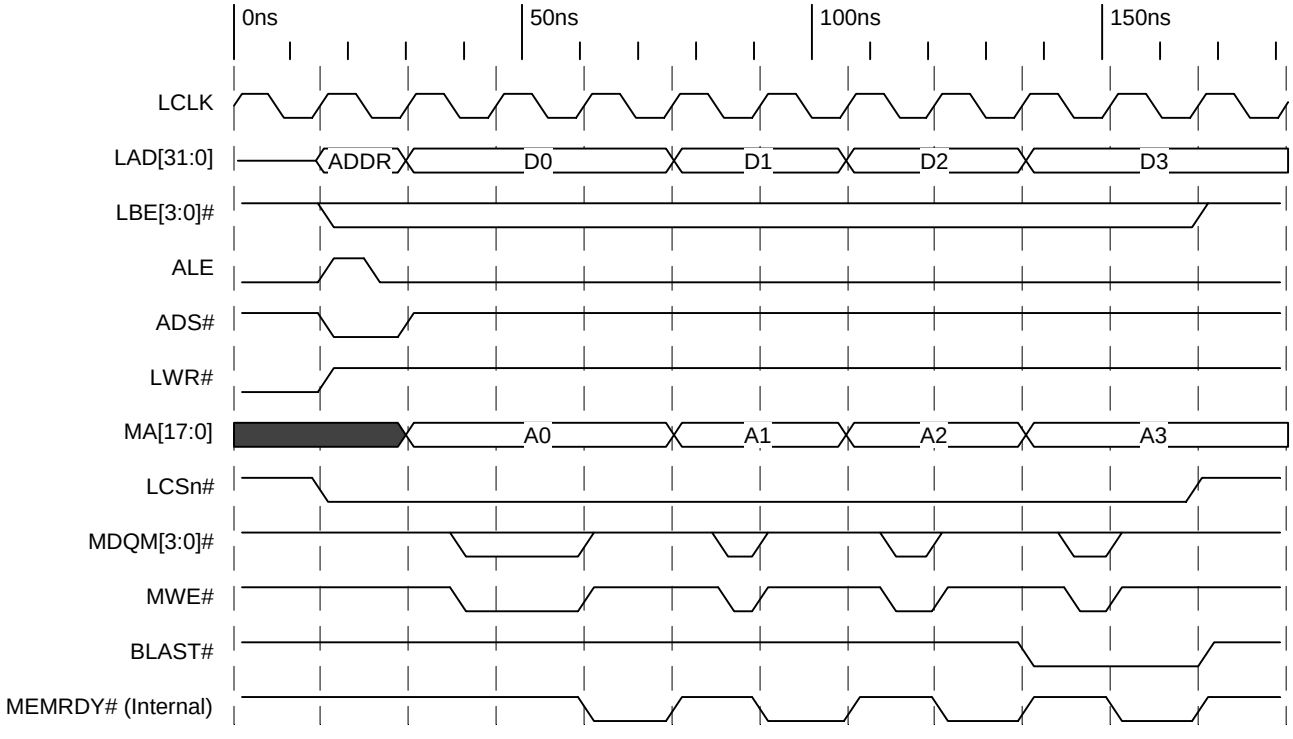
**Timing Diagram 12-1. SRAM Burst Write, 32-Bit Bus;
WAD=0, WDD=0, WDLY=0, WHLD=0, WRCV=0; Master=IOP 480**



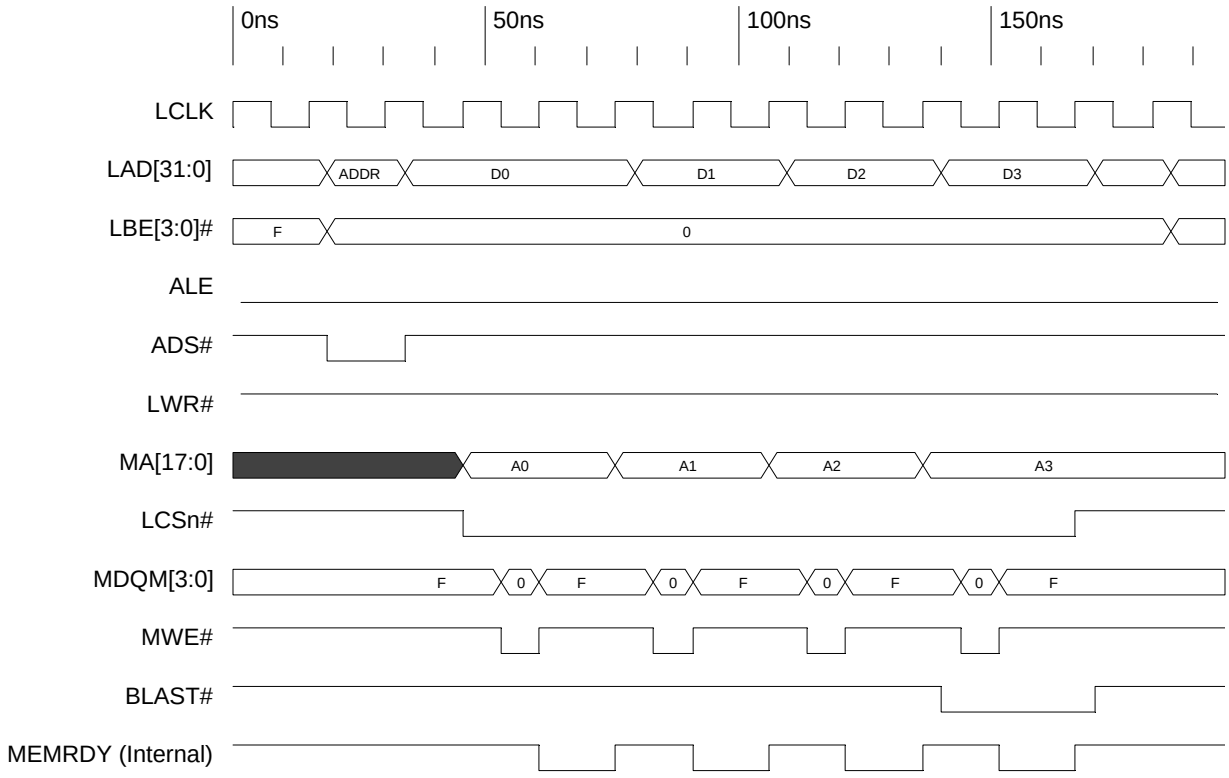
**Timing Diagram 12-2. SRAM Burst Write, 32-Bit Bus, 2-1-1-1 Wait States, 2 Recovery States;
WAD=2, WDD=1, WDLY=0, WHLD=1, WRCV=2; LCS x BRD [19]=1; Master=IOP 480**



Timing Diagram 12-3. SRAM Burst Write, 32-Bit Bus, 2-1-1-1 Wait States, 1 Write Enable Delay; WAD=2, WDD=1, WDLY=1, WHLD=1, WRCV=0; Master=IOP 480



Timing Diagram 12-4. SRAM Burst Write, 32-Bit Bus, 1 Write Hold Delay; WAD=1, WDD=0, WDLY=0, WHLD=1, WRCV=0; Master=IOP 480



Timing Diagram 12-5. SRAM Burst Write, 32-Bit Bus; WAD=0, WDD=0, WDLY=0, WHLD=1, WRCV=0; Master=External

12.2.7 SRAM Read Access

An SRAM read access is started when ADS# is asserted, LWR# is low, and address falls into the appropriate range. If the IOP 480 is controlling the access, the multiplexed address/data bus, LAD[19:2], is loaded into the MA[17:0] counter at the end of the period in which ADS# is asserted. If an external Local Bus Master is controlling the access, the multiplexed address/data bus, LAD[19:2], is loaded into the MA[17:0] counter at the end of the period in which ADS# is asserted.

After each transfer, MA[17:0] is incremented, thus pointing to the next 32-bit word. The MOE# signal is connected to OE# input of the SRAMs, and enables the output drivers during a read access. To prevent bus contention between the address and first data word, RAD and RDLYA are typically be programmed to a value of 1. Burst accesses are not permitted to cross a 32 KB page boundary if an internal burst address is being used. If an external address counter is used, the BTERM# input can be used to terminate a burst when page boundary is reached.

There are five types of programmable timing parameters associated with SRAM read accesses as illustrated in Table 12-4.

As each word is written to memory, the Memory Controller generates an internal Ready signal, MEMRDY#, which signals the internal Local Bus controller that the transfer has occurred. MEMRDY# is based on the internal wait state generator, and is driven out onto the READY# pin when READY# input pin is disabled in the Configuration registers.

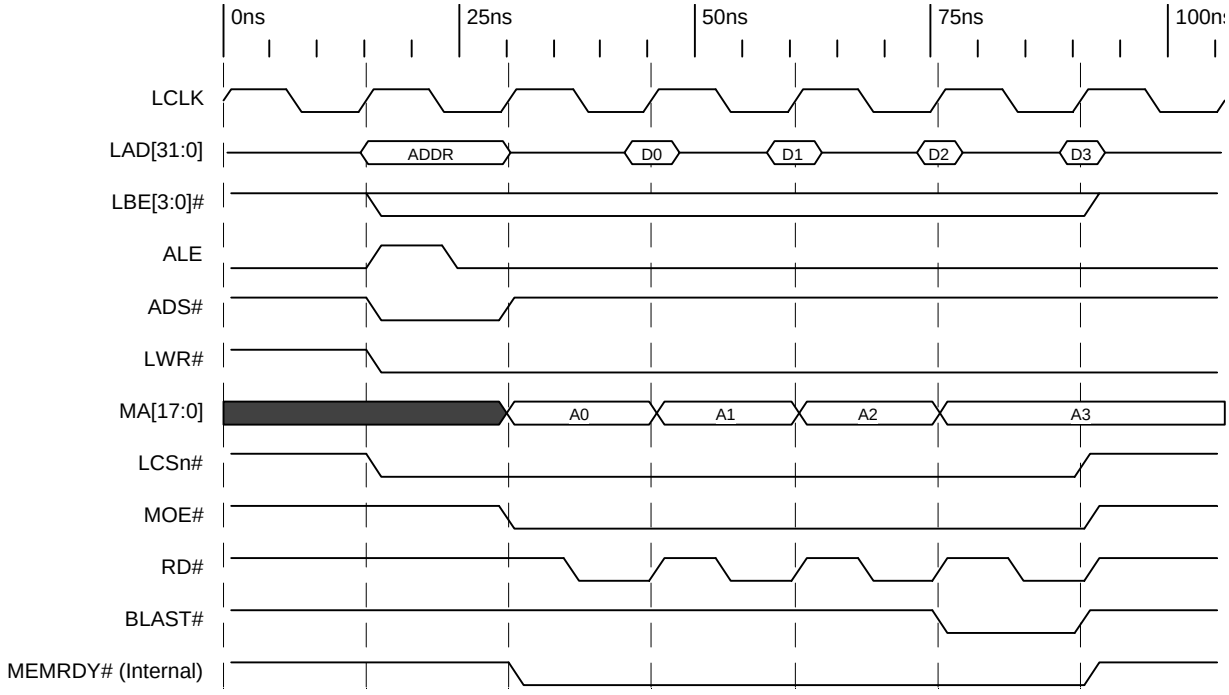
If the Ready/Recover Mode bit is set in the Bus Region Descriptor Register, then the READY# pin is also driven during the recovery states. This can be used by an external Local Bus Master to delay the start of the next access until the recovery period has expired.

If DRAM is also being used on the Local Bus, refresh accesses can cause an SRAM access to be delayed. This is a result of the sharing of Memory controller pins between SRAM and DRAM.

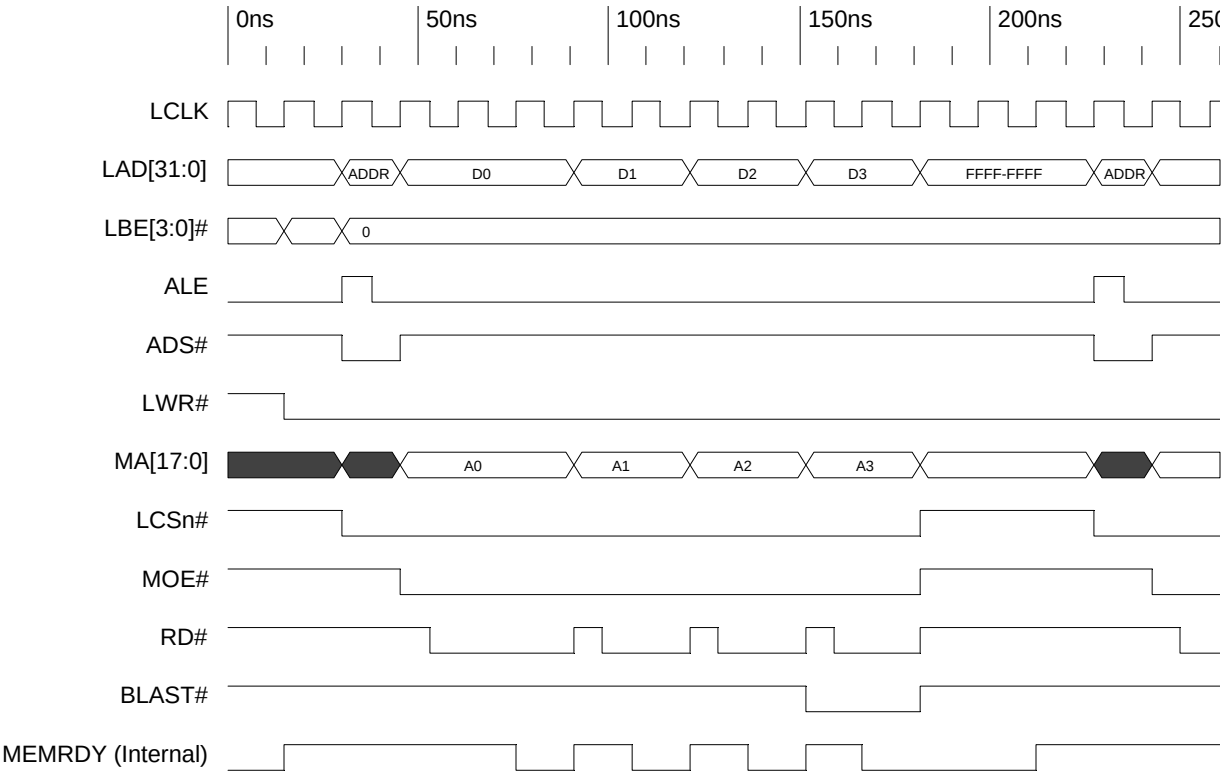
The timing parameter values shown at the top of each timing diagram refer to delay clock periods. This is not necessarily the value written to the corresponding field in the DRAMTIM register. The DRAMTIM register description details the mapping between the value written and resulting number of delay clock periods.

Table 12-4. SRAM Read Access Programmable Timing Parameters

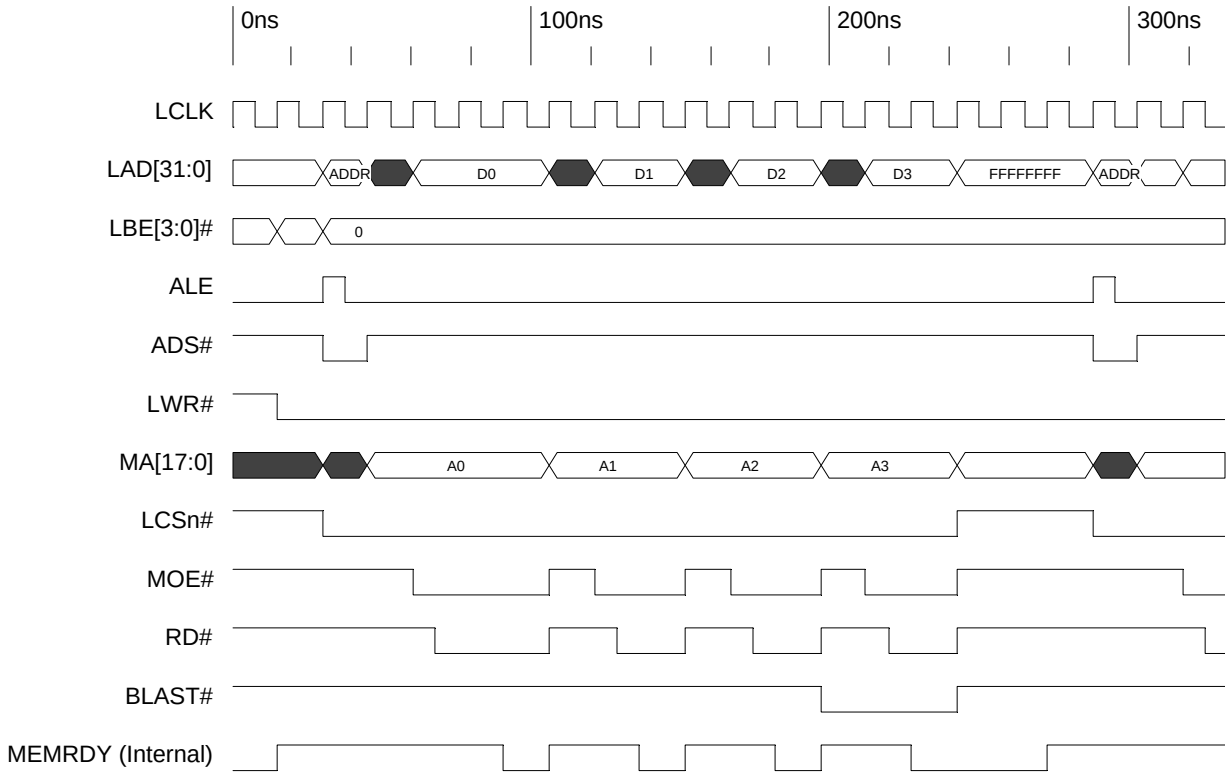
Field	Description
RAD	Number of read address-to-data wait states (0-15). Determines number of wait states for a single transfer, or for the first word of a burst transfer. In cases where the memory being accessed has some start-up latency associated with it, this number is set larger than the data-to-data wait states.
RDD	Number of Read data-to-data wait states (0-15). Determines number of wait states between burst data transfers. When this value is set to 0, data is transferred on every clock access.
RDLYA	Number of read enable delay states (0-7) for a single transfer or the first burst transfer. Determines delay from assertion of chip select until RD# and MOE# are first asserted. When this value is set to 0, RD# is asserted one-half clock access after the end of the address phase and MOE# is asserted at the end of the address phase. If this field is set to a non-zero value, then the RAD value must be set larger to accommodate read strobe delay. This field does not cause wait states to be added automatically.
RDLYD	Number of read enable delay states (0-7) for successive burst transfers. Determines the delay between de-assertion and re-assertion of RD# for successive transfers of a Burst Read access. When this value is set to 0, RD# is reasserted one-half clock cycle after it is de-asserted from the previous transfer. If this field is set to a non-zero value, then the RDD value must be set larger to accommodate the read strobe delay. This field does not cause wait states to be automatically added.
RRCV	Number of read recovery states (1-8). Determines the number of recovery states after a single read transfer or last data transfer of a burst. This field is used to provide the device with time to float its outputs before the next address appears on the bus.



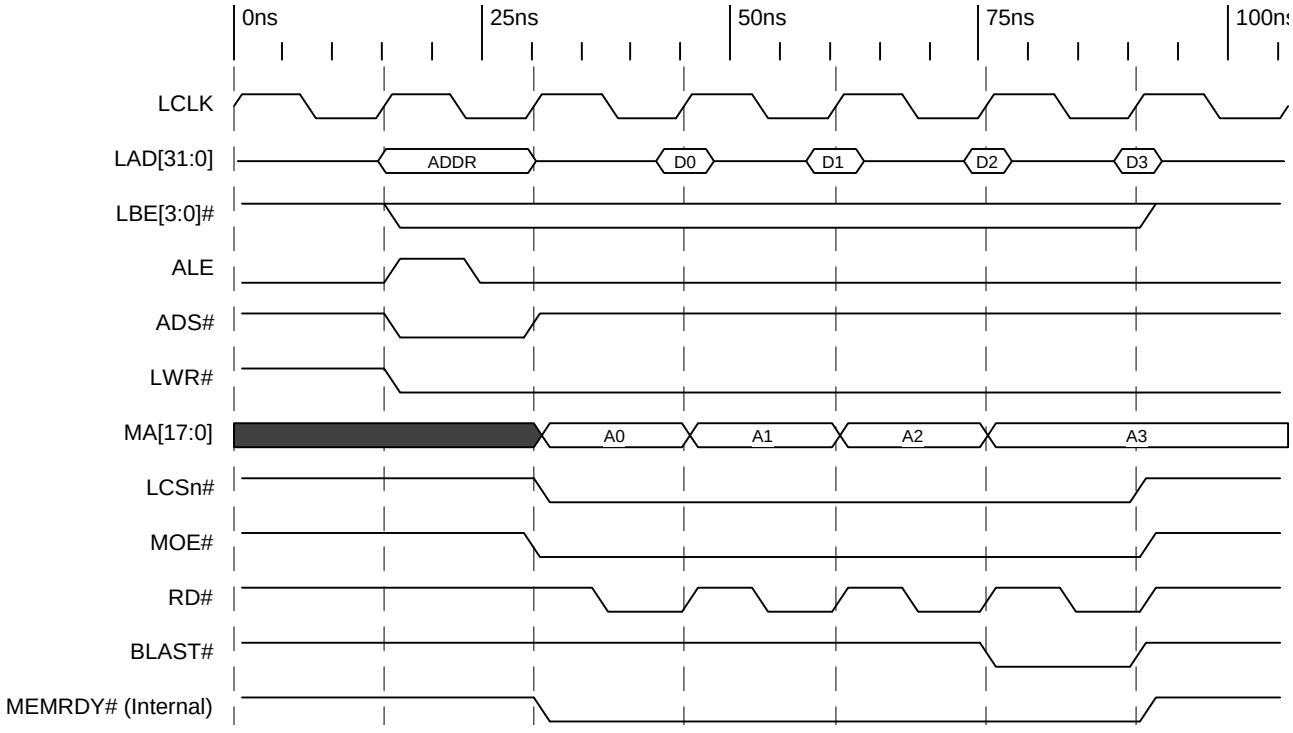
Timing Diagram 12-6. SRAM Burst Read, 32-Bit Bus, 0 Wait States; RAD=0, RDD=0, RDLYA=0, RDLYD=0, RRCV=0; Master=IOP 480



Timing Diagram 12-7. SRAM Burst Read, 32-Bit Bus, 2-1-1-1 Wait States, 2 Recovery States; RAD=2, RDD=1, RDLYA=0, RDLYD=0, RRCV=2; Master=IOP 480



Timing Diagram 12-8. SRAM Burst Read, 32-Bit Bus, 3-2-2-2 Wait States, 2 Recovery States; RAD=3, RDD=2, RDLYA=1, RDLYD=1, RRCV=2; Master=IOP 480



Timing Diagram 12-9. SRAM Burst Read, 32-Bit Bus, 0 Wait States; RAD=0, RDD=0, RDLYA=0, RDLYD=0, RRCV=0; Master=External Local Bus Master

12.3 DRAM

The Memory controller supports up to four banks of SDRAM or EDO DRAM. All four banks must use the same type of memory device. With 14 address lines, SDRAM or EDO DRAM banks can be up to 64 MB using 4-bit-wide devices. The following address, data, and control signals are used to access DRAM.

Table 12-5. DRAM Control Signals

Source	Signals	EDO DRAM	SDRAM
Memory Controller	RAS[3:0]# / MCS[3:0]#	Row Address Strobe. Used to strobe the row address into EDO DRAM, and as part of CAS-before-RAS refresh sequence. Each RAS enables one of up to four banks of EDO DRAM.	Chip select. Asserted to initiate read, write, and refresh accesses. Each chip select enables one of four banks of SDRAM.
	MRAS#	Unused.	Row Address Strobe. Asserted to initiate mode register set, precharge, activate, and refresh accesses.
	MCAS# / MOE#	Output Enable. Used to enable the DRAM output buffers during a read access.	Column Address Strobe. Asserted to initiate mode register set, read, write, and refresh access.
	CAS[3:0]# / MDQM[3:0]#	Column Address Strobe. Used to strobe the column address into the EDO DRAM. Each CAS# signal corresponds to one byte in the 32-bit-wide memory.	Input Mask, Output Enable. Used to select bytes during write and read accesses.
	MWE#	Write Enable. Asserted during write accesses.	Write Enable. Asserted to initiate mode register set, precharge, and memory write accesses.
	MA13 / LCS3#, MA[12:0]	Address. 14-bit multiplexed address allows access to 64 MB of EDO DRAM per bank. Eight 16 MB x 4 devices = 64 MB Four 8 MB x 8 devices = 32 MB Two 4 MB x 16 devices = 16 MB Eight 4 MB x 4 devices = 16 MB Four 2 MB x 8 devices = 8 MB Two 1 MB x 16 devices = 4 MB	Address. 14-bit multiplexed address allows access up to 64 MB of SDRAM per bank. Eight 16 MB x 4 devices = 64 MB Four 8 MB x 8 devices = 32 MB Two 4 MB x 16 devices = 16 MB Eight 4 MB x 4 devices = 16 MB Four 2 MB x 8 devices = 8 MB Two 1 MB x 16 devices = 4 MB Also used to provide mode register data during initialization.
Local Bus Controller	MCKE	Unused.	Memory Clock Enable. Used to place memory into a suspend state for low power operation.
Local Bus Master	LCLK	Unused.	Clock. Use local clock as the SDRAM clock. All controls, address, and data are synchronous to this clock.

12.3.1 Synchronous DRAM (SDRAM)

12.3.1.1 SDRAM Signal Connections

Table 12-6 shows the connections between the IOP 480 memory signals and signal names found in most SDRAM specifications.

Table 12-6. SDRAM Signals

SDRAM Signal	IOP 480 Signal	Description
A[11:0]	MA[11:0]	Multiplexed Address Bus
BA0	MA12	Bank Address bit 0
BA1	DP0 / MA13	Bank Address bit 1
DQ[31:0]	LAD[31:0]	Data Bus
CLK	LCLK	Local Bus Clock
CKE	MCKE	Clock Enable
CS#	MCS[3:0]#	Chip Select
RAS#	MRAS#	Row Address Strobe
CAS#	MCAS#	Column Address Strobe
WE#	MWE#	Write Enable
DQM[3:0]#	MDQM[3:0]#	I/O Mask (Byte Enables)

12.3.1.2 SDRAM Example

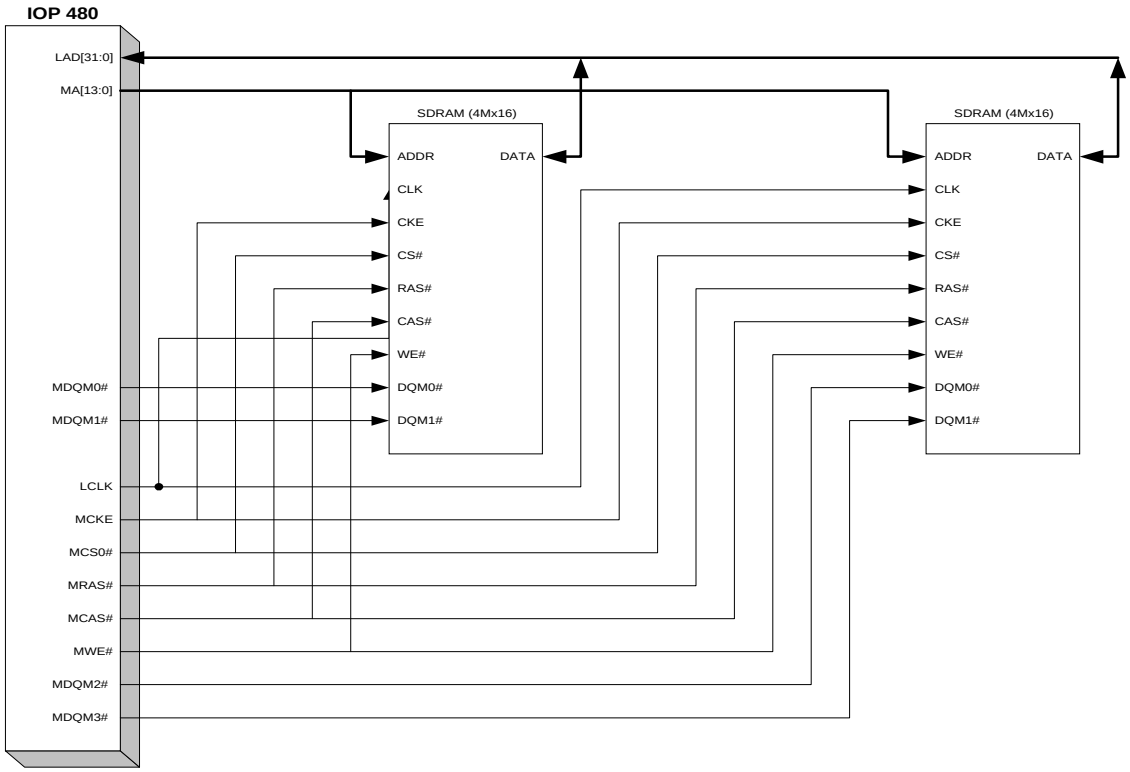


Figure 12-5. SDRAM (Two 4M x 16 Devices)

12.3.1.3 SDRAM Addressing

The multiplexed Memory Address Bus, MA[13:0], is connected to the following Local Address Bus signals for row, column, and bank address. Address bits marked with an asterisk are unused in DRAM.

Note: *For 64-megabit SDRAMs, DP0/MA13 is required as a bank select, thus preventing the use of data parity.*

Table 12-7. 16-Megabit SDRAM Addressing

MA Bus	1M x 16		2M x 8		4M x 4		Mode Register
	Row(11)	Col(8)	Row(11)	Col(9)	Row(11)	Col(10)	
0	10	2	11	2	12	2	1
1	11	3	12	3	13	3	1 (Full Page burst)
2	12	4	13	4	14	4	1
3	13	5	14	5	15	5	0 (Sequential burst)
4	14	6	15	6	16	6	0
5	15	7	16	7	17	7	1 (CAS Latency of 2)
6	16	8	17	8	18	8	0
7	17	9	18	9	19	9	0
8	18	0	19	10	20	10	0
9	19	0	20	0	21	11	0
10	20	0	21	0	22	0	0
11	21 = Bank Address		22 = Bank Address		23 = Bank Address		0
12	22*		23*		24*		0
13	23*		24*		25*		0

Table 12-8. 64-Megabit SDRAM Addressing

MA Bus	4M x 16		8M x 8		16M x 4		Mode Register
	Row(12)	Col(8)	Row(12)	Col(9)	Row(12)	Col(10)	
0	10	2	11	2	12	2	1
1	11	3	12	3	13	3	1 (Full Page burst)
2	12	4	13	4	14	4	1
3	13	5	14	5	15	5	0 (Sequential burst)
4	14	6	15	6	16	6	0
5	15	7	16	7	17	7	1 (CAS Latency of 2)
6	16	8	17	8	18	8	0
7	17	9	18	9	19	9	0
8	18	0	19	10	20	10	0
9	19	0	20	0	21	11	0
10	20	0	21	0	22	0	0
11	21	0	22	0	23	0	0
12	22 = BA0		23 = BA0		24 = BA0		0
13	23 = BA1		24 = BA1		25 = BA1		0

12.3.1.4 SDRAM Initialization

After the SDRAMs are powered up and have a stable clock, no commands are issued for at least 200 µs. This timeout is based on 13200 clock ticks at Local Bus clock frequency of 66 MHz. If a slower Local Bus clock is used, then the power-on delay increases. The delay is calculated as follows: delay = (1/Local Bus clock frequency) x 13200. All of the command signals (MRAS#, MCAS#, MWE#, MCS[3:0]#, and MDQM[3:0]#) are held in an inactive state during this period. After the 200 µs delay, a precharge command is initiated with MCKE asserted, precharging both banks. Now the SDRAMs are ready to have their mode registers loaded with the operating mode.

The mode register is used to define specific operating modes of SDRAM, and is programmed with the LOAD MODE REGISTER command. The mode register is loaded when both banks are idle. The controller must wait the specified time before initiating subsequent operations. The value to be programmed into the mode register is presented to SDRAM on the multiplexed address bus MA[13:0]. A configuration register in the IOP 480 provides the capability to reload the mode register into the SDRAMs if the default values are not appropriate for the application. The Memory controller uses the current value of the SDRAM Initialization Word in the DRAMINIT register to adjust for different values of CAS latency. When the DRAMINIT register SDRAM Initialization Order bit is low, the mode register is loaded first, followed by eight Auto-Refresh accesses. When the DRAMINIT register SDRAM Initialization Order bit is high, the eight Auto-Refresh accesses are run first, followed by the mode register load. The SDRAM is now ready to accept an activate command. Following is a description of the mode register.

12.3.1.4.1 Burst Length

The Burst length determines the maximum number of columns that can be accessed during a Read or Write burst. When a read or write command is issued, a block of columns equal to the burst length is effectively selected. For the IOP 480, the burst length in SDRAMs should be set to “full-page.” In this mode, a burst with an arbitrary length is supported. Full-page bursts wrap within the page if the boundary is reached. All burst accesses are sequential for full-page Burst mode, and bursts cannot cross a page boundary. Page lengths and boundaries are illustrated in Table 12-9.

12.3.1.4.2 Burst Type

Burst accesses may be sequential or interleaved. For the IOP 480, the type should always be set to sequential.

12.3.1.4.3 CAS Latency

CAS latency is the delay from the Read command to the first valid Read data, and can be set to 1, 2, or 3. For Local Bus speeds between 33 and 66 MHz, set CAS latency to 2.

12.3.1.4.4 Operating Mode

The operating mode should always be set to standard operation. Other values are for testing purposes only.

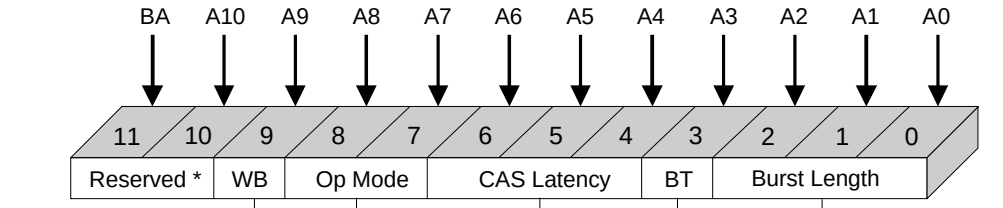
12.3.1.4.5 Write Burst Mode

The Write Burst mode determines whether Write accesses are Burst or Single-access only. If this bit is 0, both Read and Write accesses can be Burst accesses. If this bit is 1, then only Read accesses can be Burst accesses.

Figure 12-6 is a diagram showing the bit locations of the mode register.

Table 12-9. SDRAM Page Lengths/Boundaries

Number of Columns	Max Page Length	Page Boundary
8 (x 16 devices)	1024	1024
9 (x 8 devices)	2048	2048
10 (x 4 devices)	4096	4096



* Should program
M11, M10 = 0, 0
to ensure compatibility
with future devices

M2 M1 M0	Burst Length	
	M3 = 0	M3 = 1
0 0 0	1	1
0 0 1	2	2
0 1 0	4	4
0 1 1	8	8
1 0 0	Reserved	Reserved
1 0 1	Reserved	Reserved
1 1 0	Reserved	Reserved
1 1 1	Full Page	Reserved

M3	Burst Type
0	Sequential
1	Interleave

M6 M5 M4	CAS Latency
0 0 0	Reserved
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	Reserved
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

M8	M7	M6-M0	Operating Mode
0	0	Defined	Standard Operation
–	–	–	All other states reserved

M9	Write Burst Mode
0	Programmed Burst Length
1	Single Location Access

Figure 12-6. SDRAM Mode Register

12.3.1.5 SDRAM Commands

The following table lists commands that can be issued to SDRAMs. Signal names relate to the SDRAM device specifications.

Table 12-10. SDRAM Commands

Command	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQs	Notes
COMMAND INHIBIT (NOP/continue previous operation)	H	X	X	X	X	X	X	—
NO OPERATION (NOP/continue previous operation)	L	H	H	H	X	X	X	—
ACTIVE (Select bank and activate row)	L	L	H	H	X	bank/row	X	3
READ (Select bank and column and start READ burst)	L	H	L	H	X	bank/col	X	4
WRITE (Select bank and column and start WRITE burst)	L	H	L	L	X	bank/col	Valid	4
BURST TERMINATE	L	H	H	L	X	X	Active	—
PRECHARGE (Deactivate row in bank or banks)	L	L	H	L	X	Code	X	5
AUTO-REFRESH or SELF-REFRESH (enter Self-Refresh mode)	L	L	L	H	X	X	X	6, 7
LOAD MODE REGISTER	L	L	L	L	X	Op-code	X	2
Write enable/output enable	—	—	—	—	L	—	Active	8
Write inhibit/output High-Z	—	—	—	—	H	—	High-Z	8

Table Notes:

1. CKE is high for all commands shown except Self-Refresh.
2. **16-Megabit Device:** A0 through A10 and BA defines the op-code written to the mode register.

   64-Megabit Device: A0 through A11 defines the op-code written to the mode register.
3. **16-Megabit Device:** A0 through A10 provide the row address, and BA determines which bank is made active.

   (BA Low → Bank 0, BA HIGH → Bank 1).

   64-Megabit Device: A0 through A11 provide the row address, and BA0, BA1 determine which bank is made active.
4. **16-Megabit Device:** A0 through A9 provide the column address (A9 is a “don’t care” for x8 devices and A8,9 are “don’t cares” for x16 devices). A10 High enables the Auto Precharge feature (nonpersistent) while A10 Low disables the Auto Precharge feature. BA determines which bank is being read from or written to (BA Low → Bank 0, BA HIGH → Bank 1).

   64-Megabit Device: A0 through A9 provide the column address (A9 is a “don’t care” for x8 devices and A8,9 are “don't cares” for x16 devices). A10 High enables the Auto Precharge feature (nonpersistent), while A10 Low disables the Auto Precharge feature. BA0 and BA1 determines which bank is being read from or written to.
5. **16-Megabit Device:** A10 LOW: BA determines bank being precharged

   (BA Low → Bank 0, BA HIGH → Bank 1)

   A10 HIGH: both banks precharged and BA is a “don’t care.”

   64-Megabit Device: A10 LOW: BA0, BA1 determine the bank being precharged. A10 HIGH: All banks precharged and BA0 and BA1 are “don’t care.”
6. This command is Auto Refresh if CKE is high, and Self-Refresh if CKE is low.
7. Internal refresh counter controls row addressing; all inputs and
   I/Os are “don’t care” except for CKE.
8. Activates or deactivates the DQs during WRITEs (zero-clock delay) and READs (two-clock delay).

12.3.1.5.1 Command Inhibit

Command Inhibit prevents new commands from being executed and the SDRAM is effectively deselected. Operations in progress continue.

12.3.1.5.2 No Operation

No Operation prevents new commands from being executed, while the SDRAM is selected (CS# low). This prevents commands from being registered during idle or waits states. Operations in progress continue.

12.3.1.5.3 Active

Active activates a row for subsequent accesses. This row remains active until a precharge command is issued—precharge command must be issued before activating a different row in the same bank.

12.3.1.5.4 Read

Read initiates a burst read access to the active row. The value on BA inputs selects the bank, while the address on A0-9 selects the starting column. The value on A10 determines whether AUTO PRECHARGE is used. If AUTO PRECHARGE is selected, then the row being accessed is precharged at the end of burst. If AUTO PRECHARGE is not selected, the row remains open for access. Read data appears on the outputs after the CAS latency period, and subject to the value on DQM inputs two clocks earlier (the output enable latency on DQM bits is two clock accesses).

12.3.1.5.5 Write

Write initiates a burst write access to the active row. The value on BA inputs selects the bank, and the address on A[9:0] selects the starting column. The value on A10 determines whether AUTO PRECHARGE is used. If AUTO PRECHARGE is selected, then the row being accessed is precharged at the end of burst. If AUTO PRECHARGE is not selected, the row remains open for access. Data is written to memory if the DQM inputs are active coincident with the valid data. The first word to be written can be presented coincident with the Write command.

12.3.1.5.6 Burst Terminate

Burst Terminate terminates only full-page bursts.

12.3.1.5.7 Precharge

Precharge deactivates open rows in one or both banks. If input A10 is low, BA determines which bank is precharged. If input A10 is high, all banks are precharged. Once a bank has been precharged, it remains in the idle state and must be activated prior to new READ or WRITE commands to that bank.

12.3.1.5.8 Auto-Precharge

Auto-Precharge performs a precharge without an explicit precharge command, but is not used in full-page Burst mode.

12.3.1.5.9 Auto-Refresh

Auto-Refresh is analogous to CAS-before-RAS refresh in EDO DRAMs. Refresh request must be explicitly made. The refresh address is generated internally. A refresh access must be run every 15.6 µs in order to refresh 4096 rows every 64 ms.

12.3.1.5.10 Self-Refresh

Self-Refresh retains data when the rest of the system is powered down.

Use the following procedure to enter Self-Refresh mode:

- Issue a burst of 4096 Auto-Refresh accesses
- Bring CKE, CS#, RAS#, CAS# low and WE# high for one clock period
- Keep CKE low; other inputs are “don’t cares”

During Self-Refresh, SDRAM provides its own internal clocking, refresh requests, and address. The following procedure must be used to exit Self-Refresh mode:

- Wait for CLK to stabilize
- Bring CKE high
- Issue NOP commands for t_{XSR} time (at least 96 ns)
- Issue a burst of 4096 auto refresh accesses

12.3.1.6 Operation

12.3.1.6.1 Page Mode

Once a row is activated, any column within the current row can be accessed for reads or writes by asserting CAS# with the appropriate column address. If the Page Mode bit in the DRAM Configuration registers is enabled, no PRECHARGE command is asserted after the end of an access. During the Address phase of the next access, the new row address is compared with previous row address. If they match, a page hit has occurred and the column can be addressed without activating a new row. If a page miss occurs, then the

device must precharge and a new row activated. If the application tends towards sequential accesses, then enabling Page mode improves performance. Random accesses achieve the best performance by having Page mode disabled. There is only one Page mode row address register that is shared between the four banks of SDRAMs. Therefore, only one row out of all of the banks can be kept active when Page mode is enabled. If a page miss occurs, all banks are precharged. If Page mode is disabled, all banks are precharged at the end of every single or burst access. If the Page mode is enabled, then disabled, the SDRAM needs to be re-initialized by writing to the DRAMINIT register.

12.3.1.6.2 Parity Checking

Even or odd parity is generated for each byte on the Local Data Bus. The Parity Select bit in DRAMBRD register selects polarity of the parity. When data is being read from SDRAM, parity is checked if it is enabled in the corresponding configuration register. When the IOP 480 is the Local Bus Master, all four bytes are checked for parity. When an external Master controls the Local Bus, the LBE[3:0]# signals determine which bytes are checked for parity. Parity errors can be programmed to cause an interrupt.

12.3.1.6.3 Burst Length

The length of a burst access depends upon the Master accessing SDRAM, and number of columns in SDRAM. When the Direct Slave interface or internal DMA controller is accessing SDRAM, a Burst read can be of any length, as long as it fits within a page of SDRAM. When the internal IOP 480 CPU is accessing SDRAM, burst are limited to a maximum of four, 32-bit sequential words. Local Bus Masters can perform Burst accesses of any length, up to the maximum page length of the SDRAM without crossing a page boundary.

12.3.1.6.4 Auto-Refresh

An Auto-Refresh access is run at a rate determined by the DRAMCTL Refresh Interval bit(s) and the Local Bus clock frequency. A typical refresh rate for most DRAMs is 15.625 µs, and is calculated as follows:

register value = refresh rate x LCLK clock frequency.

Typically, the register value = 15.625 µs x 66.666 MHz = 1041 = 0x411. The value after reset is 0x407.

The Refresh access has priority over other DRAM requests. If a refresh request occurs during an SDRAM burst access, the burst is preempted. The refresh address is generated internally in SDRAM devices when using Auto-Refresh.

If a Refresh request occurs during a DRAM Burst access, then the Master of the Burst access is preempted.

Table 12-11. Refresh Arbitration

Master	Action
Internal IOP 480 CPU	Wait for end of burst (limited to four words)
Direct Slave Controller	Assert BLAST# and terminate burst
DMA Controllers	Assert BLAST# and terminate burst
LHOLDREQ0	Assert BOFF#
LHOLDREQ1	None

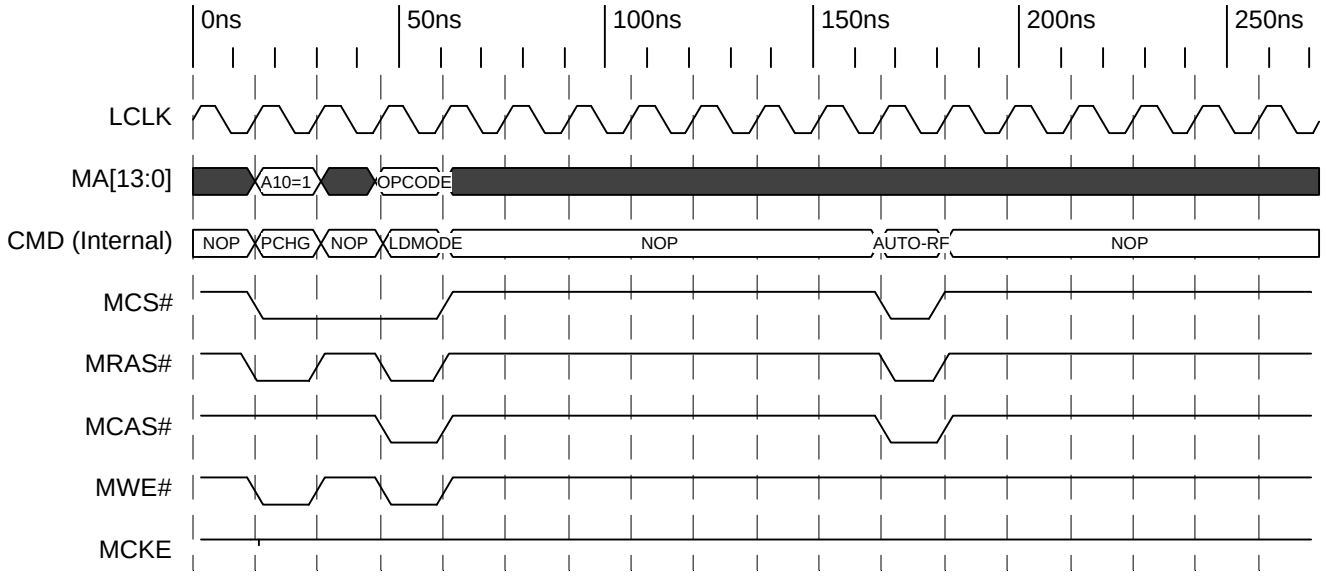
The following table lists the commands to perform Auto-Refresh.

Table 12-12. Auto Refresh Commands

Command	Description
PRECHARGE	All banks if any rows are active
NOP	To satisfy t_{RP}; precharge command period
AUTO-REFRESH	Issues auto-refresh command
7 NOPs	To satisfy t_{RC}; AUTO-REFRESH to ACTIVE delay

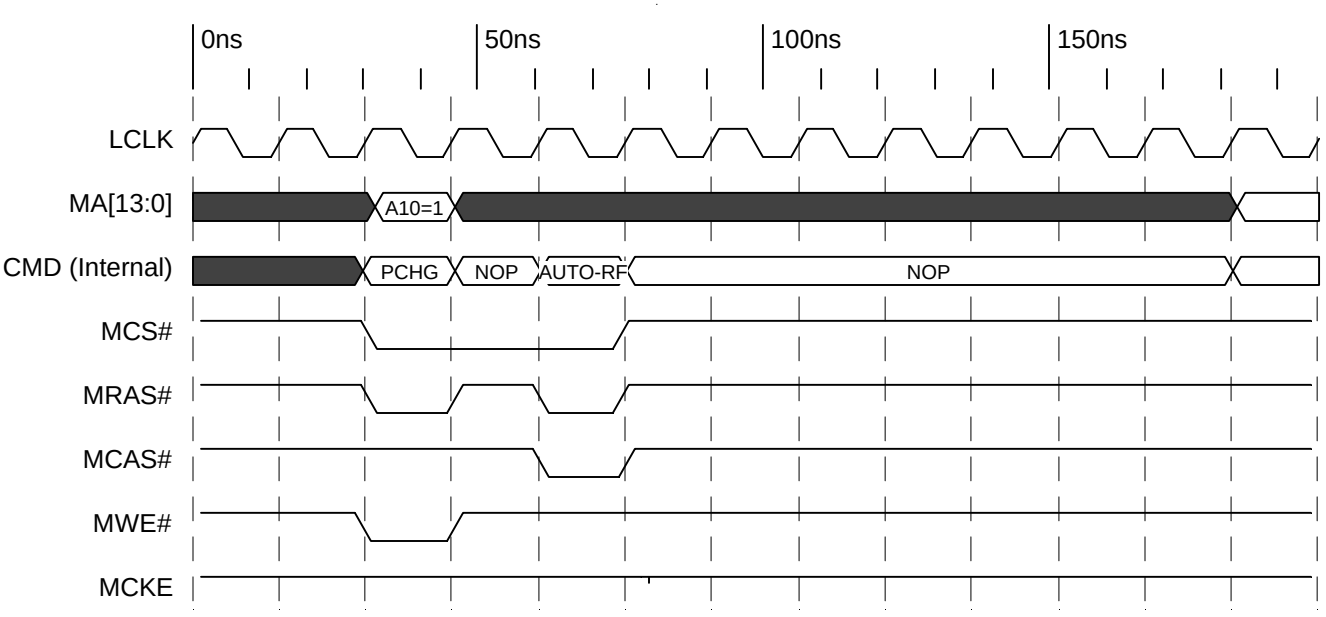
12.3.1.6.5 Self-Refresh

Self-Refresh retains system data when the rest of the system is powered down. It is initiated when the Power Management State matches the state specified in the Self-Refresh field of the DRMCTL Configuration register.

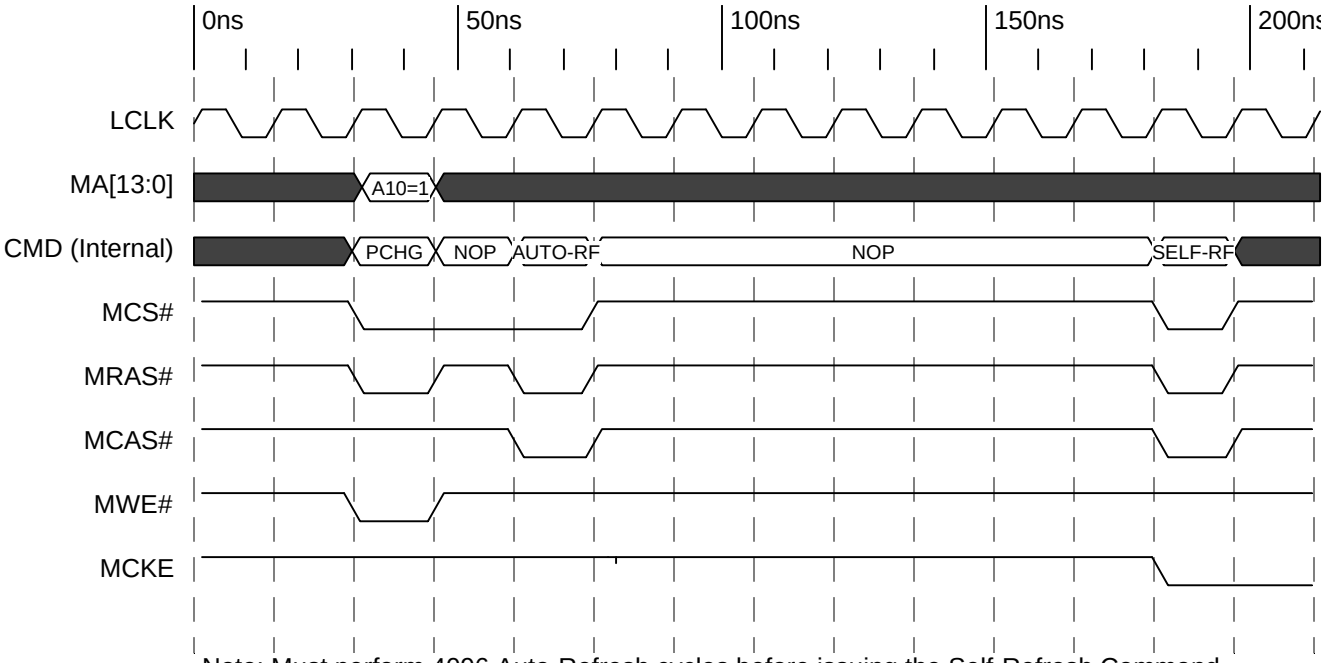


Note 1: Controller waits for 200 us after Vcc and Clock are stable before issuing the Precharge Command.
Note 2: Controller issues eight Auto-Refresh cycles after the Load Mode command is issued.

Timing Diagram 12-10. SDRAM Initialization

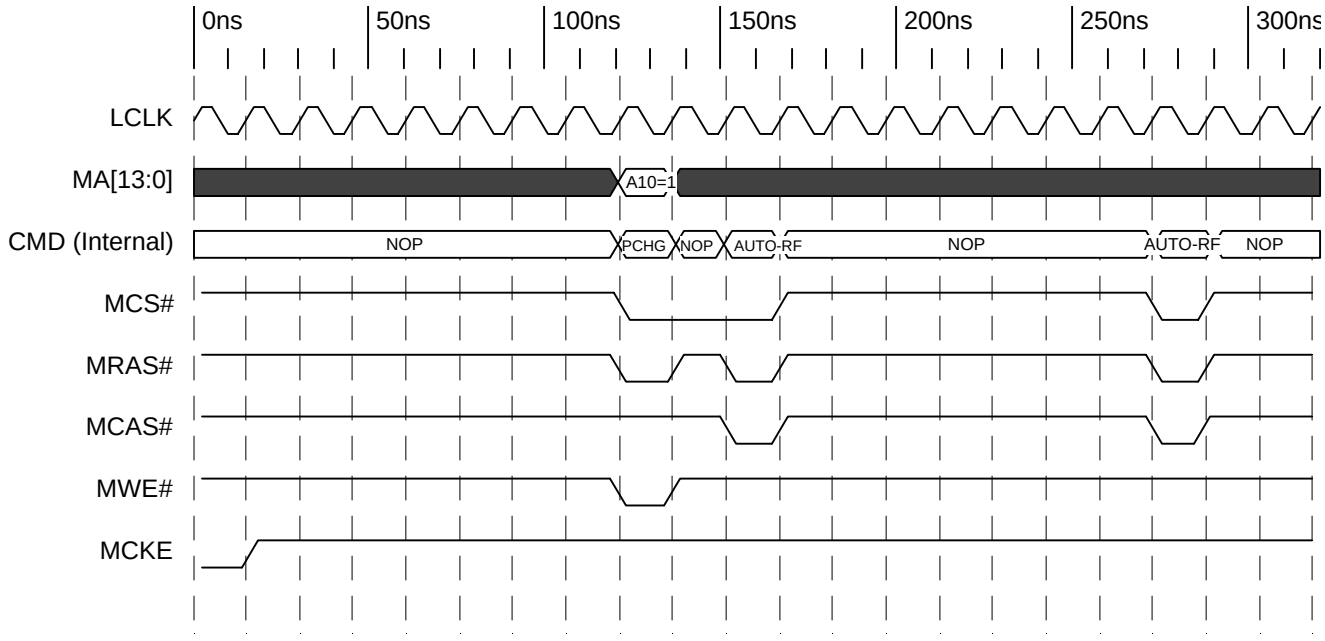


Timing Diagram 12-11. SDRAM Auto-Refresh



Note: Must perform 4096 Auto-Refresh cycles before issuing the Self-Refresh Command.

Timing Diagram 12-12. SDRAM Self-Refresh Start



Note: Must perform 4096 Auto-Refresh cycles after MCKE is asserted.

Timing Diagram 12-13. SDRAM Self-Refresh End

12.3.1.6.6 SDRAM Initialization

The Mode register is loaded at initialization, and is used to set the operating parameters of SDRAM.

12.3.1.6.7 SDRAM Write Access

An SDRAM write access starts when ADS# is asserted, LWR# high, and address falls into the appropriate range. If the IOP 480 is controlling the access, the row address is loaded into the MA[13:0] counter at beginning of the period in which ADS# is asserted. If an external Local Bus Master is controlling the access, the row address is loaded into the MA[13:0] counter at the end of the period in which ADS# is asserted.

During the next clock period, a row is activated by assertion of one of the MCS[3:0]# signals and MRAS#. The ACTIVE to WRITE delay is determined by the A2C value in the DRAM Timing Configuration register. The MA[13:0] Bus is switched to the column address as soon as the ACTIVE command is registered. The WRITE command is initiated by asserting MCAS#, MWE#, and the appropriate MDQM[3:0]# bits. Each MDQM[3:0]# signal corresponds to one of the byte lanes (DQM3# → LAD[31:24],…, DQM0# → LAD[7:0]). The first word of data is written into SDRAM during this clock cycle. As long as the MDQM[3:0]# bits remain asserted, data is written on each successive cycle. When BLAST# is detected, the Burst write is terminated by issuing a PRECHARGE command.

The MDQM[3:0]# bits are also de-asserted, thus preventing unwanted writes.

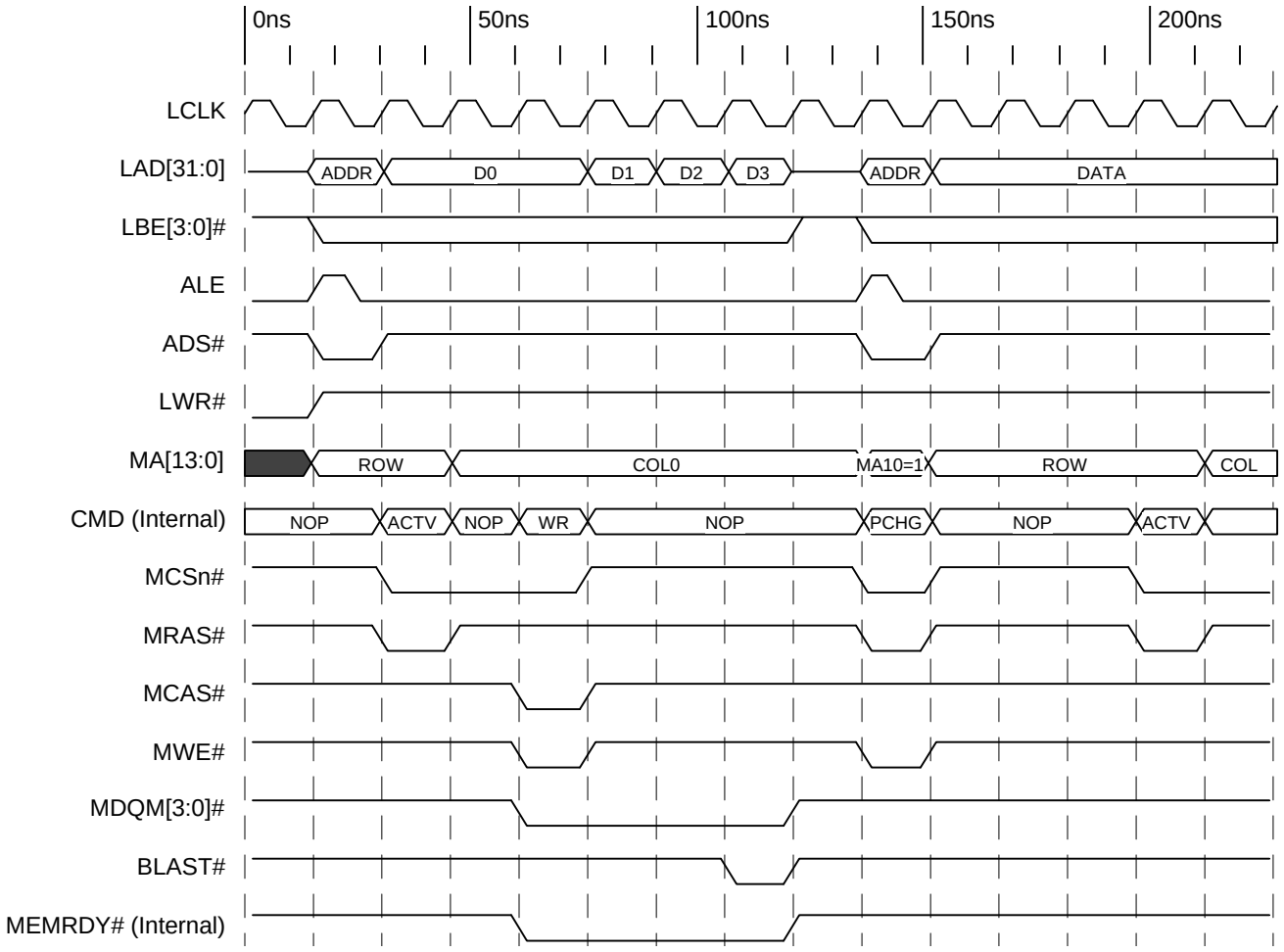
Burst accesses are not permitted to cross the natural page boundary of SDRAM devices.

Three programmable timing parameters are associated with the SDRAM access, as illustrated in Table 12-13.

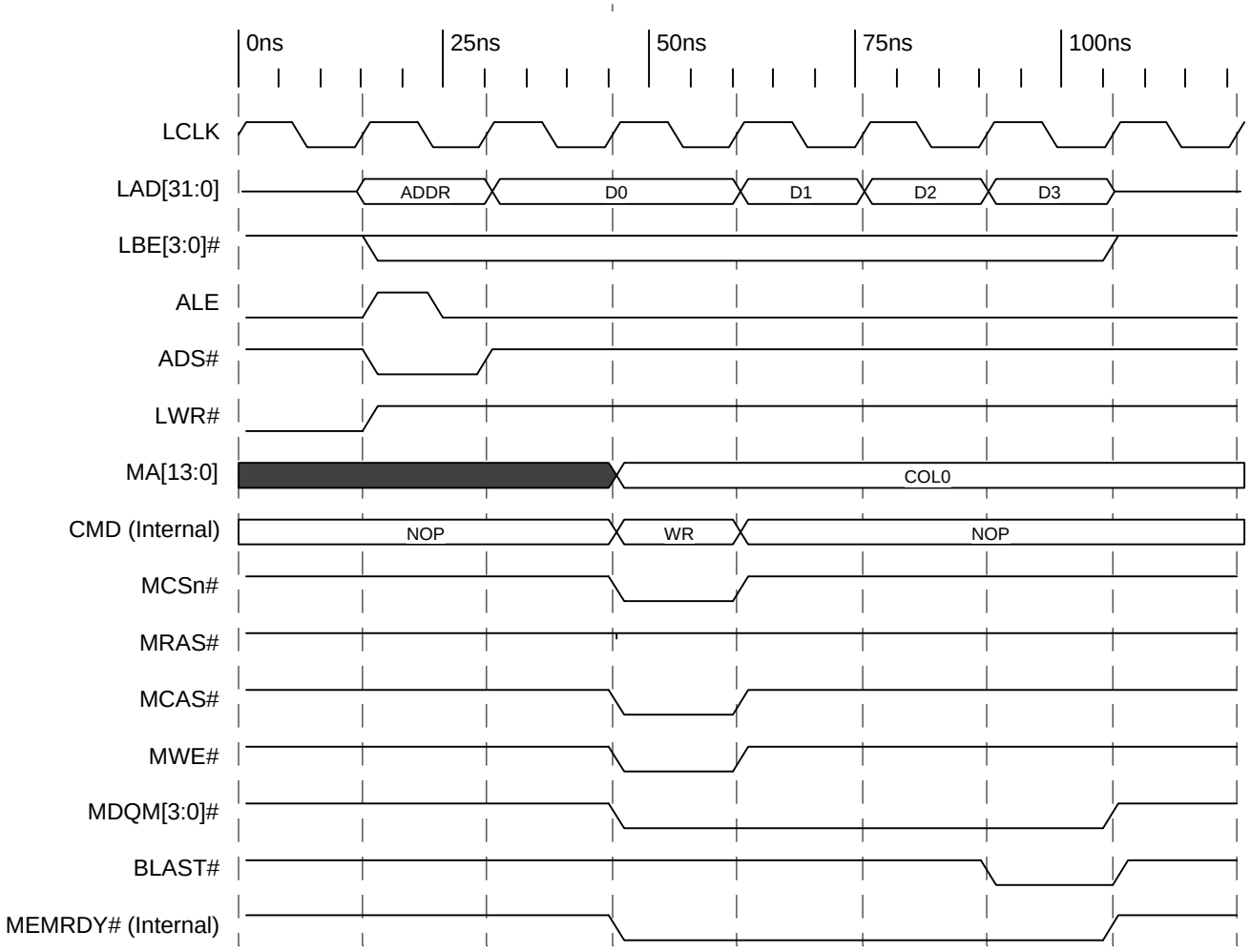
The timing parameter values shown at the top of each timing diagram refer to delay clock periods. This is not necessarily the value written to the corresponding field in the DRAMTIM register. The DRAMTIM register description details the mapping between the value written and resulting number of delay clock periods.

Table 12-13. SDRAM Write Access Programmable Timing Parameters

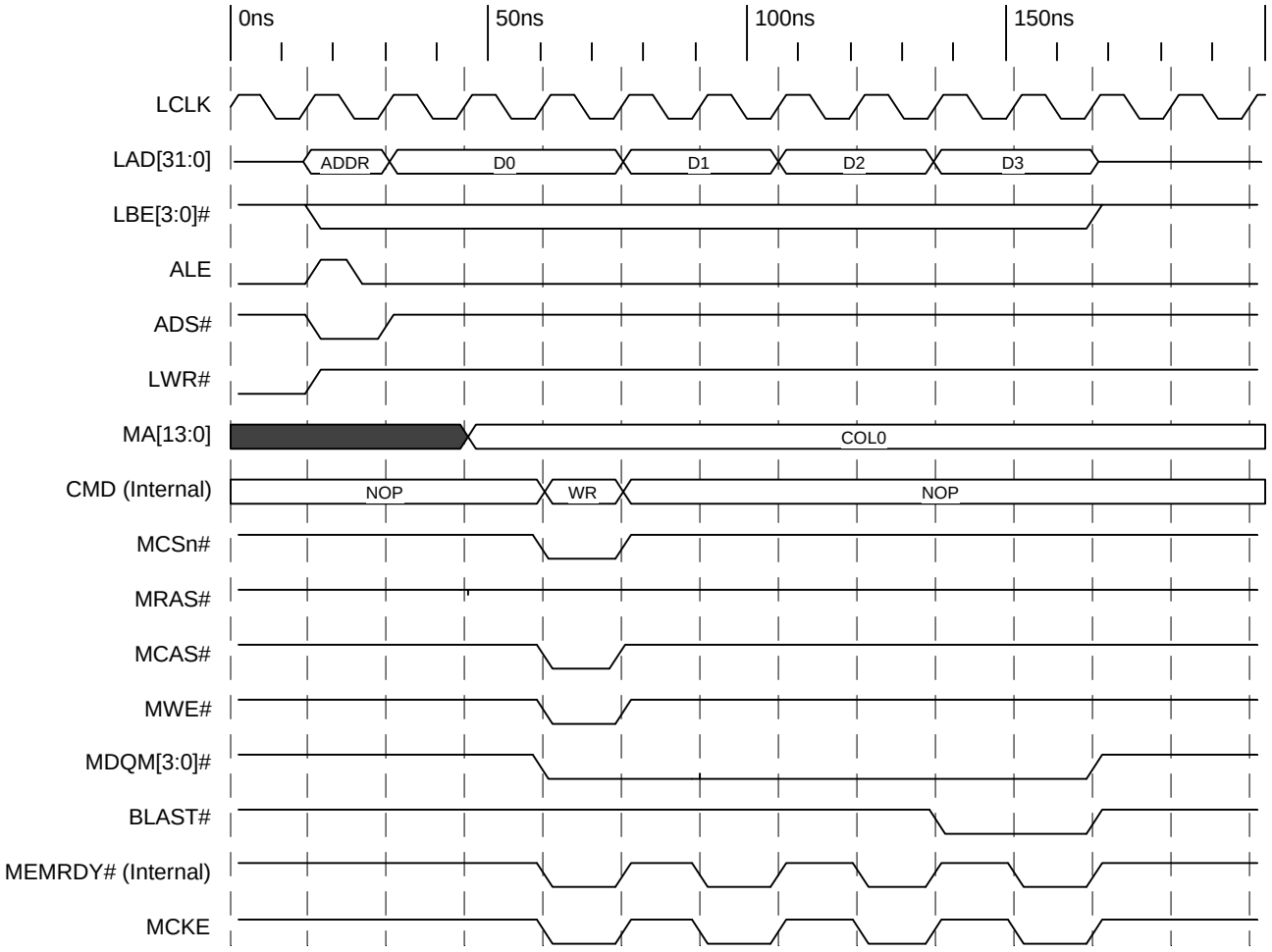
Field	Description
A2C	ACTIVE command to READ/WRITE command delay (1-4). Determines number of clock delays between assertion of ACTIVE and READ/WRITE commands. For a 66 MHz bus, this number is typically set to 1.
W2W	Delay between words of a burst write (0-3). Determines number of clock delays between successive words of a burst write. MCKE is de-asserted during the wait states. For a 66 MHz bus, this number is typically set to 0.
PRCG	Precharge delay (1-4). Determines the number of clock delays required between PRECHARGE and next ACTIVE command. For a 66 MHz bus, this number is typically set to 2.



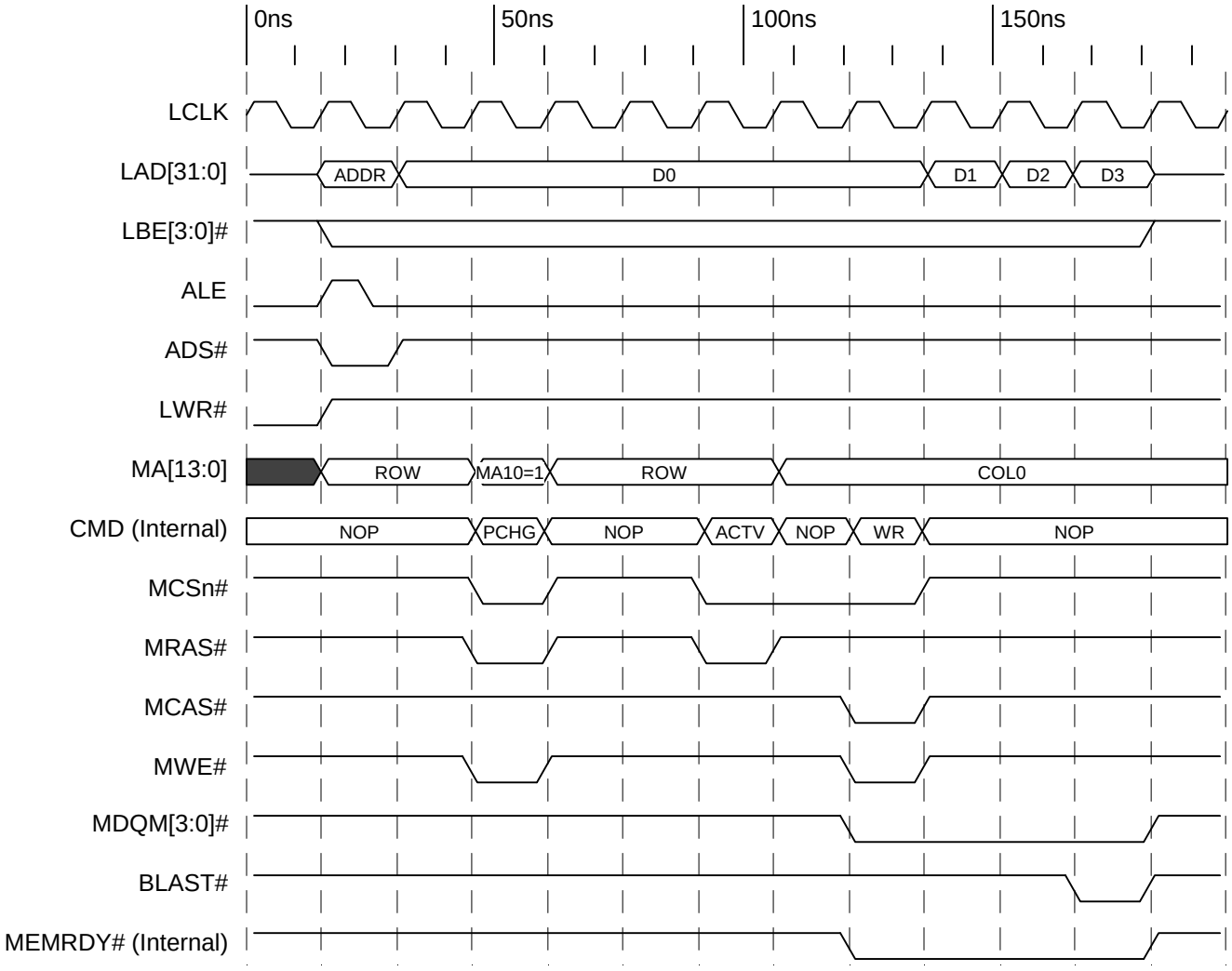
Timing Diagram 12-14. SDRAM Four Word Non-Page Mode Burst Write, 2-0-0-0 Wait States; A2C=1, W2W=0, PRCG=3; Master=IOP 480



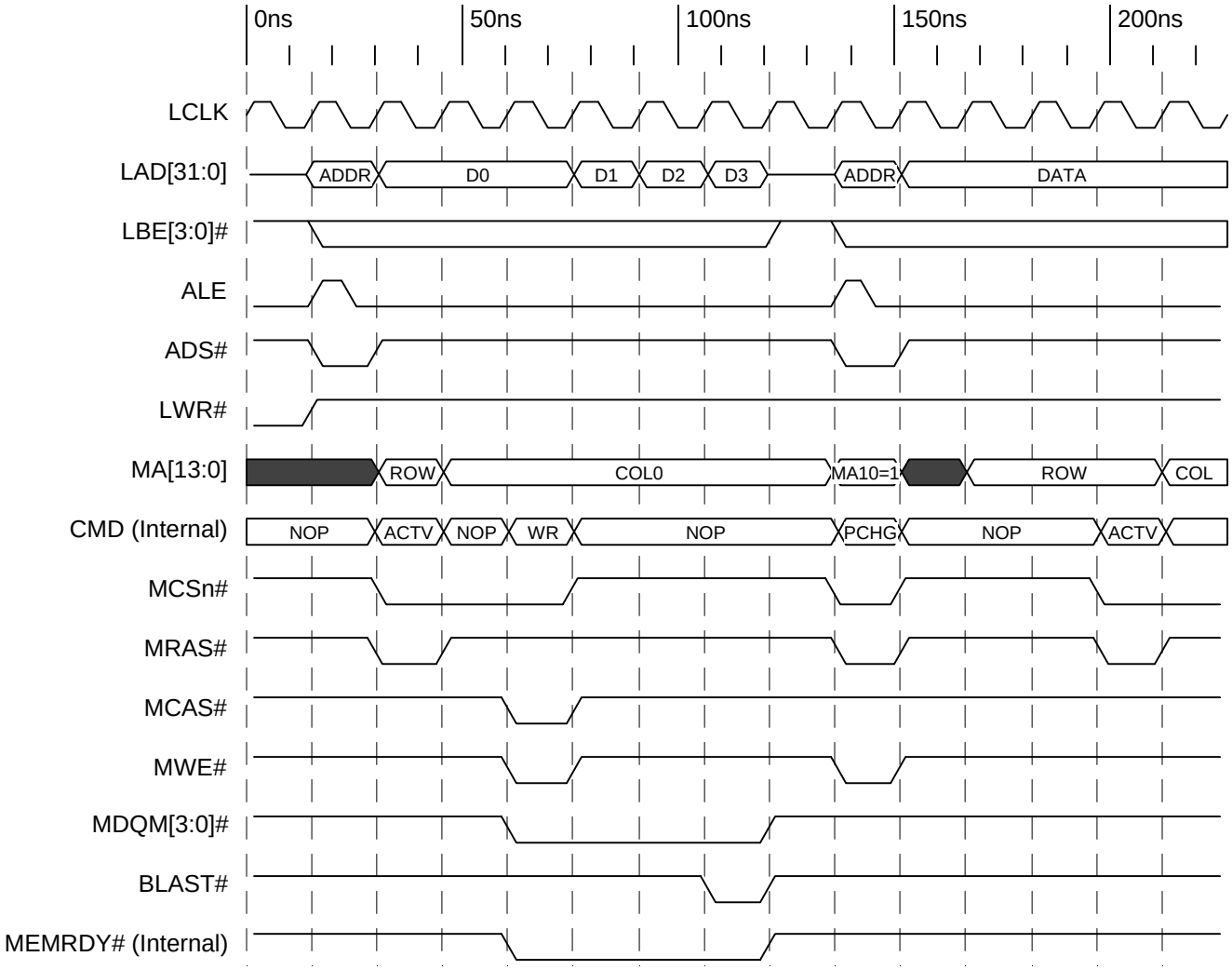
Timing Diagram 12-15. SDRAM Page Hit Burst Write, 1-0-0-0 Wait States; W2W=0; Master=IOP 480



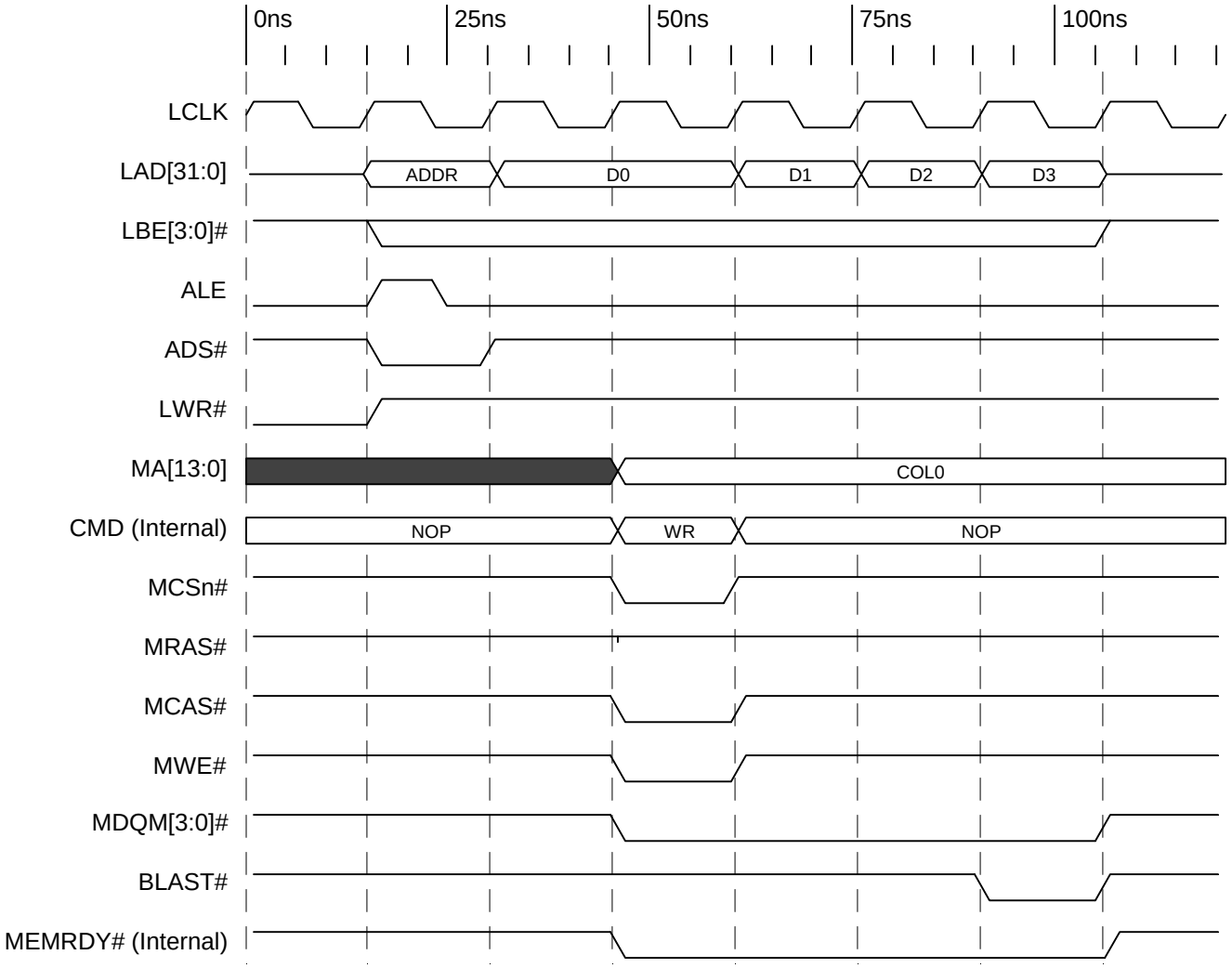
Timing Diagram 12-16. SDRAM Page Hit Burst Write, 2-1-1-1 Wait States; W2W=1; Master=IOP 480



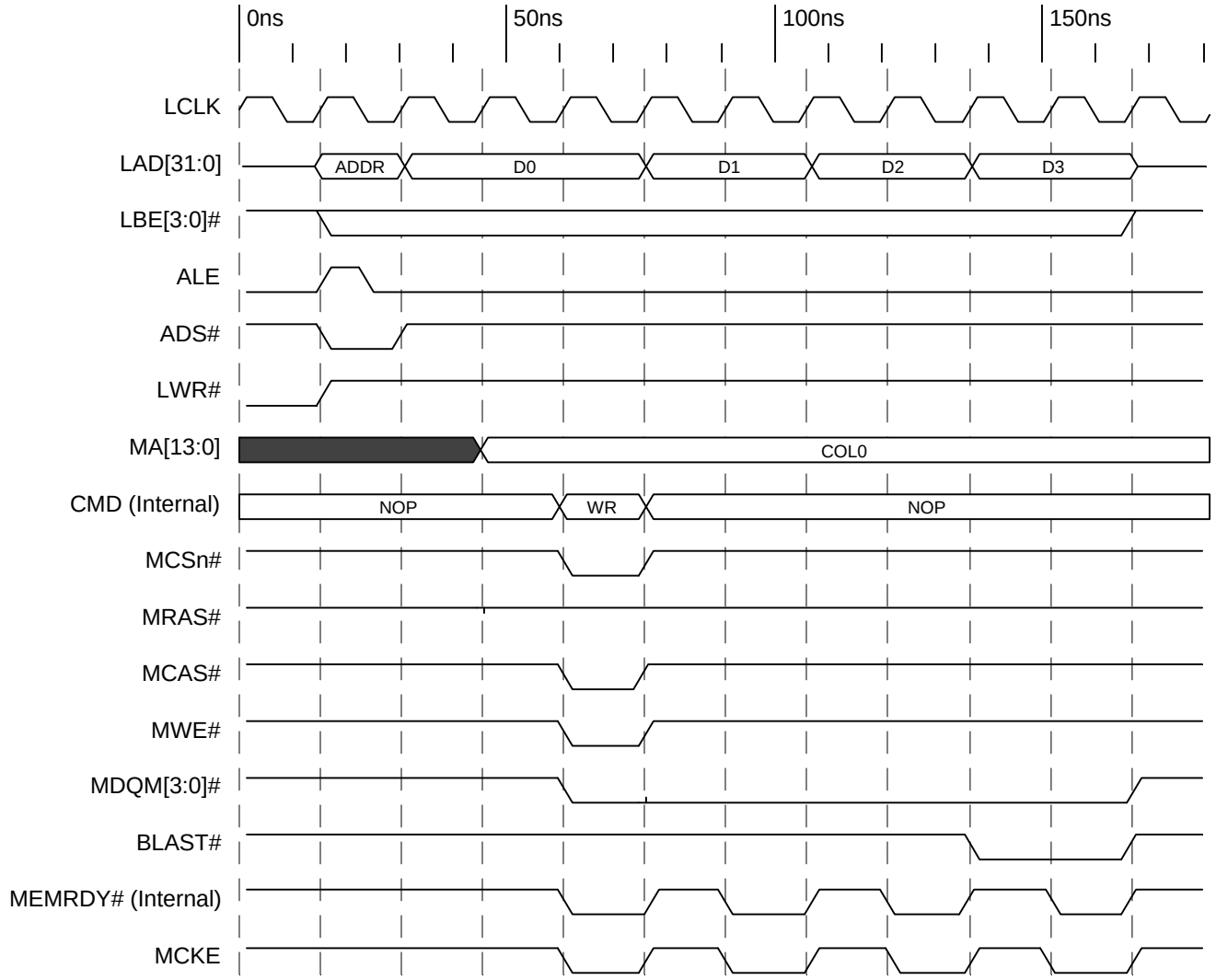
Timing Diagram 12-17. SDRAM Page Miss Burst Write, 6-0-0-0 Wait States; A2C=1, W2W=0, PRCG=2; Master=IOP 480



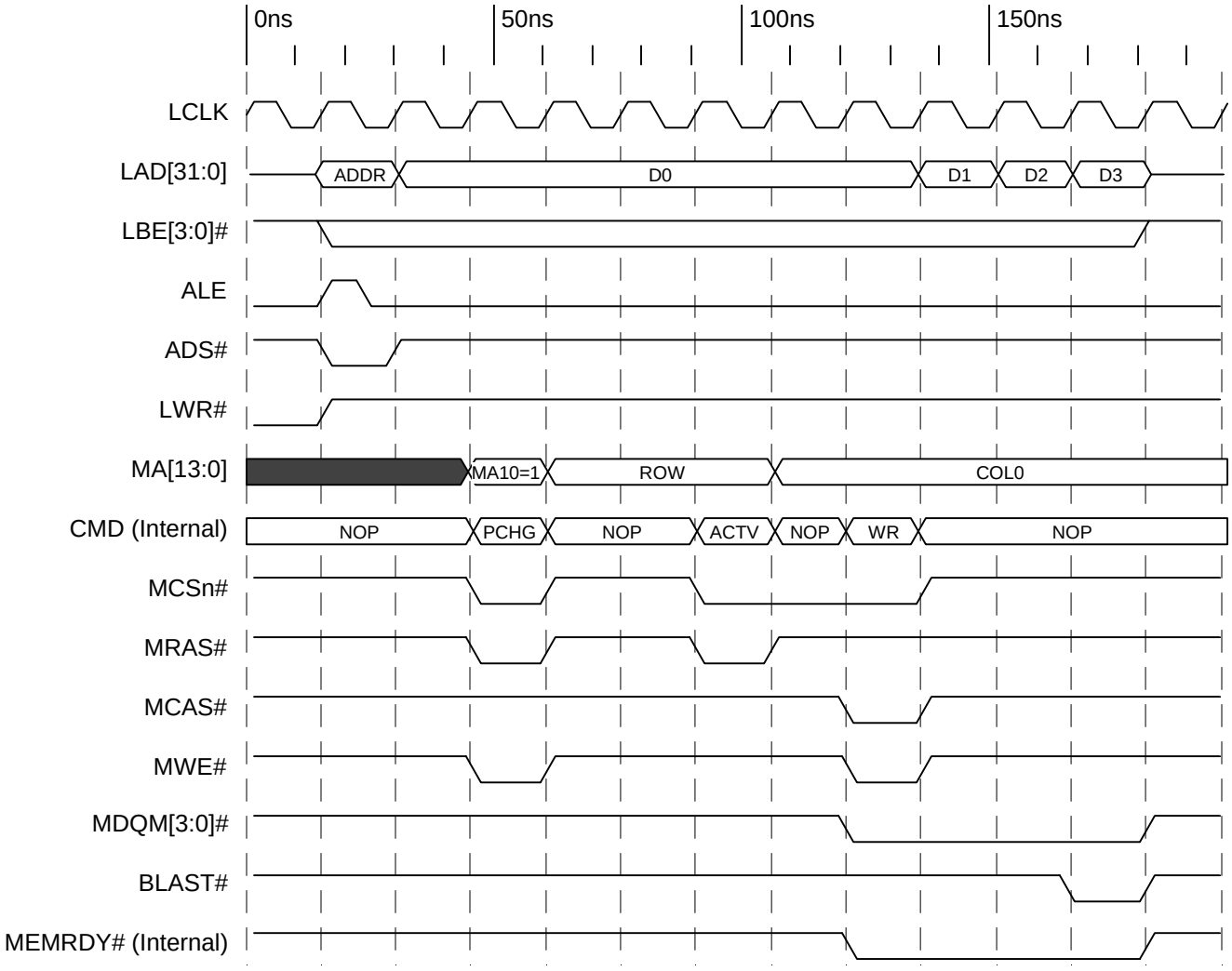
Timing Diagram 12-18. SDRAM Four Word Non-Page Mode Burst Write, 2-0-0-0 Wait States; A2C=1, W2W=0, PRCG=3; Master=External Local Bus Master



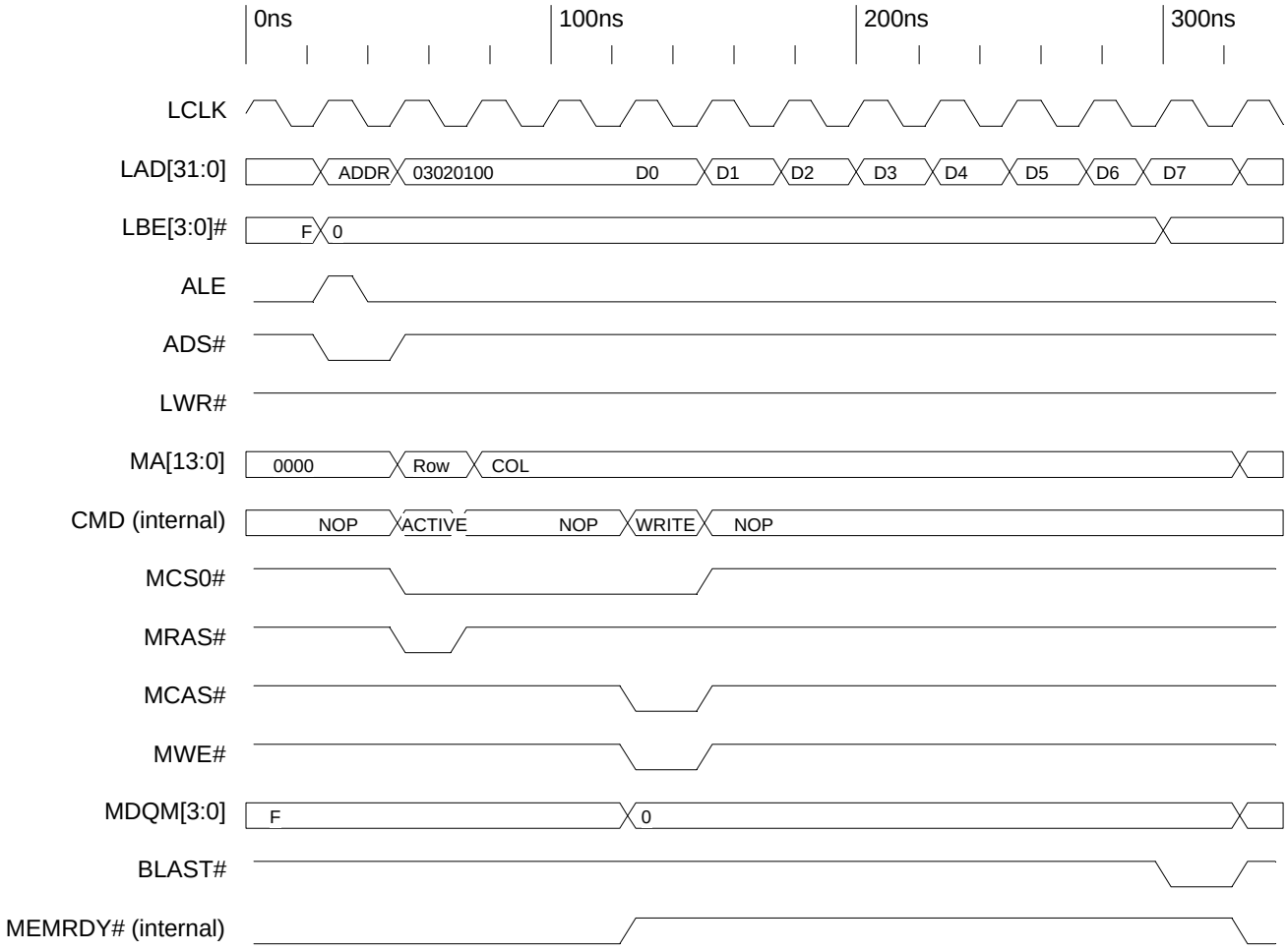
Timing Diagram 12-19. SDRAM Page Hit Burst Write, 1-0-0-0 Wait States; W2W=0; Master=External Local Bus Master



Timing Diagram 12-20. SDRAM Page Hit Burst Write, 2-1-1-1 Wait States; W2W=1; Master=External Local Bus Master



Timing Diagram 12-21. SDRAM Page Miss Burst Write, 6-0-0-0 Wait States; A2C=1, W2W=0, PRCG=2; Master=External Local Bus Master



Timing Diagram 12-22. Long SDRAM Burst Write

12.3.1.6.8 SDRAM Read Access

An SDRAM read access starts when ADS# is asserted, LWR# is low, and the address falls into the appropriate range. If the IOP 480 is controlling the access, the row address is loaded into the MA[13:0] counter at the beginning of the period in which ADS# is asserted. If an external Local Bus Master is controlling the access, the row address is loaded into the MA[13:0] counter at the end of the period in which ADS# is asserted.

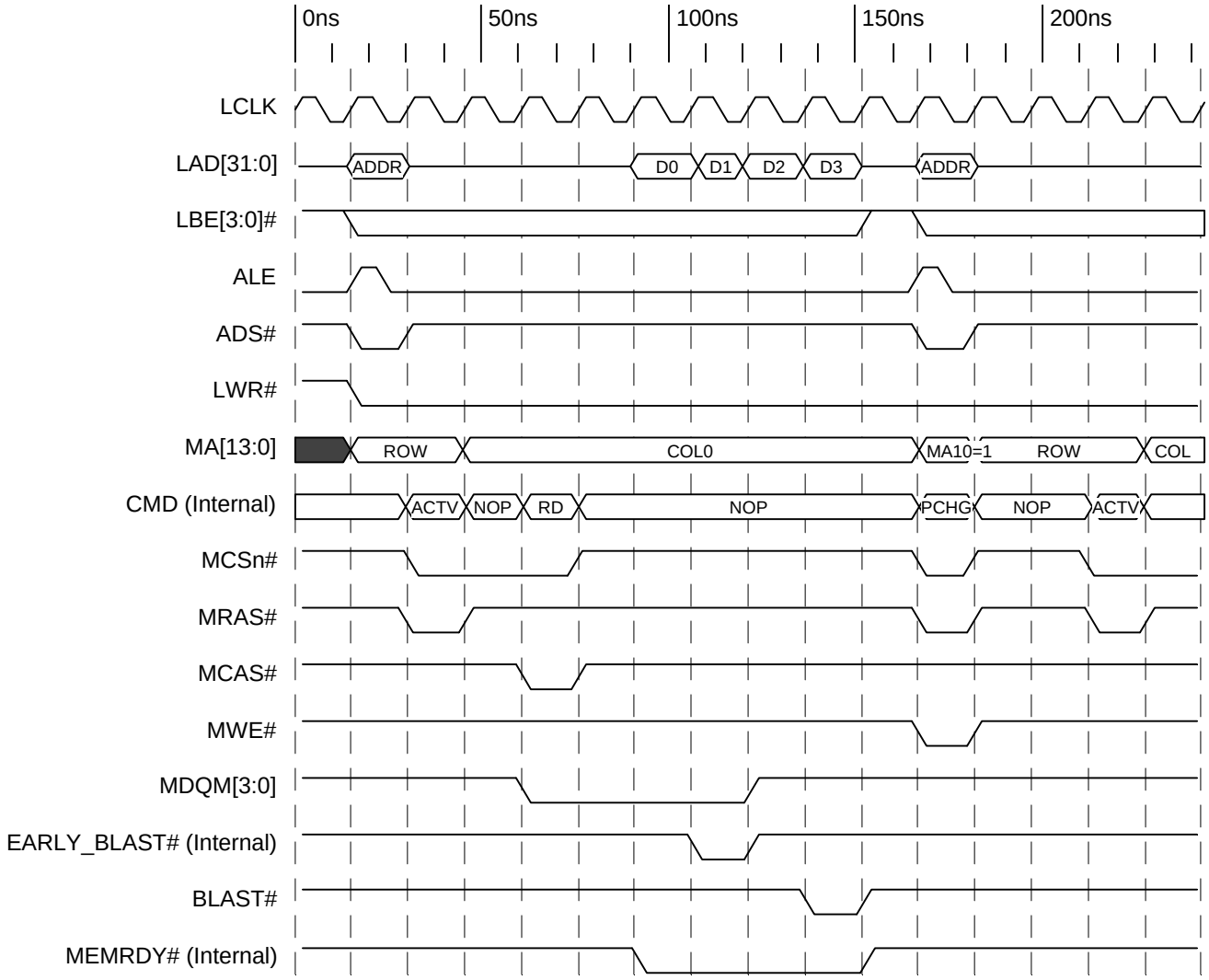
During the next clock period, a row is activated by the assertion of one of the MCS[3:0]# signals and MRAS#. The ACTIVE to READ delay is determined by the A2C value in the DRAM Timing Configuration register. The MA[13:0] bus is switched to the column address as soon as the ACTIVE command is registered. The READ command is initiated by asserting MCAS# and the appropriate MDQM[3:0]# bits. Each MDQM[3:0]# signal corresponds to one of the byte lanes (DQM3# → LAD[31:24],…, DQM0# → LAD[7:0]), and for reads, all MDQM bits are asserted. The first word of data is available during the second clock after the READ command is registered, resulting from CAS latency being set to 2. As long as the MDQM bits remain asserted, data is read on each successive access. When BLAST# is detected, the Burst read is terminated by issuing a PRECHARGE command. MDQM bits are also de-asserted. Because the MDQM bits have two-clock latency, two additional words are read during the PRECHARGE command. For Masters capable of providing advance warning of the end of the burst (internal IOP 480 CPU and PCI interface), the Precharge command can be issued during second to last transfer, thereby eliminating the two extra Read accesses. Burst accesses are not permitted to cross the natural page boundary of SDRAM devices.

There are three programmable timing parameters associated with an SDRAM access (refer to Table 12-14).

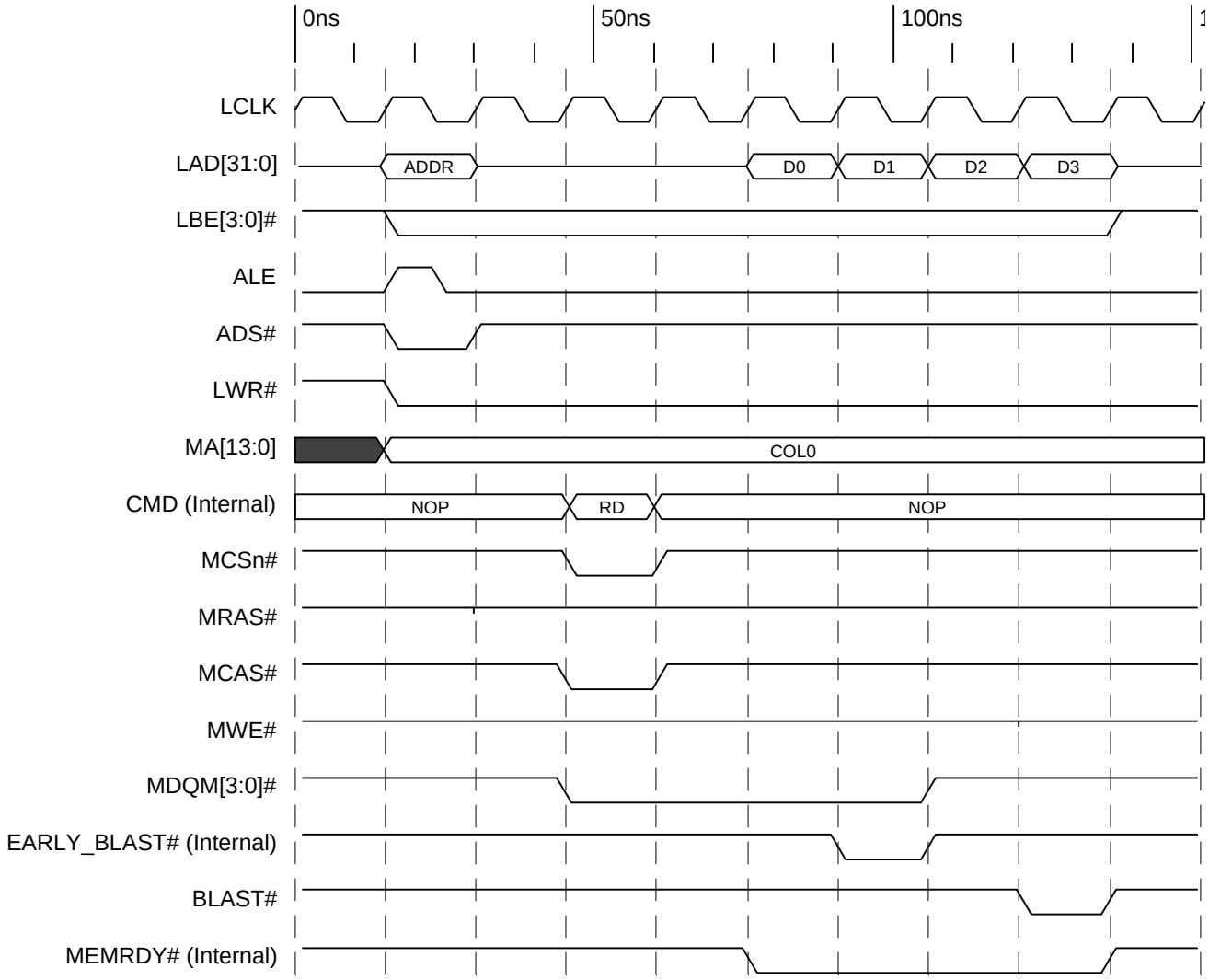
The timing parameter values shown at the top of each timing diagram refer to delay clock periods. This is not necessarily the value written to the corresponding field in the DRAMTIM register. The DRAMTIM register description details the mapping between the value written and resulting number of delay clock periods.

Table 12-14. SDRAM Read Access Programmable Timing Parameters

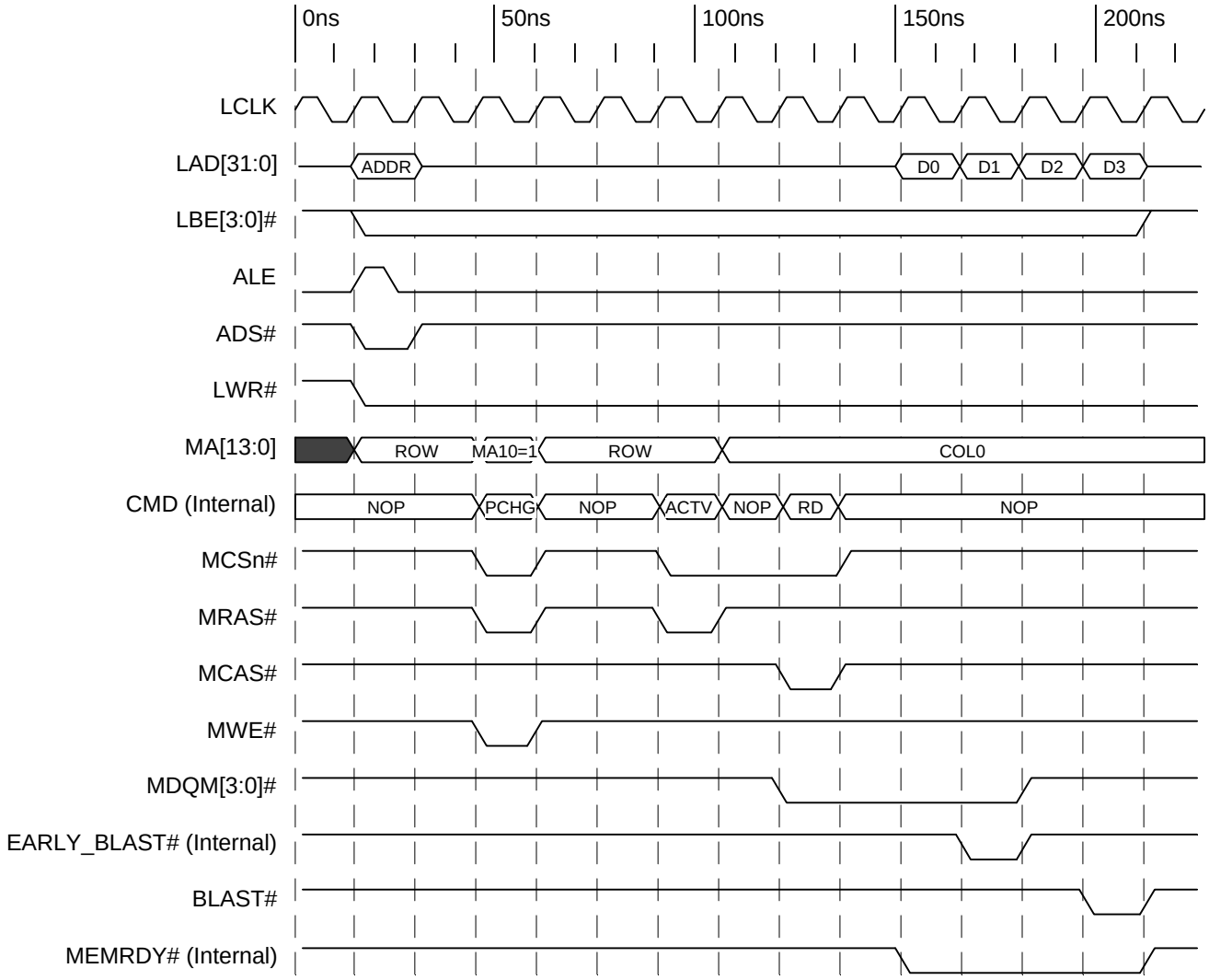
Field	Description
A2C	ACTIVE command to READ/WRITE command delay (1-4). Determines number of clock delays between assertion of ACTIVE and READ/WRITE commands. For a 66 MHz bus, this number is typically set to 1.
PRCG	Precharge delay (1-4). Determines the number of clock delays required between PRECHARGE command and next ACTIVE command. For a 66 MHz bus, this number is typically set to 2.
RRCV	Number of read recovery states (1-4). Determines the number of extra wait states inserted at the end of DRAM access, and used to provide the device with adequate time to float its data output buffers. For a 66 MHz bus, this number is typically set to 2 clocks. For an external Local Bus Master, two extra words are always read at end of a burst read. The RRCV value is loaded into a timer, which operates concurrently with the recovery states generated due to the two-word, read overrun. In this case, the minimum number of recovery states is three, and the maximum is four.



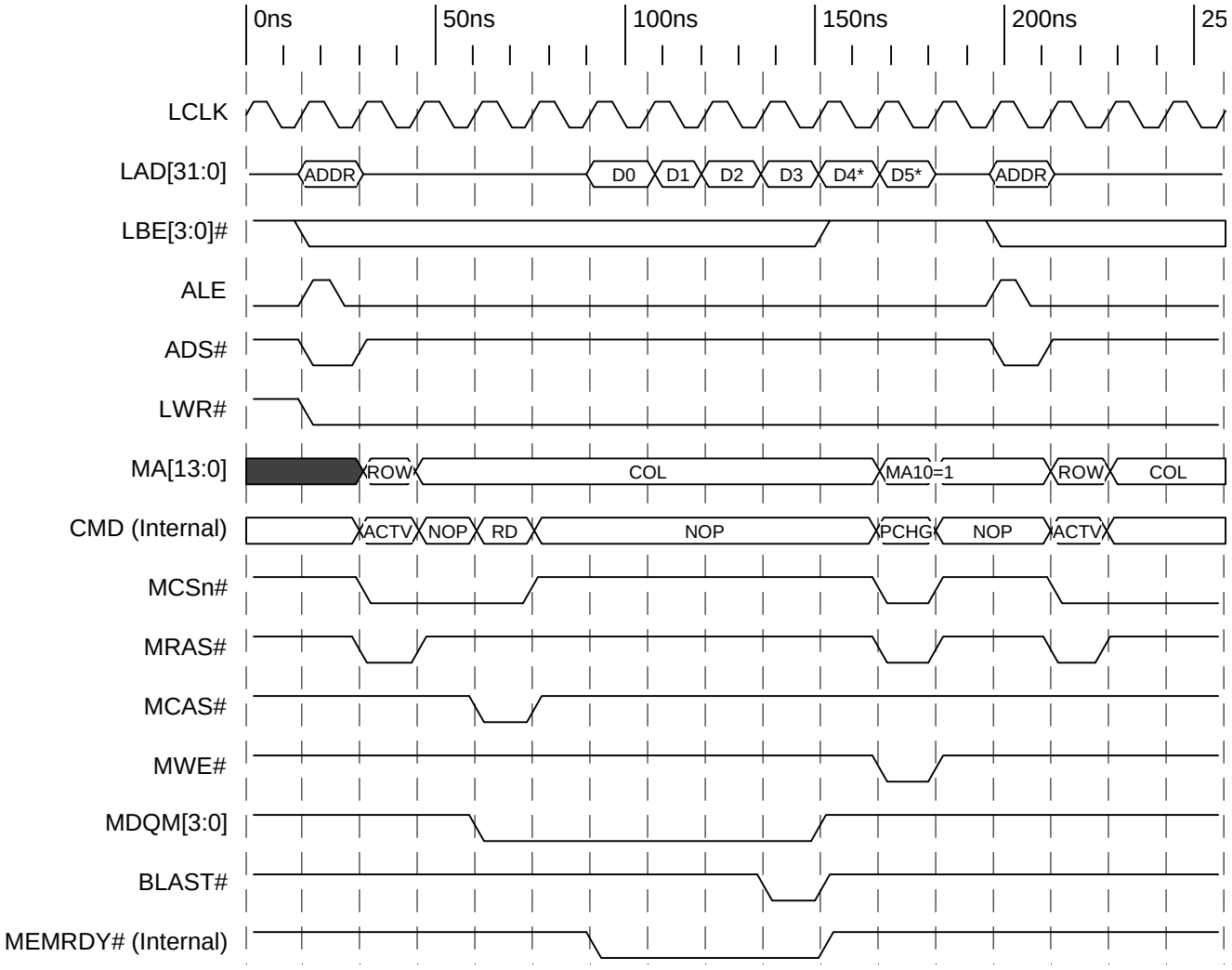
Timing Diagram 12-23. SDRAM Non-Page Mode Burst Read, 4-0-0-0 Wait States; A2C=1, PCHG=2; Master=IOP 480



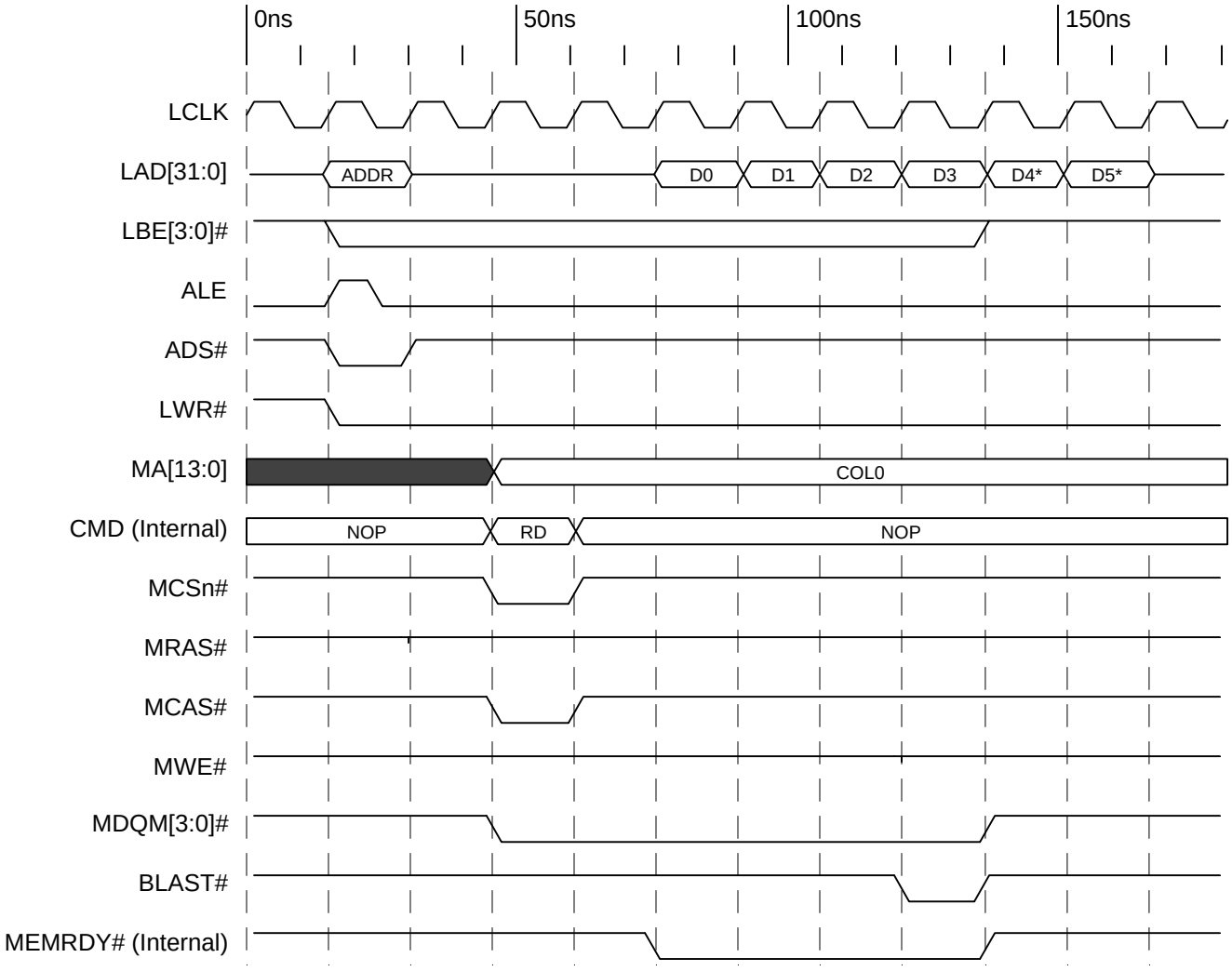
Timing Diagram 12-24. SDRAM Page Hit Burst Read, 3-0-0-0 Wait States; Master=IOP 480



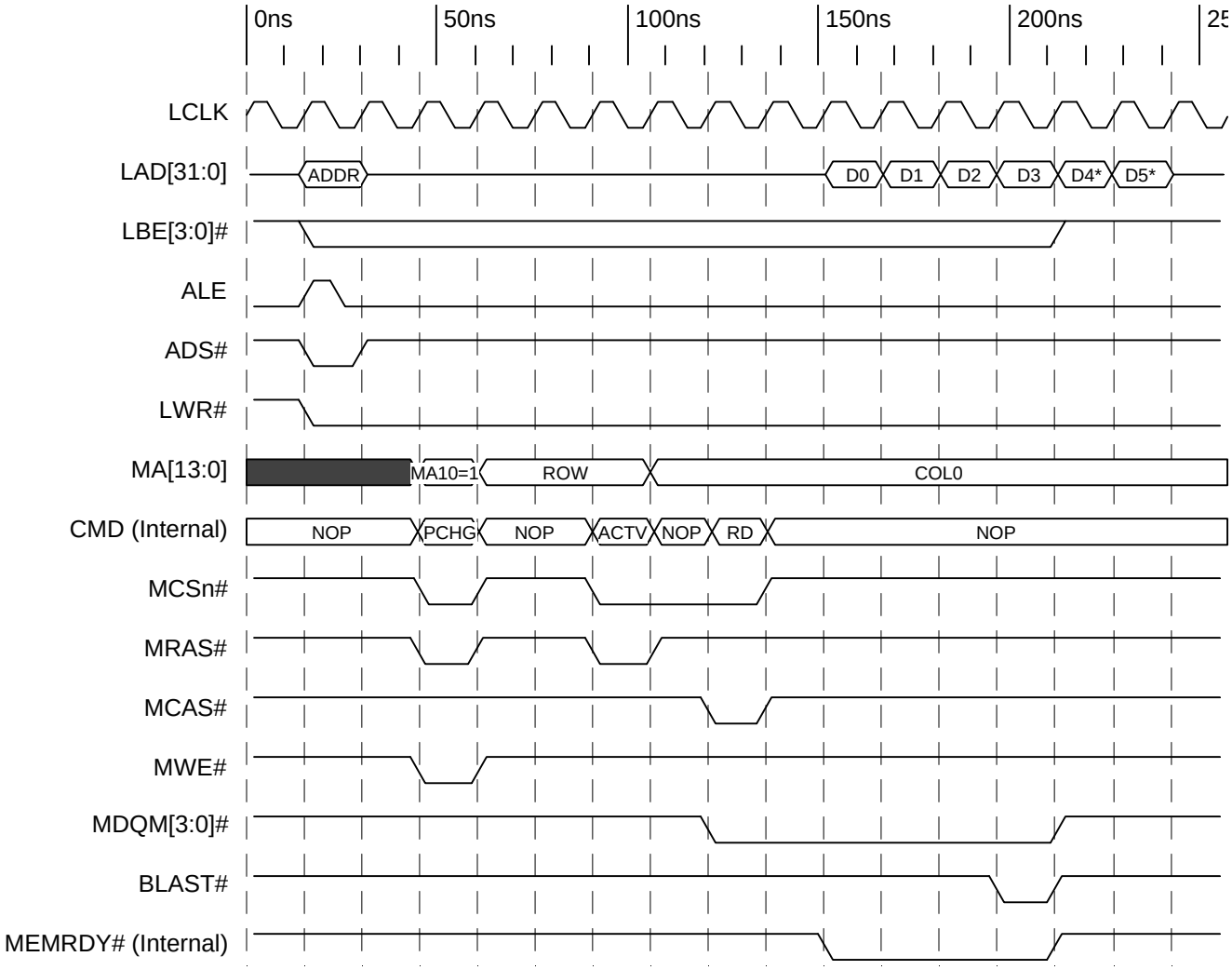
Timing Diagram 12-25. SDRAM Page Miss Burst Read, 8-0-0-0 Wait States; A2C=1, PRCG=2; Master=IOP 480



Timing Diagram 12-26. SDRAM Non-Page Mode Burst Read, 4-0-0-0 Wait States; A2C=1, PCHG=2; Master=External Local Bus Master



Timing Diagram 12-27. SDRAM Page Hit Burst Read, 3-0-0-0 Wait States; Master=External Local Bus Master



Timing Diagram 12-28. SDRAM Page Miss Burst Read, 8-0-0-0 Wait States; A2C=1, PCHG=2; Master=External Local Bus Master

12.3.2 EDO DRAM

12.3.2.1 EDO Signal Connections

Table 12-15 lists the connections between the IOP 480 Memory signals and signal names found in most EDO DRAM specifications.

Table 12-15. EDO Signals

EDO DRAM Signal	IOP 480 Signal	Description
A[12:0]	MA[12:0]	Multiplexed Address Bus
DQ[31:0]	LAD[31:0]	Data Bus
RAS#	RAS*n*#	Row Address Strobe
CAS#	CAS*n*#	Column Address Strobe
WE#	MWE#	Write Enable
OE#	MOE#	Output Enable

Note: *"n" is equal to pin number 0, 1, 2, or 3.*

12.3.2.2 EDO DRAM Examples

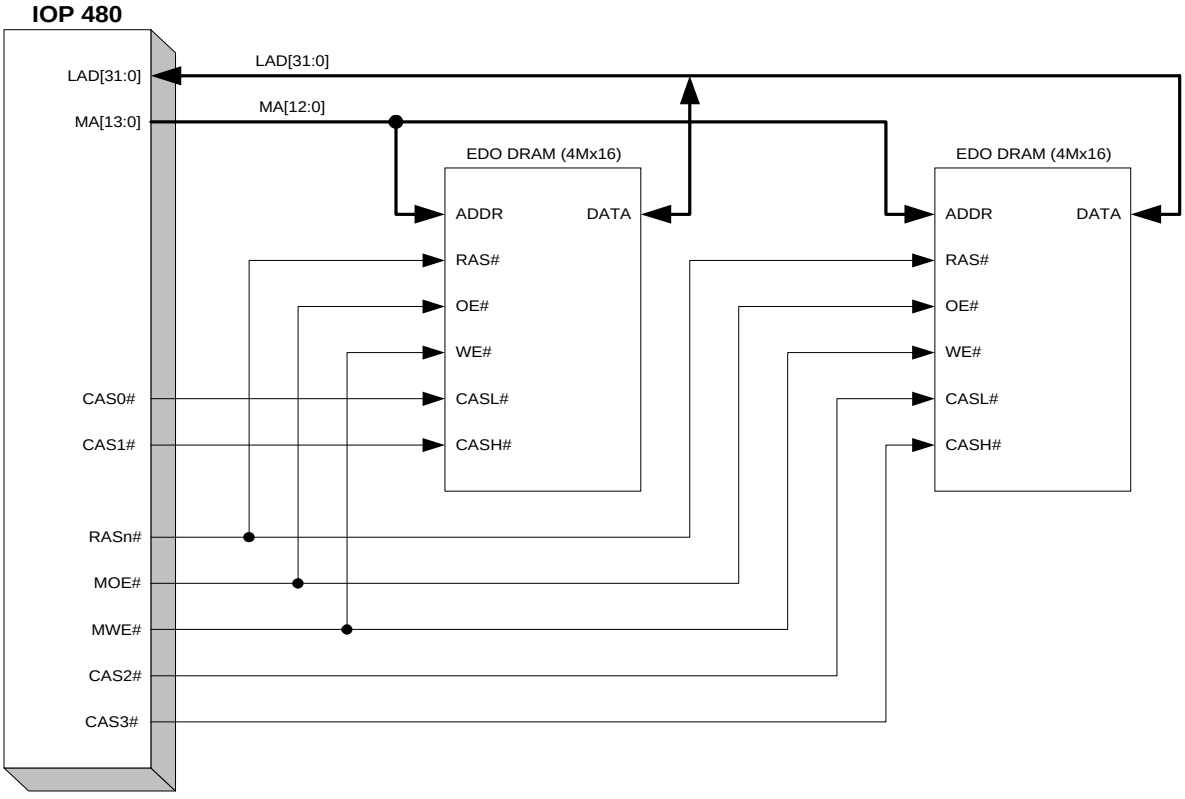


Figure 12-7. EDO DRAM (Two 4M x 16 Devices)

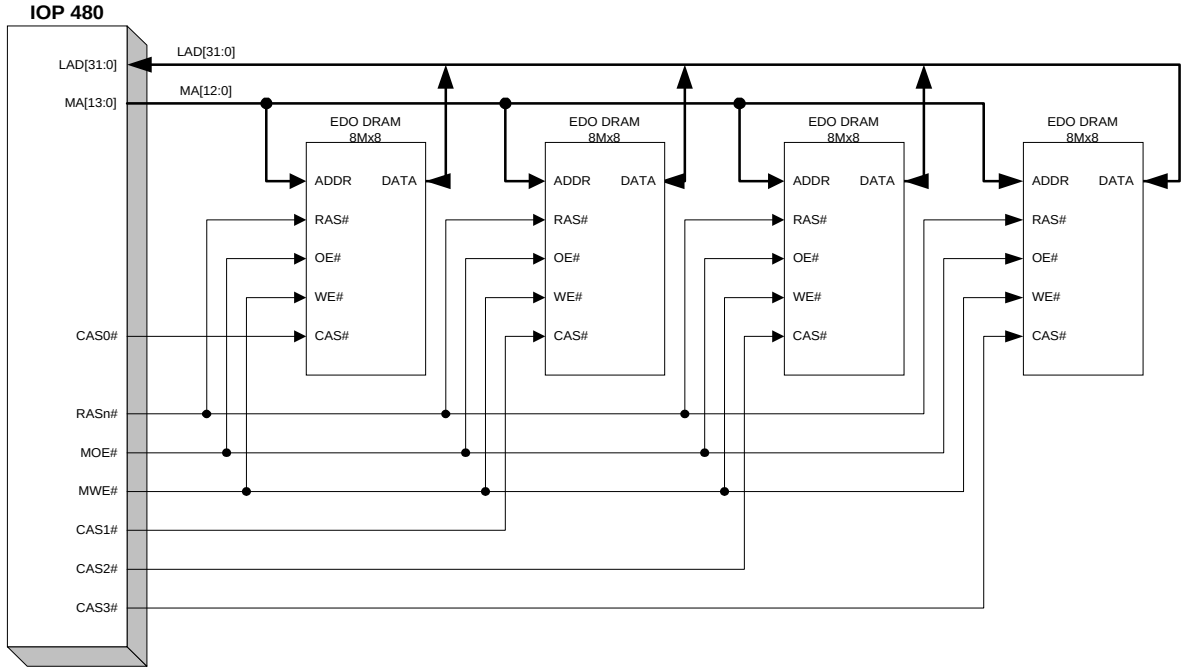


Figure 12-8. EDO DRAM (Four 8M x 8 Devices)

12.3.2.3 EDO Addressing

The multiplexed Memory Address Bus, MA[12:0], is connected to the following Local Address Bus signals for the row and column. The most significant bits of the multiplexed Memory Address Bus, MA[17:13], are not used for EDO DRAMs. DRAM manufacturers offer several configurations of the numbers of rows and columns in a particular device; therefore, the correct selection must be made in the configuration registers. Address bits marked with an asterisk are unused in DRAM. Table 12-17 and Table 12-18 illustrate 16- and 64-megabit EDO Addressing.

The maximum burst length is determined by column size. Bursts cannot cross page boundaries.

Table 12-16. EDO Page Lengths/Boundaries

Number of Columns	Max Page Length	Page Boundary
8	1024	1024
9	2048	2048
10	4096	4096
11	8192	8192
12	16384	16384

Table 12-17. 16-Megabit EDO Addressing

MA Bus	1M x 16				2M x 8				4M x 4			
	8 Column		10 Column		9 Column		10 Column		10 Column		11 Column	
	Row(12)	Col(8)	Row(10)	Col(10)	Row(12)	Col(9)	Row(11)	Col(10)	Row(12)	Col(10)	Row(11)	Col(11)
0	10	2	12	2	11	2	12	2	12	2	13	2
1	11	3	13	3	12	3	13	3	13	3	14	3
2	12	4	14	4	13	4	14	4	14	4	15	4
3	13	5	15	5	14	5	15	5	15	5	16	5
4	14	6	16	6	15	6	16	6	16	6	17	6
5	15	7	17	7	16	7	17	7	17	7	18	7
6	16	8	18	8	17	8	18	8	18	8	19	8
7	17	9	19	9	18	9	19	9	19	9	20	9
8	18	0	20	10	19	10	20	10	20	10	21	10
9	19	0	21	11	20	0	21	11	21	11	22	11
10	20	0	22*	0	21	0	22	0	22	0	23	12
11	21	0	23*	0	22	0	23*	0	23	0	24*	0
12	22*	0	24*	0	23*	0	24*	0	24*	0	25*	0
13	23*	0	25*	0	24*	0	25*	0	25*	0	26*	0

Table 12-18. 64-Megabit EDO Addressing

MA Bus	4M x 16				8M x 8				16M x 4			
	9 Column		10 Column		10 Column		11 Column		11 Column		12 Column	
	Row(13)	Col(9)	Row(12)	Col(10)	Row(13)	Col(10)	Row(12)	Col(11)	Row(13)	Col(11)	Row(12)	Col(12)
0	11	2	12	2	12	2	13	2	13	2	14	2
1	12	3	13	3	13	3	14	3	14	3	15	3
2	13	4	14	4	14	4	15	4	15	4	16	4
3	14	5	15	5	15	5	16	5	16	5	17	5
4	15	6	16	6	16	6	17	6	17	6	18	6
5	16	7	17	7	17	7	18	7	18	7	19	7
6	17	8	18	8	18	8	19	8	19	8	20	8
7	18	9	19	9	19	9	20	9	20	9	21	9
8	19	10	20	10	20	10	21	10	21	10	22	10
9	20	0	21	11	21	11	22	11	22	11	23	11
10	21	0	22	0	22	0	23	12	23	12	24	12
11	22	0	23	0	23	0	24	0	24	0	25	13
12	23	0	24*	0	24	0	25*	0	25	0	26*	0
13	24*	0	25*	0	25*	0	26*	0	26*	0	27*	0

12.3.2.4 Initialization

After EDO DRAMs are powered up, no DRAM accesses are started for at least 200 µs. This timeout is based on 13200 clock ticks at a Local Bus clock frequency of 66 MHz. If a slower Local Bus clock is used, then the power-on delay increases. The delay is calculated as follows: delay = (1/Local Bus clock freq) x 13200. All control signals (RAS#, CAS#, WE#, and OE#) are held in an inactive state during this period. After the 200 µs delay, eight CAS-before-RAS (CBR) refresh accesses are run. The EDO DRAMs are now ready for Read and Write accesses.

12.3.2.5 Refresh

A CAS-before-RAS (CBR) refresh access is run at a rate determined by the Refresh Interval field of the DRAM Control Register and the Local Bus clock frequency. A typical refresh rate for most DRAMs is 15.625 µs, and is calculated as follows:

register value = refresh rate x LCLK clock frequency.

Typically, the register value = 15.625 µs x 66.666 MHz = 1041 = 0x411. The value after reset is 0x407.

Refresh access has priority over other DRAM requests. A refresh access only occurs when both the EDO and SRAM state machines are idle. (Refer to Figure 12-29.)

If a Refresh request occurs during a DRAM Burst access, then the Master of the Burst access is preempted.

Table 12-19. Refresh Arbitration

Master	Action
Internal IOP 480 CPU	Wait for end of burst (limited to four words)
Direct Slave Controller	Assert BLAST# and terminate burst
DMA Controllers	Assert BLAST# and terminate burst
LHOLDREQ0	Assert BOFF#
LHOLDREQ1	None

12.3.2.6 Page Mode

When RAS# is asserted, any column within the current row can be accessed by asserting CAS# with the appropriate column address. If the Page mode bit in DRAM configuration registers is enabled, RAS# remains asserted after the end of an access. During the next access address cycle, the new row address is compared with previous row address. If they match, a page hit occurs and the column can be addressed without strobing in a new row address. If a page miss occurs, then RAS# must be de-asserted, the RAS precharge time must be satisfied, and a new row address strobed in before data is accessed. If an application tends towards sequential accesses, then enabling Page mode improves performance. With Page mode disabled, random accesses achieve the best performance.

12.3.2.7 EDO Write Access

An EDO Write access is started when ADS# is asserted, LWR# is high, and the address falls into the appropriate range. The MOE# signal is connected to the OE# input of DRAMs, and disables output drivers during a Write access. If the IOP 480 is controlling the access, the row address is loaded into the MA[12:0] counter at the beginning of the period in which ADS# is asserted. If an external Local Bus Master is controlling the access, the row address is loaded into the MA[12:0] counter at the end of the period in which ADS# is asserted. The MWE# signal is also asserted at the same time as the row address. Because MWE# is asserted before the CAS[3:0]# signals, an early Write access is utilized. During an early write, the data output remains in the high-impedance state regardless of the OE# input, and data is latched on the leading (falling) edge of CAS[3:0]#. Each CAS signal corresponds to one of the byte lanes (CAS3# → LAD[31:24],…,CAS0# → LAD[7:0]).

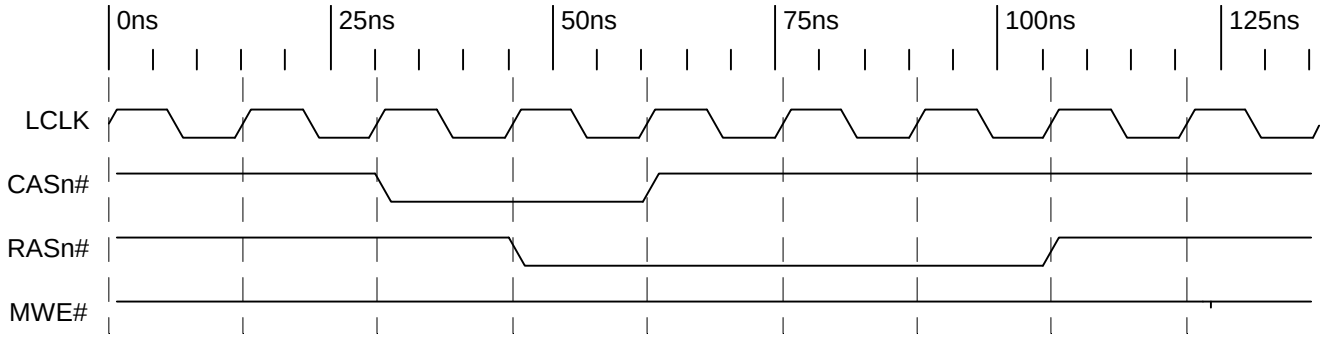
After the Row Address to RAS delay, one of the RAS[3:0]# signals is asserted, strobing the row address into the device. After the RAS to Column Address delay, the MA[12:0] Address Bus is switched to the column address. After the Column Address to CAS delay, CAS[3:0]# is asserted, strobing the column address into the device. The write is complete when CAS[3:0]# is de-asserted. If a burst access is requested, then the MA[12:0] address is incremented and CAS[3:0]# is re-asserted. Burst accesses are not permitted to cross a page boundary (refer to Table 12-16).

There are five types of programmable timing parameters associated with an EDO DRAM write access, as illustrated in Table 12-20.

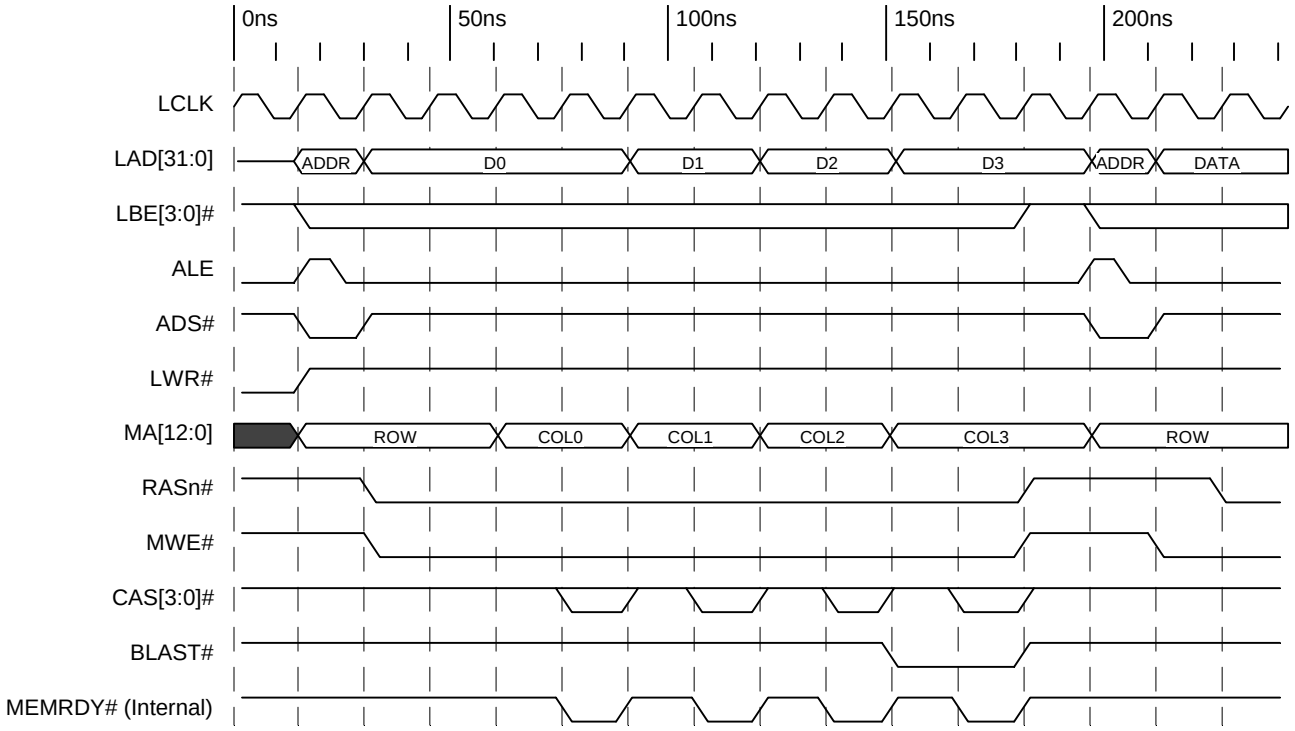
The timing parameter values shown at the top of each timing diagram refer to delay clock periods. This is not necessarily the value written to the corresponding field in the DRAMTIM register. The DRAMTIM register description details the mapping between the value written and resulting number of delay clock periods.

Table 12-20. EDO DRAM Write Access Programmable Timing Parameters

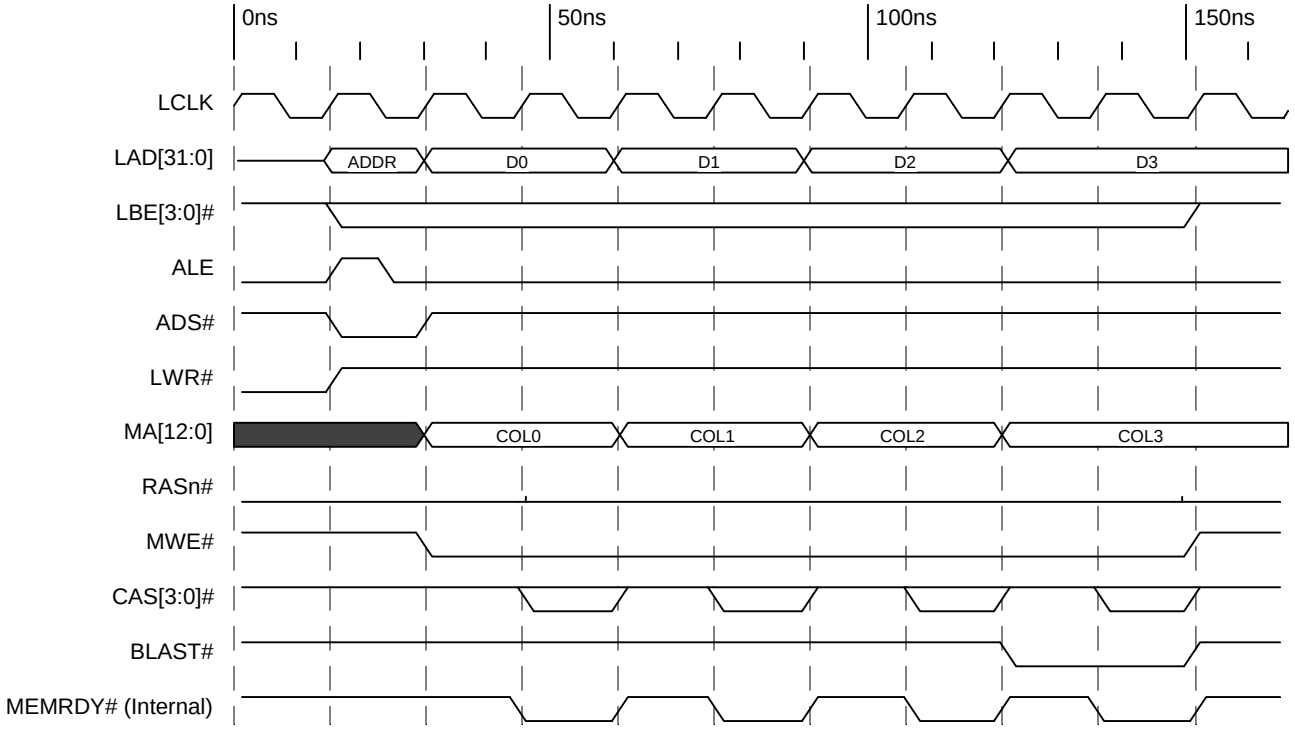
Field	Description
R2R	Row Address to RAS delay (1-4). Determines the number of clock delays between assertion of the row address and RAS[3:0]#. For a 66 MHz bus, this number is typically set to 1 clock for a 60 ns DRAM. For a heavily loaded MA[12:0] bus, or when using external buffers, this value may need to be increased.
R2C	RAS to Column Address delay (1-4). Determines the number of clock delays between assertion of RAS[3:0]# and the first column address. For a 66 MHz bus, this number is typically set to 2 clocks for a 60 ns DRAM. For a heavily loaded MA[12:0] bus, or when using external buffers, this value may need to be increased.
C2C	Column Address to CAS delay (1-4). Determines the number of clock delays between assertion of column address and CAS[3:0]#. For a 66 MHz bus, this number is typically set to 1 clock for a 60 ns DRAM. For a heavily loaded MA[12:0] bus, or when using external buffers, this value may need to be increased. Note that during a page hit when IOP 480 is Bus Master, there is always additional delay from column address to CAS[3:0]# because the address comes out one clock earlier.
WCW	Write CAS Width (1-4). Determines the width of each CAS[3:0]# pulse. For a 66 MHz bus, this number is typically set to 1 clock. The internal MEMRDY# signal is asserted in the last clock period during which CAS[3:0]# is asserted.
PRCG	Number of RAS precharge states (1-4). Determines the number of states between de-assertion of one RAS[3:0]# and assertion of the next RAS[3:0]#, and is used to provide adequate RAS precharge time. For a 66 MHz bus, this number is typically set to 3 clocks.



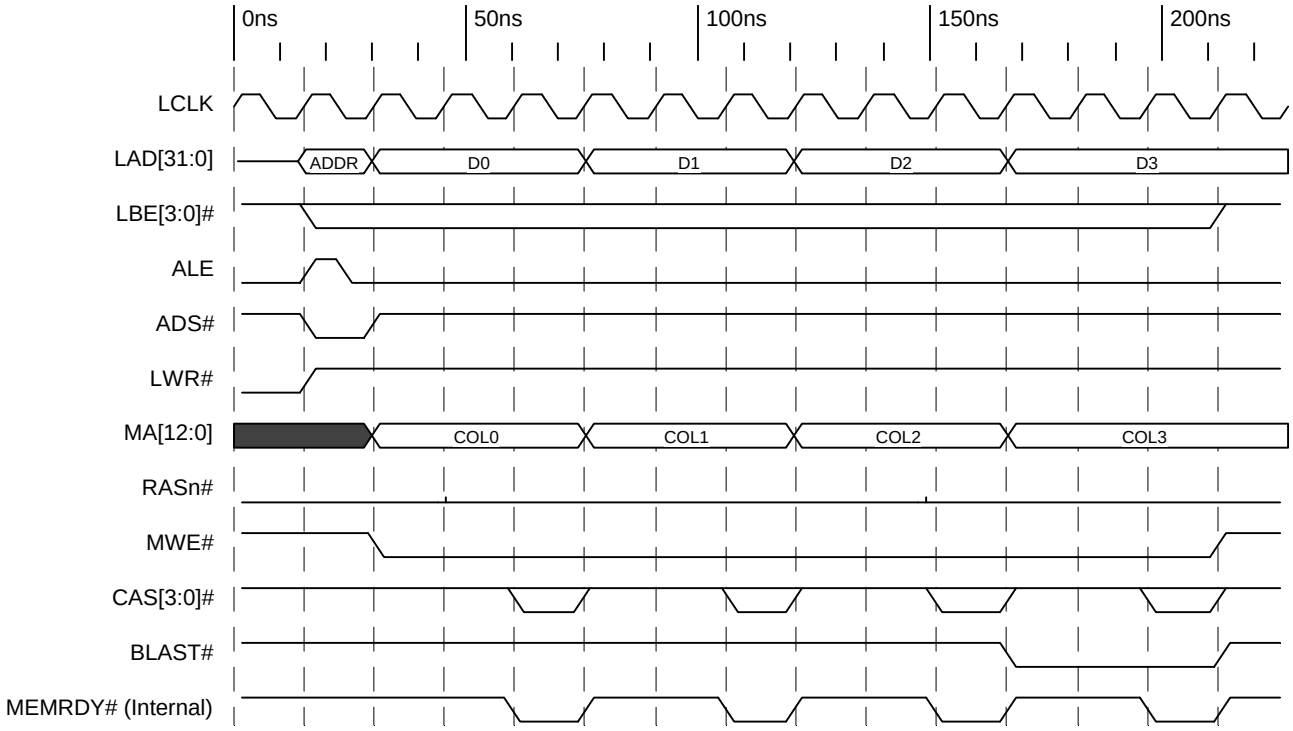
Timing Diagram 12-29. EDO DRAM Refresh



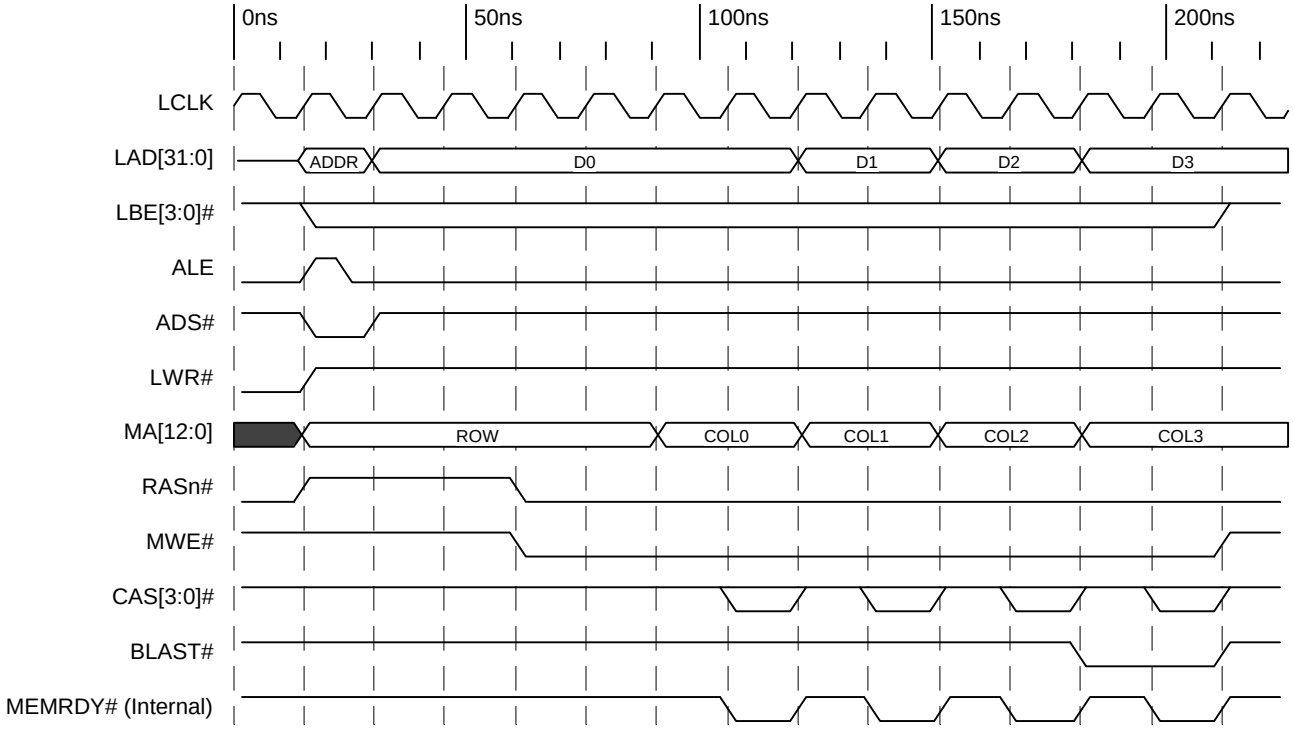
Timing Diagram 12-30. EDO DRAM Non-Page Mode Burst Write, 3-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, WCW=1, PRCG=3; Master=IOP 480



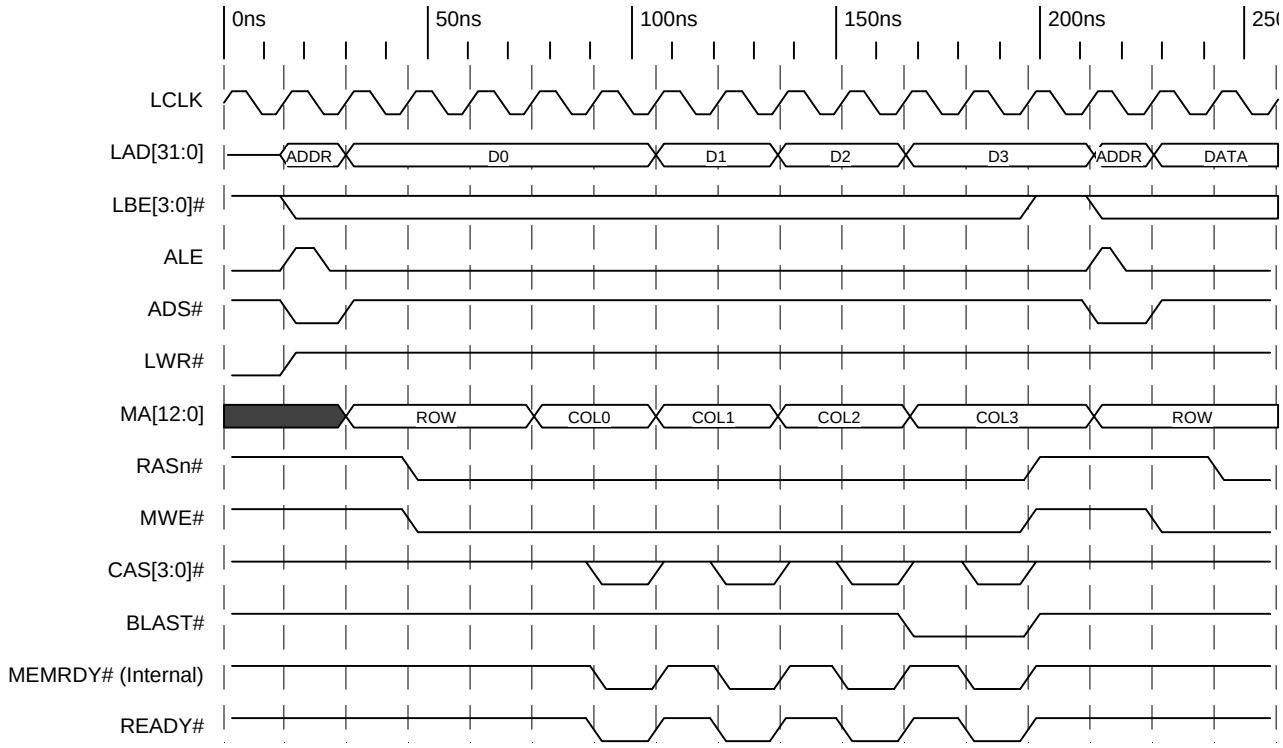
Timing Diagram 12-31. EDO DRAM Page Hit Burst Write, 1-1-1-1 Wait States; C2C=1, WCW=1; Master=IOP 480



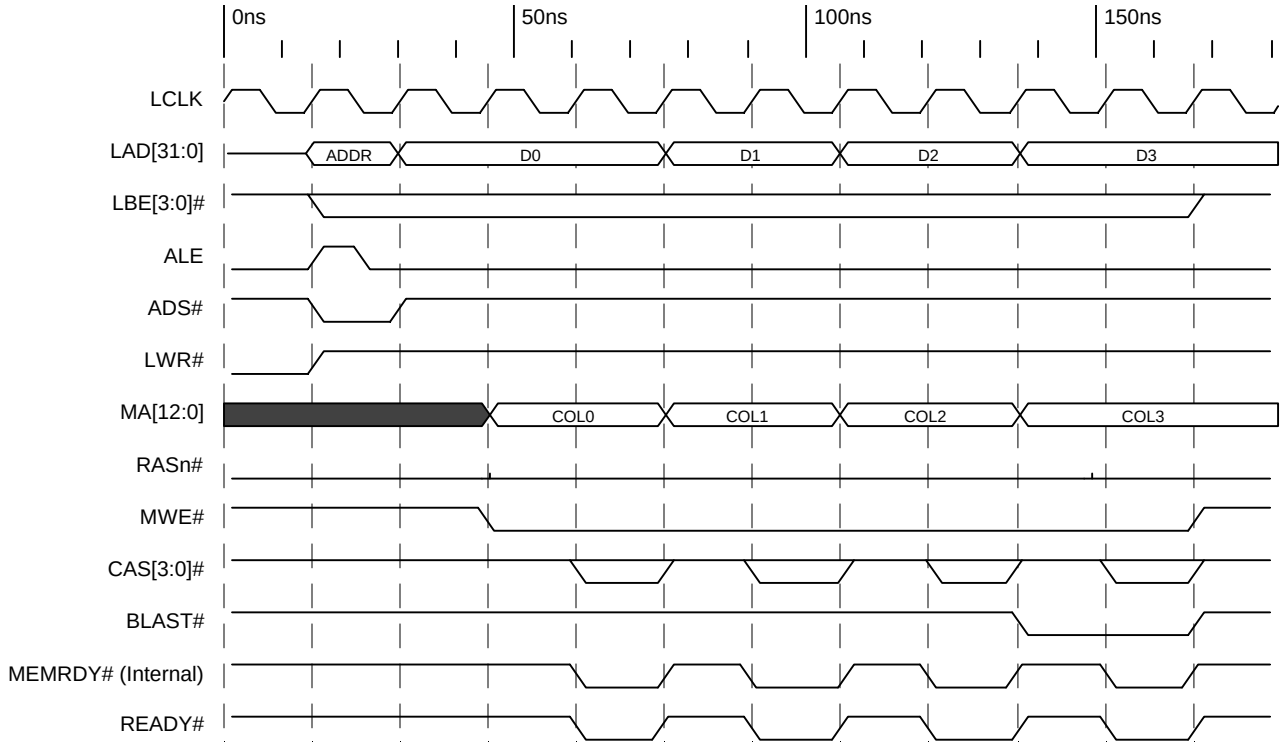
Timing Diagram 12-32. EDO DRAM Page Hit Burst Write, 2-2-2-2 Wait States; C2C=2, WCW=1; Master=IOP 480



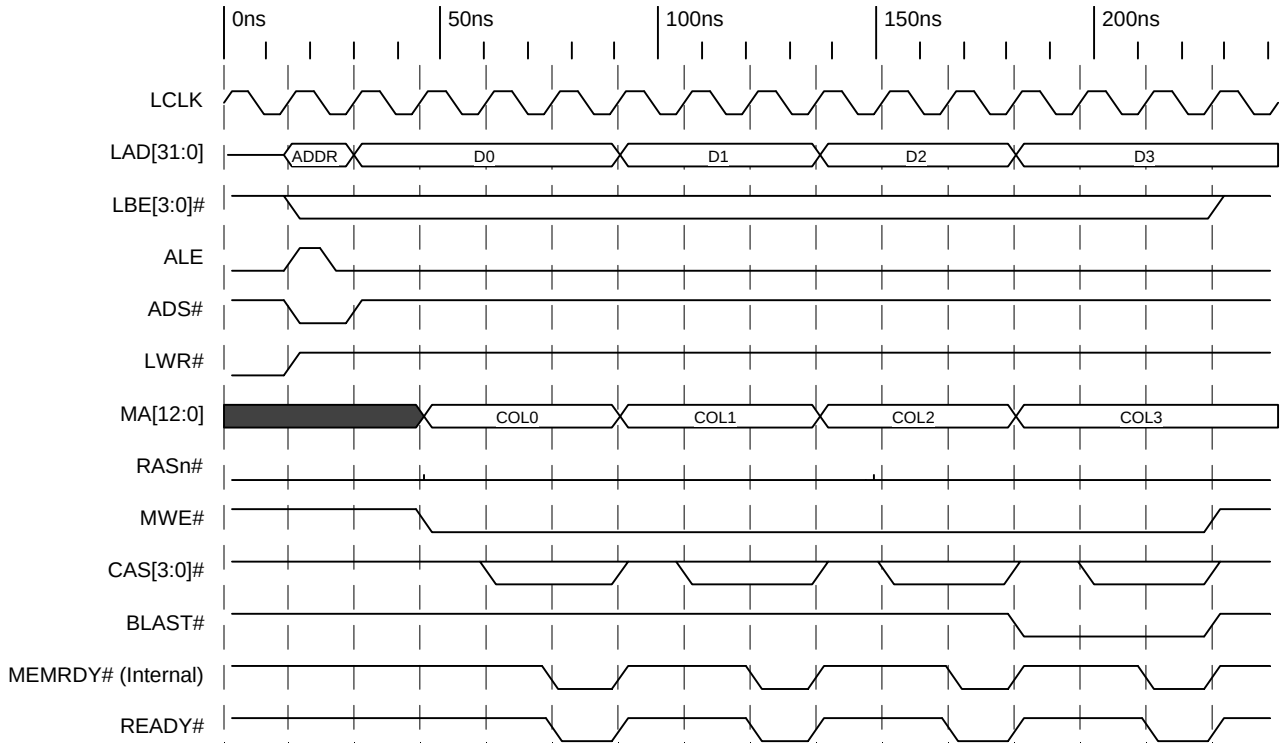
Timing Diagram 12-33. EDO DRAM Page Miss Burst Write, 6-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, WCW=1, PRCG=3; Master=IOP 480



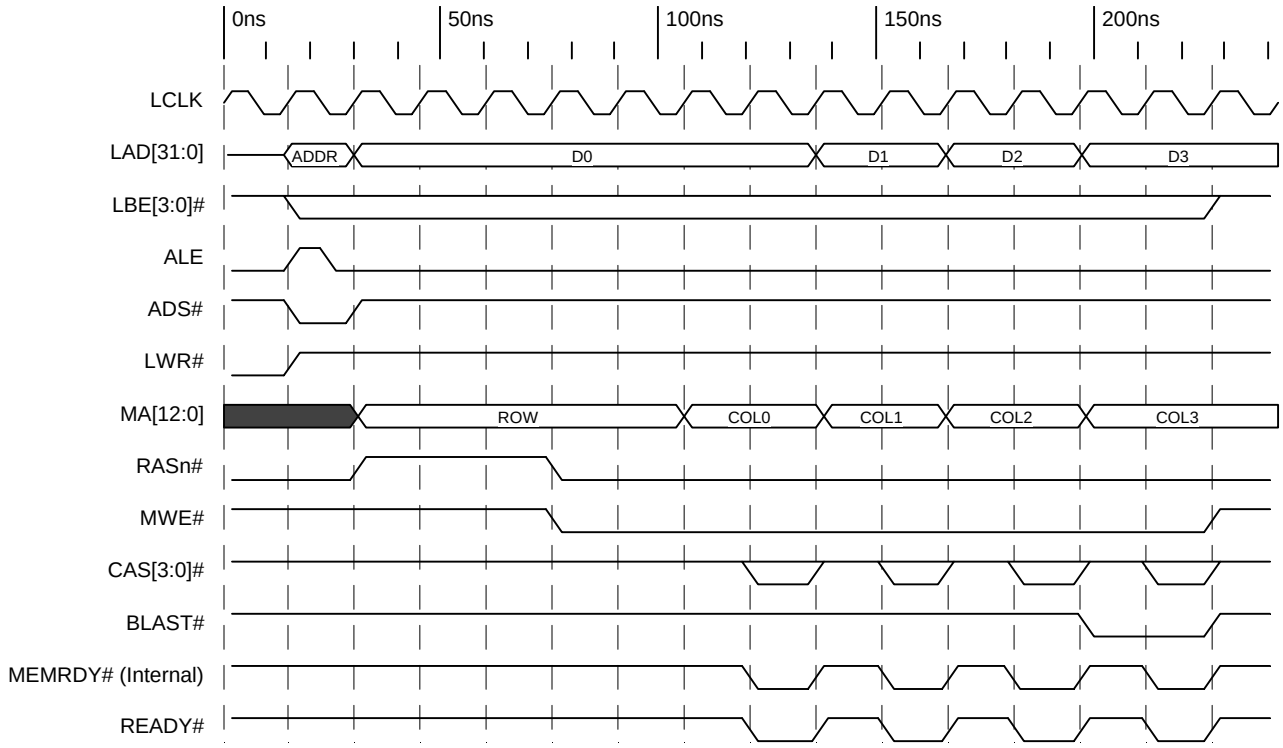
Timing Diagram 12-34. EDO DRAM Non-Page Mode Burst Write, 4-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, WCW=1, PRCG=3; Master=External Local Bus Master



Timing Diagram 12-35. EDO DRAM Page Hit Burst Write, 2-1-1-1 Wait States; C2C=1, WCW=1; Master=External Local Bus Master



Timing Diagram 12-36. EDO DRAM Page Hit Burst Write, 3-2-2-2 Wait States; C2C=1, WCW=2; Master=External Local Bus Master



Timing Diagram 12-37. EDO DRAM Page Miss Burst Write, 6-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, WCW=1, PRCG=3; Master=External Local Bus Master

12.3.2.8 EDO Read Access

An EDO read access is started when ADS# is asserted, LWR# low, and the address falls into the appropriate range. The MOE# signal is connected to OE# input of the DRAMs, and enables the output drivers during a Read access. If the IOP 480 is controlling the access, the row address is loaded into the MA[12:0] counter at the beginning of the period in which ADS# is asserted. If an external Local Bus Master is controlling the access, the row address is loaded into the MA[12:0] counter at the end of the period in which ADS# is asserted. The MWE# signal remains de-asserted throughout read the access.

After the Row Address to RAS delay, one of the RAS[3:0]# signals is asserted, strobing the row address into the device. After RAS to Column Address delay, the MA[12:0] address bus is switched to the column address. After the Column Address to CAS delay, CAS[3:0]# is asserted, strobing the column address into the device. For a 60 ns DRAM, Read data is available 60 ns after RAS# is asserted, 15 ns after CAS[3:0]# is asserted, and 30 ns after the column address is available. If a burst access is requested, then the MA[12:0] address is incremented and CAS[3:0]# is reasserted. Burst accesses are not permitted to cross a page boundary (refer to Table 12-16 on page 12-40).

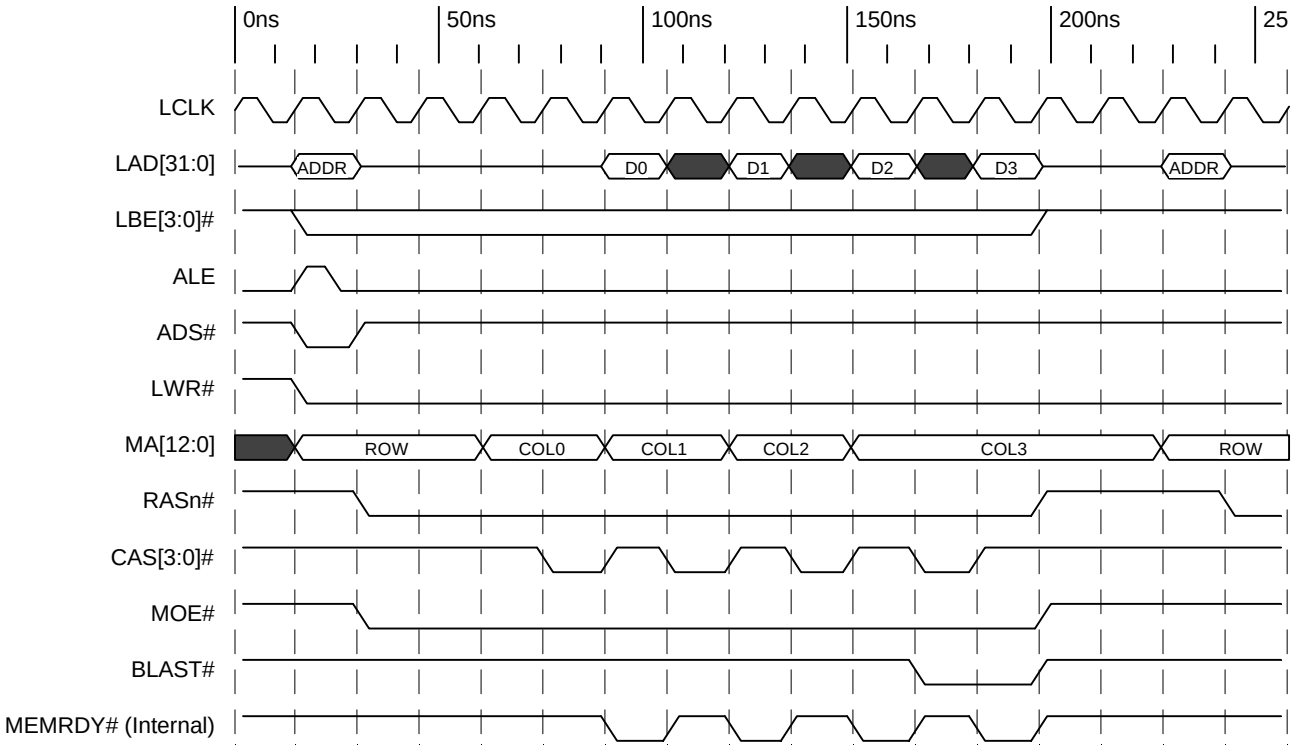
There are six types of programmable timing parameters associated with EDO DRAM read access, as illustrated in Table 12-21.

The timing parameter values shown at the top of each timing diagram refer to delay clock periods. This is not necessarily the value written to the corresponding field in the DRAMTIM register. The DRAMTIM register description details the mapping between the value written and resulting number of delay clock periods.

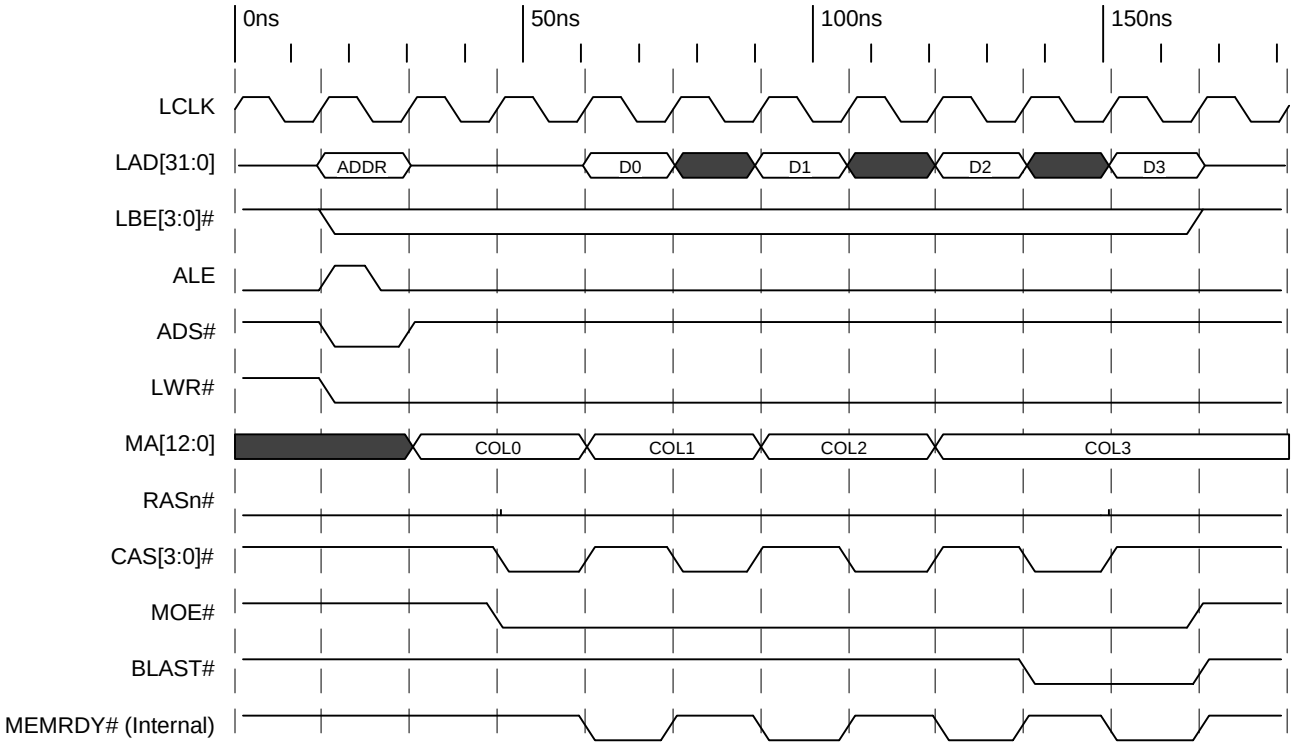
If the Ready/Recover Mode bit is set in the Bus Region Descriptor register, then the READY# pin is also driven during the recovery states. This can be used by an external Local Bus Master to delay the start of the next access until the recovery period has expired.

Table 12-21. EDO DRAM Read Access Programmable Timing Parameters

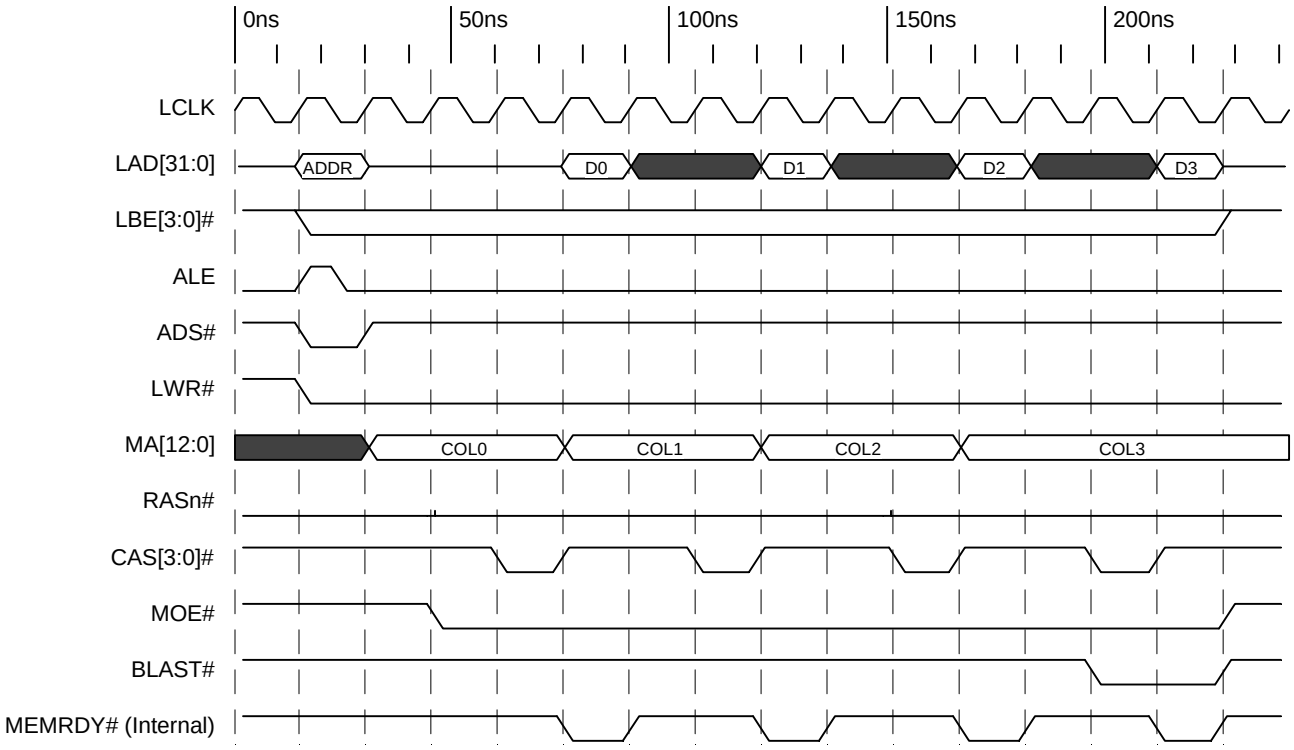
Field	Description
R2R	Row Address to RAS delay (1-4). Determines number of clock delays between the assertion of the row address and RAS[3:0]#. For a 66 MHz bus, this number is typically set to 1 clock for a 60 ns DRAM. For a heavily loaded MA[12:0] bus, or when using external buffers, this value may need to be increased.
R2C	RAS to Column Address delay (1-4). Determines number of clock delays between the assertion of RAS[3:0]# and first column address. For a 66 MHz bus, this number is typically set to 2 clocks for a 60 ns DRAM. For a heavily loaded MA[12:0] bus, or when using external buffers, this value may need to be increased.
C2C	Column Address to CAS delay (1-4). Determines number of clock delays between assertion of the column address and CAS[3:0]#. For a 66 MHz bus, this number is typically set to 1 clock for a 60 ns DRAM. For a heavily loaded MA[12:0] bus, or when using external buffers, this value may need to be increased. Note that during a page hit when IOP 480 is Bus Master, there is always one additional delay from column address to CAS[3:0]# because bus turnaround time is required between the address and first word of Read data.
RCW	Read CAS Width (1-4). Determines the width of each CAS[3:0]# pulse. For a 66 MHz bus, this number is typically set to 1 clock. The internal MEMRDY# signal is asserted in the last clock period during which CAS[3:0]# is asserted.
PRCG	Number of RAS precharge states (1-4). Determines the number of states between de-assertion of one RAS[3:0]# and assertion of the next RAS[3:0]#, and is used to provide adequate RAS precharge time. For a 66 MHz bus, this number is typically set to 3 clocks.
RRCV	Number of recovery states (1-4). Determines the number of extra wait states inserted at the end of DRAM access, and is used to provide the device with adequate time to float its data output buffers. For a 66 MHz bus, this number is typically set to 2 clocks.



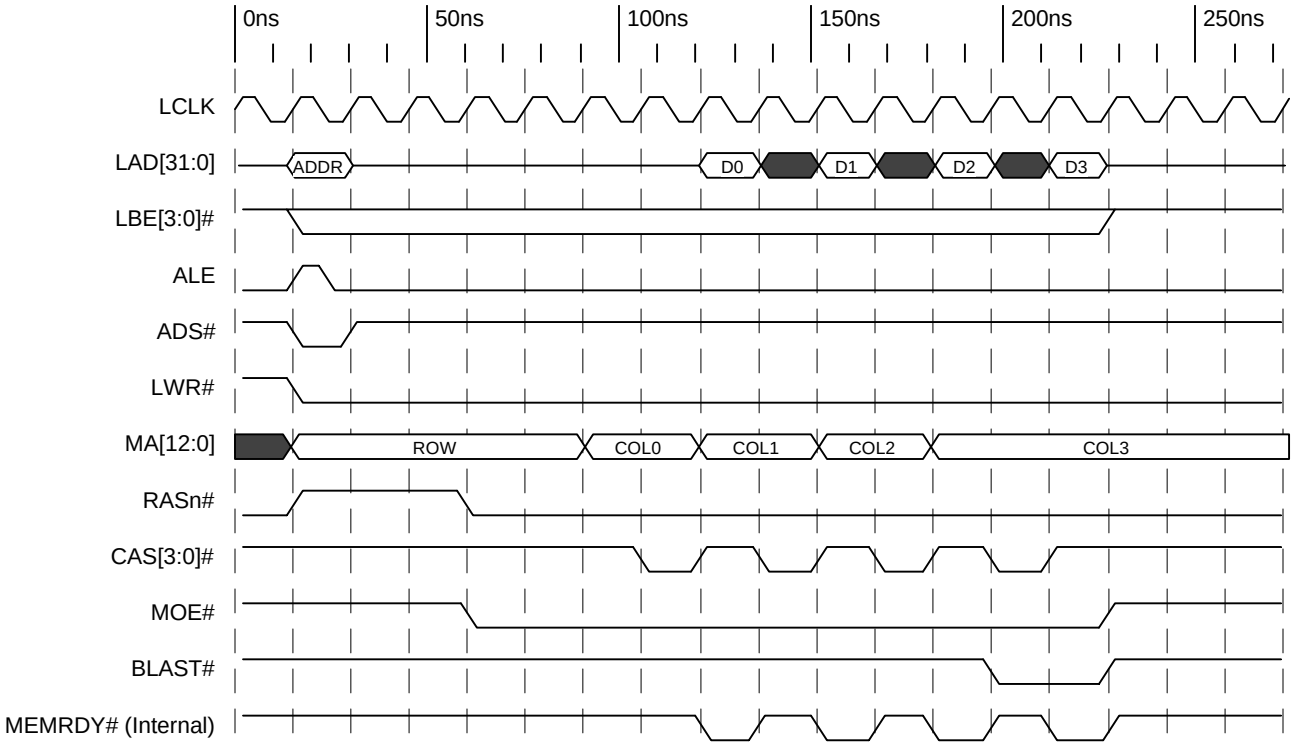
Timing Diagram 12-38. EDO DRAM Non-Page Mode Burst Read, 4-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, RCW=1, RRCV=2, PRCG=3; Master=IOP 480



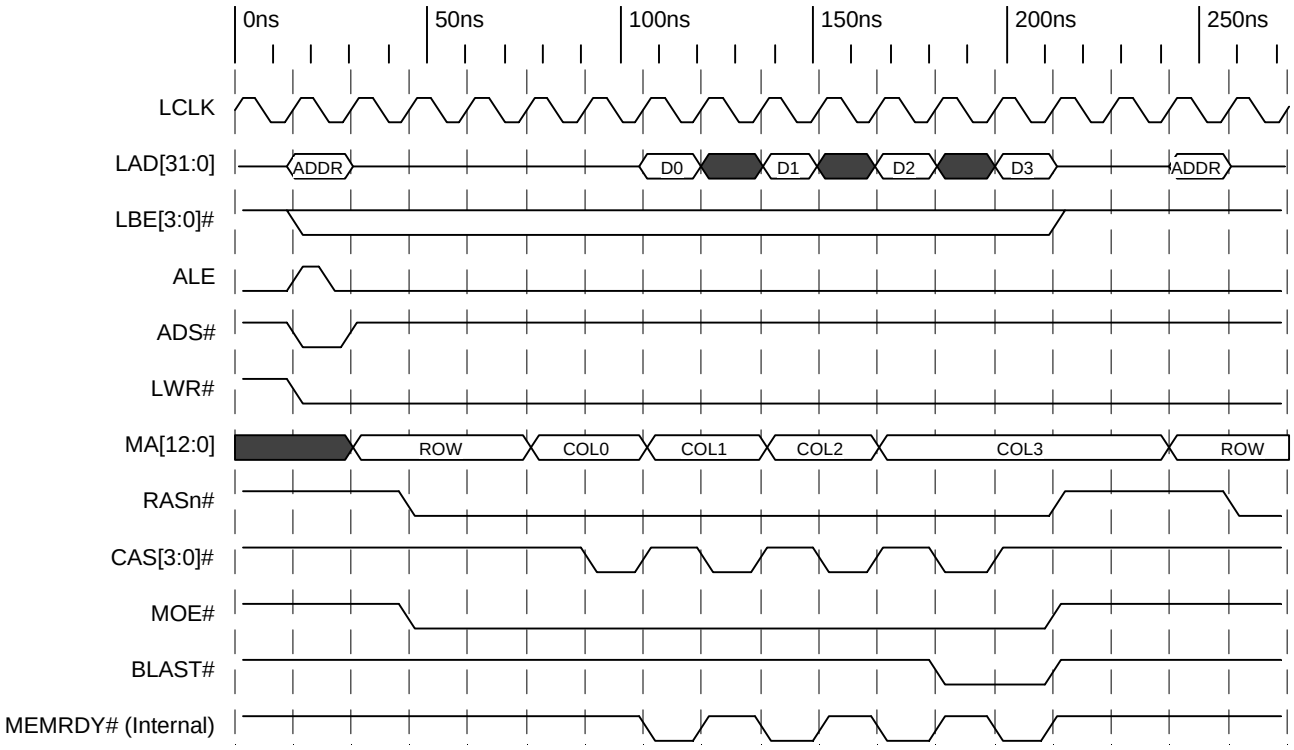
Timing Diagram 12-39. EDO DRAM Page Hit Burst Read, 2-1-1-1 Wait States; C2C=1, RCW=1; Master=IOP 480



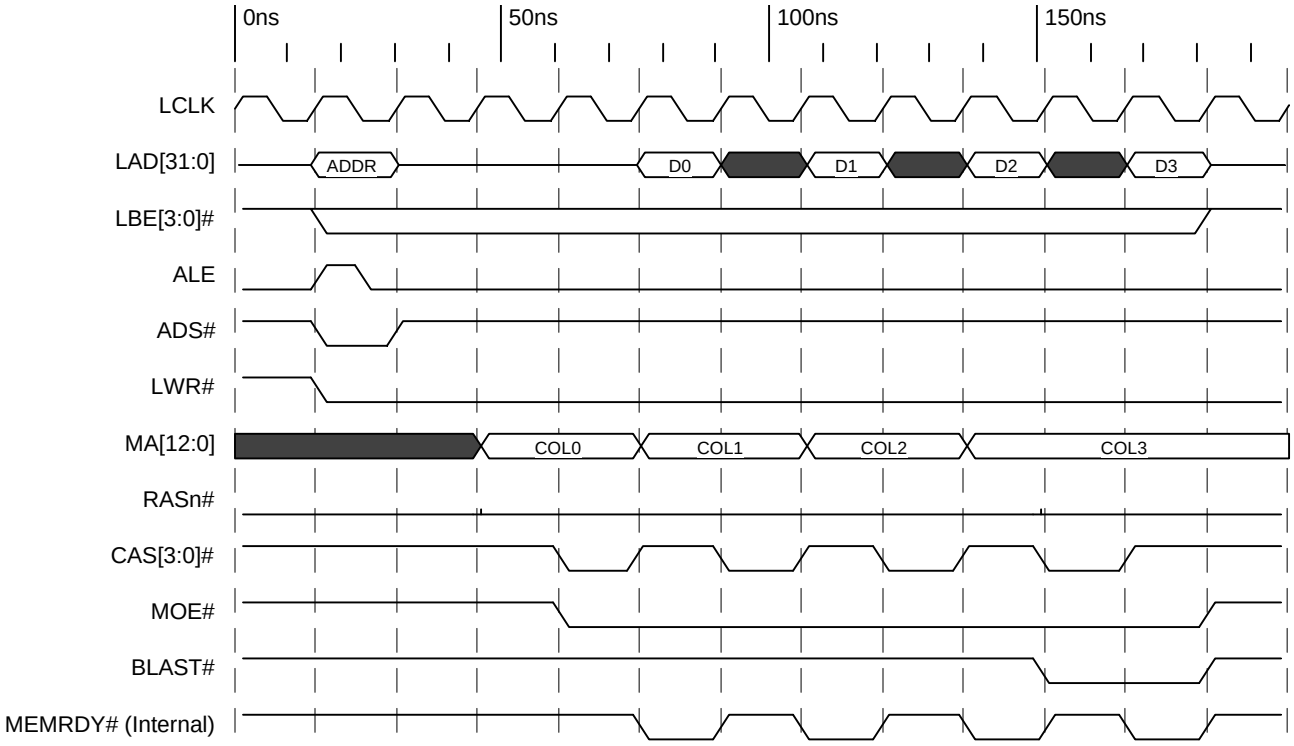
Timing Diagram 12-40. EDO DRAM Page Hit Burst Read, 3-2-2-2 Wait States; C2C=1, RCW=1; Master=IOP 480



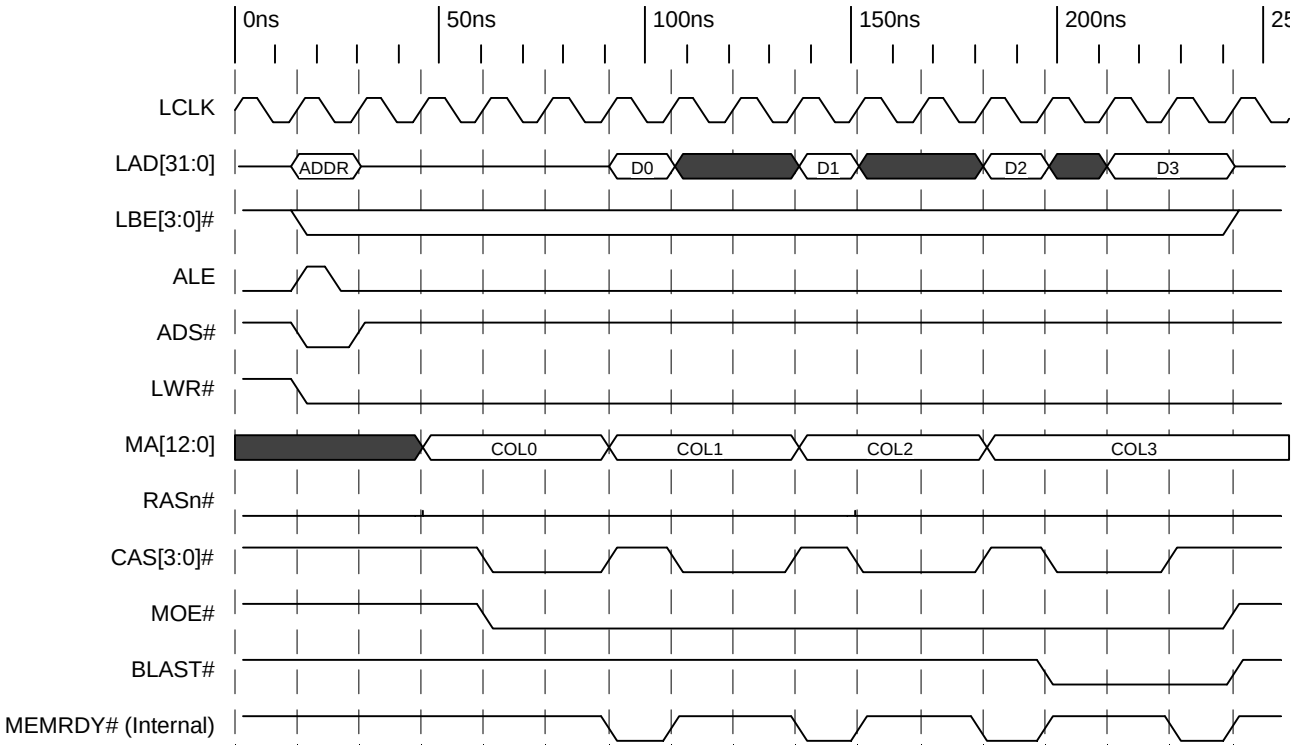
Timing Diagram 12-41. EDO DRAM Page Miss Burst Read, 6-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, RCW=1, PRCG=3; Master=IOP 480



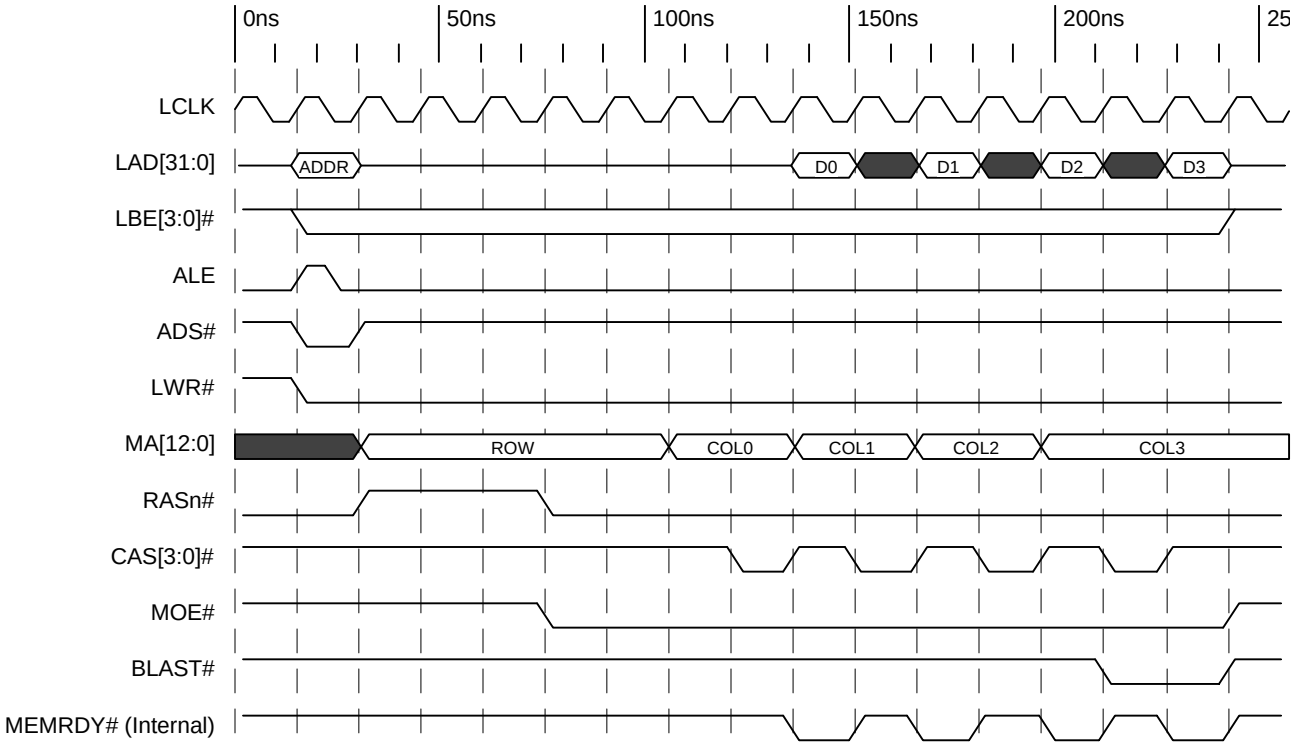
Timing Diagram 12-42. EDO DRAM Non-Page Mode Burst Read, 5-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, RCW=1, RRCV=2, PRCG=3; Master=External Local Bus Master



Timing Diagram 12-43. EDO DRAM Page Hit Burst Read, 3-1-1-1 Wait States; C2C=1, RCW=1; Master=External Local Bus Master



Timing Diagram 12-44. EDO DRAM Page Hit Burst Read, 3-2-2-2 Wait States; C2C=1, RCW=2; Master=External Local Bus Master



Timing Diagram 12-45. EDO DRAM Page Miss Burst Read, 7-1-1-1 Wait States; R2R=1, R2C=2, C2C=1, RCW=1, RRCV=2, PRCG=3; Master=External Local Bus Master

12.4 SIGNAL LOADING

12.4.1 SDRAMs

Signal Loading:

- MA[13:0]—5 pF
- MRAS#, MCAS#, MWE#, DQM[3:0]#, LCLK, MCKE, MCS[3:0]#—5 pF
- LAD[31:0]—6 pF

Table 12-22 illustrates SDRAM signal loading.

Because the IOP 480 can drive a maximum load of 50 pF, the maximum number of SDRAMs that can be attached without external buffering is eight. Using 4M x 16 devices yields four 16 MB banks, for a total of 64 MB of memory. Since inputs to SDRAM need to meet specified setup times, adding extra wait states does not compensate for heavily loaded signals. A slower Local Bus clock frequency would allow for a larger number of SDRAMs.

At 66 MHz, the pins maximum allowable load is reduced to 25 pF; therefore, the number of SDRAMs that can be attached is reduced.

12.4.2 EDO DRAMs

Signal Loading:

- MA[12:0]—5 pF
- RAS[3:0]#, CAS[3:0]#, MWE#, MOE#—7 pF
- LAD[31:0]—7 pF

Table 12-23 illustrates EDO Signal Loading.

Because the IOP 480 can drive a maximum load of 50 pF, the maximum number of EDO DRAMs that can be attached without external buffering is six. Using six 4M x 16 devices yields three 16 MB banks, for a total of 48 MB of memory.

If external buffers are required, then additional states in the memory access can be added, using the timing parameters listed in Table 12-24.

At 66 MHz, the pins maximum allowable load is reduced to 25 pF; therefore, the number of EDO DRAMs that can be attached is reduced.

12.5 OVERLAPPING ADDRESS SPACES

The IOP 480 monitors the Local Address Bus and decodes the following types of accesses—it is possible to program the configuration registers such that the address spaces overlap. In that case, the spaces decode with the priority listed in Table 12-25.

Table 12-22. SDRAM Signal Loading

Signal	Min Load (One Bank of Two x 16 Devices)	Max Load (Four Banks of Eight x 4 Devices)
MA[13:0]	10 pF	160 pF
MCAS# / MOE#	10 pF	160 pF
MCKE	10 pF	160 pF
LCLK	10 pF	160 pF
RAS[3:0]# / MCS[3:0]#	10 pF	40 pF
CAS[3:0]# / MDQM[3:0]#	5 pF	40 pF
MRAS#	10 pF	160 pF
MWE#	10 pF	160 pF
LAD[31:0]	6 pF	24 pF

Table 12-23. EDO Signal Loading

Signal	Min Load (One Bank of Two x 16 Devices)	Max Load (Four Banks of Eight x 4 Devices)
MA[12:0]	10 pF	160 pF
MCAS# / MOE#	14 pF	224 pF
MCKE	0 pF	0 pF
LCLK	0 pF	0 pF
RAS[3:0]# / MCS[3:0]#	14 pF	56 pF
CAS[3:0]# / MDQM[3:0]#	7 pF	56 pF
MRAS#	0 pF	0 pF
MWE#	14 pF	224 pF
LAD[31:0]	7 pF	28 pF

Table 12-24. EDO Timing Parameters

Field	Description
R2R	Row Address to RAS delay (1-4)
R2C	RAS to Column Address delay (1-4)
C2C	Column Address to CAS delay (1-4)
RCW	Read CAS Width (1-4)
WCW	Write CAS Width (1-4)

Table 12-25. Overlapping Address Spaces

Type of Access	Priority
Configuration Registers	1
Direct Master Access	2
LCS0 Memory Space	3
LCS1 Memory Space	4
LCS2 Memory Space	5
LCS3 Memory Space	6
DRAM Memory Space	7

13 INTELLIGENT I/O (I_2O)

13.1 OVERVIEW

The I_2O-*Ready* Messaging Unit supplies two paths for messages, two inbound FIFOs to receive messages from the primary PCI Bus, and two outbound FIFOs to pass messages to the primary PCI Bus. Refer to *I_2O Architecture Specification r2.0* for details.

Figure 13-1 and Figure 13-2 illustrate information about I_2O architecture.

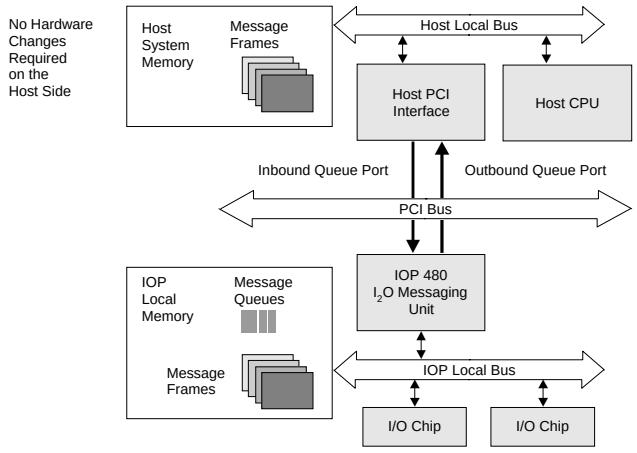


Figure 13-1. Typical I_2O Server/Adapter Card Design

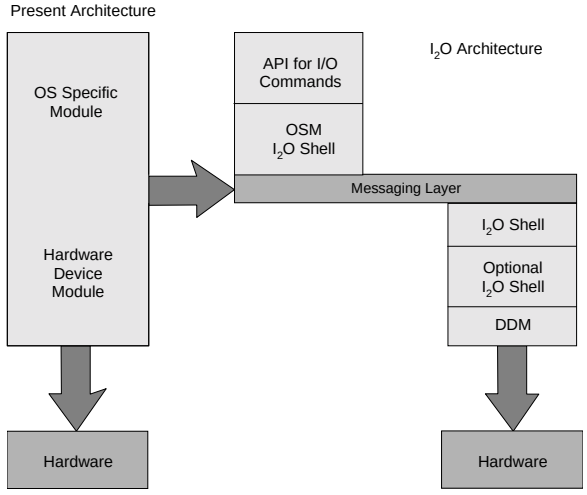


Figure 13-2. Driver Architecture Compared

13.2 REGISTERS USED

Refer to 17.2.3, “Messaging Queue Registers,” on page 17-4.

13.3 INBOUND MESSAGES

Inbound messages reside in a pool of message frames (minimum 64-byte frames) allocated in the shared Local Bus (IOP) memory. The inbound message queue is comprised of a pair of rotating FIFOs implemented in Local memory. The Inbound Free Queue FIFO holds the Message Frame Addresses (MFA) of available message frames in Local memory. The Inbound Post Queue FIFO holds the MFA of all currently posted messages.

Inbound circular FIFOs are accessed by external PCI agents through the Inbound Queue Port location in the PCI Address space. The Inbound Queue Port, when read by an external PCI agent, returns the Inbound Free Queue FIFO MFA. The external PCI agent places a message frame into the Inbound Post Queue FIFO by writing its MFA to the Inbound Queue Port location.

13.4 OUTBOUND MESSAGES

Outbound messages reside in a pool of message frames (minimum 64-byte frames) allocated in the shared PCI Bus (Host System) memory. The Outbound message queue is comprised of a pair of rotating FIFOs implemented in local memory. The Outbound Free Queue FIFO holds the Message Frame Addresses (MFA) of available message frames in system memory. The Outbound Post Queue FIFO holds the MFA of all currently posted messages.

Outbound circular FIFOs are accessed by external PCI agents through the Outbound Queue Port location in the PCI Address space. The Outbound Queue Port, when read by an external PCI agent, returns the Outbound Post Queue FIFO MFA. The External PCI agent places free message frames into the Outbound Free Queue FIFO by writing the free MFA into the Outbound Queue Port location.

Memory for the circular FIFOs themselves must be allocated in local (IOP) memory. The base address of the queue is contained in the Queue Base Address bits (QBAR[31:20]). Each FIFO entry is a 32-bit data value. Each read and write of the queue must be a single 32-bit access.

Circular FIFOs range in size from 128 to 64K entries. All four FIFOs must be the same size and contiguous. Therefore, the total amount of local memory needed for circular FIFOs ranges from 2 KB to 1 MB. FIFO size is specified in the Circular Queue Size bits (MQCR[3:1]).

The starting address of each FIFO is based on the Queue Base address and FIFO size.

Table 13-1. Queue Starting Address

FIFO	Starting Address
Inbound Free Queue	QBAR
Inbound Post Queue	QBAR + (1 * FIFO Size)
Outbound Post Queue	QBAR + (2 * FIFO Size)
Outbound Free Queue	QBAR + (3 * FIFO Size)

13.5 I_2O POINTER MANAGEMENT

FIFOs always reside in shared local (IOP) memory and are allocated and initialized by the IOP. Before setting the Queue Enable bit (MQCR[0]=1), the local processor must initialize the following registers, with the initial offset set according to configured FIFO size:

- Inbound Free Head and Tail Pointer registers (IFHPR, IFTPR)
- Inbound Post Head and Tail Pointer registers (IPHPR, IPTPR)
- Outbound Free Head and Tail Pointer registers (OFHPR, OFTPR)
- Outbound Post Head and Tail Pointer registers (OPHPR, OPTPR)

The Messaging Unit (MU) automatically adds the Queue Base Address to offset in each head and tail pointer register. The software can then enable I_2O. After initialization, the local software should not write to the pointers managed by the MU hardware.

Empty flags are set if the queues are disabled (MQCR[0]=0) and head and tail pointers are equal. This occurs independent of how the head and tail pointers are set.

An empty flag is cleared, signifying not empty, only if the queues are enabled and pointers become not equal.

If an empty flag is cleared and the queues are enabled, the empty flag is set only if the tail pointer is incremented and the head and tail pointers become equal.

Full flags are always cleared when the queues are disabled or the head and tail pointers are not equal.

A full flag is set when the queues are enabled, the head pointer is incremented, and the head and tail pointers become equal.

Each circular FIFO has a head pointer and a tail pointer, which are offsets from the Queue Base Address. Writes to a FIFO occur at the head of the FIFO and reads occur from the tail. Head and tail pointers are incremented by either the local processor or MU hardware. The unit that writes to the FIFO also maintains the pointer. Pointers are incremented after a FIFO access. Both pointers wrap around to the first address of the circular FIFO when they reach the FIFO size, so that the head and tail pointers continuously “chase” each other in the circular FIFO. MU wraps the pointers automatically for the pointers that it maintains. IOP software must wrap the pointers that it maintains. Whenever they are equal, the FIFO is empty. To prevent overflow conditions, I_2O specifies that the number of message frames allocated should be less than or equal to the number of entries in a FIFO. (Refer to Figure 13-3 on page 13-5.)

Each inbound MFA is specified by I_2O as the offset from the start of shared local (IOP) memory region 0 to the start of the message frame. Each outbound MFA is specified as the offset from Host memory location 0x00000000h to the start of the message frame in shared Host memory. Because MFA is an actual address, the message frames need not be contiguous. IOP allocates and initializes inbound message frames in shared IOP memory using any suitable memory allocation technique. Host allocates and initializes outbound message frames in shared Host memory using any suitable memory allocation technique. Message frames are a minimum of 64 bytes in length.

I_2O uses a “push” (write preferred) memory model. That means the IOP writes messages and data to the shared Host memory, and the Host writes messages and data to shared IOP memory. Software should make use of burst and DMA transfers whenever possible to ensure efficient use of the PCI Bus for message passing.

Additional information on message passing implementation may be found in the *I_2O Architecture Specification r2.0*.

13.6 INBOUND FREE QUEUE FIFO

The local processor allocates inbound message frames in its shared memory and can place the address of a free (available) message frame into the Inbound Free Queue FIFO by writing its MFA into the FIFO location pointed to by the Queue Base register + Inbound Free Head Pointer register. The Local processor must then increment the Inbound Free Head Pointer register.

A PCI Master (Host or another IOP) can obtain the MFA of a free message frame by reading the Inbound Queue Port Address (40h of the first PCI Memory Base Address register). If FIFO is empty (no free inbound message frames are currently available, head and tail pointers are equal), the MU returns -1 (FFFFFFFFh). If FIFO is not empty (head and tail pointers are not equal), the MU reads the MFA pointed to by the Queue Base register + Inbound Free Tail Pointer register, returns its value and increments the Inbound Free Tail Pointer register. If the Inbound Free Queue is not empty, and the Inbound Free Queue Prefetch Enable bit is set (QSR[3]=1), the next entry in the FIFO is read from the Local Bus into a prefetch register. The Prefetch register then provides data for the next PCI read from this queue, thereby reducing the number of PCI wait states. (Refer to Figure 13-3 on page 13-5.)

13.7 INBOUND POST QUEUE FIFO

A PCI master (Host or another IOP) can write a message into an available message frame in the shared local (IOP) memory. It can then post that message by writing the Message Frame Address (MFA) to the Inbound Queue Port Address, IQP (40h of the first PCI Memory Base Address register). When the port is written, the MU writes the MFA to the Inbound Post Queue FIFO location pointed to by the Queue Base register + FIFO Size + Inbound Post Head Pointer register. After the MU writes the MFA to the Inbound Post Queue FIFO, it increments the Inbound Post Head Pointer register.

The Inbound Post Tail Pointer register points to the Inbound Post Queue FIFO location which holds the MFA of the oldest posted message. The local processor maintains the tail pointer. After a local processor reads the oldest MFA, it can remove the MFA from the Inbound Post Queue FIFO by incrementing the Inbound Post Tail Pointer register.

The IOP 480 asserts a local Interrupt when the Inbound Post Queue FIFO is not empty. The Inbound Post Queue FIFO Interrupt bit in the Queue Status/ Control register (QSR[5]) indicates interrupt status. The Interrupt clears when the Inbound Post Queue FIFO is empty, and can be masked by the Inbound Post Queue FIFO Interrupt Mask bit (QSR[4]).

To prevent racing between the time the PCI write transaction is received until the data is written in local memory and the Inbound Post Head Pointer register is incremented, any Direct Slave access to the IOP 480 is issued a Retry until the write is completed.

13.8 OUTBOUND POST QUEUE FIFO

A Local Master (IOP) can write a message into an available message frame in shared Host memory. It can then post that message by writing the Message Frame Address (MFA) to the Outbound Post Queue FIFO location pointed to by the Queue Base register + Outbound Post Head Pointer register + (2 * FIFO Size). The Local processor should then increment the Outbound Post Head Pointer register.

A PCI Master can obtain the MFA of the oldest posted message by reading the Outbound Queue Port Address (44h of the first PCI Memory Base Address register). If the FIFO is empty (no more outbound messages are posted, head and tail pointers are equal), the MU returns -1 (FFFFFFFFh). If the Outbound Post Queue FIFO is not empty (head and tail pointers are not equal), the MU reads the MFA pointed to by the Queue Base register + (2 * FIFO Size) + outbound Post Tail Pointer register, returns its value, and increments the Outbound Post Tail Pointer register.

The IOP 480 asserts a PCI Interrupt when the Outbound Post Head Pointer register is not equal to the Outbound Post Tail Pointer register. The Outbound Post Queue FIFO Interrupt bit of the Outbound Post Queue Interrupt Status register (OPQIS) indicates the interrupt status. When the pointers become equal, both the interrupt and the Outbound Post Queue FIFO interrupt bit are automatically cleared. Pointers become equal when a

PCI Master (Host or other IOP) reads sufficient FIFO entries to empty the FIFO. The Outbound Post Queue FIFO Interrupt Mask register (OPQIM) can mask the interrupt.

To reduce read latency, prefetching from the tail of the queue occurs whenever the queue is not empty and the tail pointer is incremented (the queue has been read from), or when the queue is empty and the head pointer is incremented (the queue has been written to). Prefetching is enabled by setting QSR[2]. When the Host CPU reads the Outbound Post Queue, the data is immediately available.

13.9 OUTBOUND FREE QUEUE FIFO

The PCI Bus master (Host or other IOP) allocates outbound message frames in its shared memory. The PCI Bus master can place the address of a free (available) message frame into the Outbound Free Queue FIFO by writing a Message Frame Address (MFA) to the Outbound Queue Port Address (44h of the first PCI Memory Base Address register). When the port is written, the MU writes MFA to the Outbound Free Queue FIFO location pointed to by the Queue Base register + (3 * FIFO Size) + Outbound Free Head Pointer register. After the MU writes the MFA to the Outbound Free Queue FIFO, it increments the Outbound Free Head Pointer register.

When the IOP needs a free outbound message frame, it must first check whether any free frames are available. If the Outbound Free Queue FIFO is empty (outbound free head and tail pointers are equal), the IOP must wait for the Host to place additional outbound free message frames in the Outbound Free Queue FIFO. If the Outbound Free Queue FIFO is not empty (head and tail pointers are not equal), the IOP can obtain the MFA of the oldest free outbound message frame by reading the location pointed to by the Queue Base register + (3 * FIFO Size) + Outbound Free Tail Pointer register. After the IOP reads the MFA, it must increment the Outbound Free Tail Pointer register. To prevent overflow conditions, I_2O specifies the number of message frames allocated should be less than or equal to the number of entries in a FIFO. MU also checks for overflows of the Outbound Free Queue FIFO. When the Queue overflows, the MU asserts a local interrupt. The interrupt is recorded in the Queue Status/Control register (QSR[7]).

From the time the PCI Write transaction is received until the data is written into Local memory and the Outbound Free Head Pointer register is incremented, any Direct Slave access to the IOP 480 is issued a Retry.

13.10 I_2O ENABLE SEQUENCE

To enable I_2O, the local processor should perform the following:

- Initialize Space 0 address and range
- Initialize all FIFOs and Message Frame memory
- Set the PCI Base Class Code bits (PCICCR[23:16]) to be an I_2O device with programming interface 01h
- Set the I_2O Decode Enable bit (MQCR[0])
- Set Local Init Status bit to Done (DEVINIT[31]=1)

Note: *The serial EEPROM must not set the Local Init Status bit so that the IOP 480 issues Retries to all PCI accesses until the Local Init Status bit is set to Done by the Local Processor.*

All Memory-Mapped Configuration registers (for example, queue ports 40h and 44h) and Space 0 share the PCIBAR0 register. PCI accesses to offset 00h-3FFh of PCIBAR0 result in accesses to the IOP 480 Internal Configuration registers.

Accesses above offset 3FFh of PCIBAR0 result in Local Space accesses, beginning at offset 400h from the Remap PCI Address to Local Address Space 0 into Local Address Space bit(s) (LAS0BA[31:10]). Therefore, space located at offset 00h-3FFh from LAS0BA is not addressable from the PCI Bus using PCIBAR0. (Refer to Table 13-2, “Circular FIFO Summary,” on page 13-6.)

Note: *Because PCI accesses to offset 00h-3FFh of PCIBAR0 result in internal configuration accesses, Inbound Free MFAs must be greater than 3FFh.*

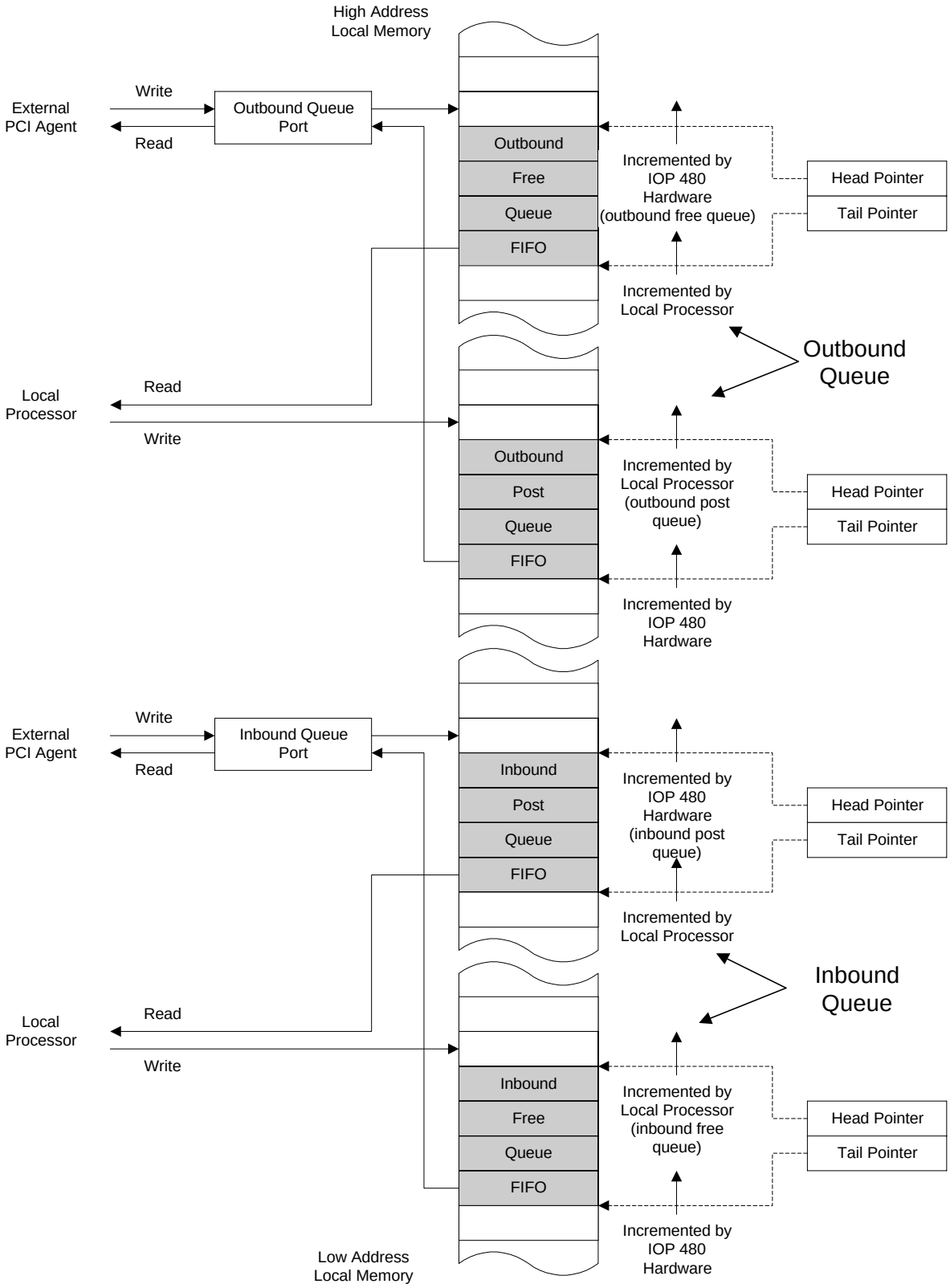


Figure 13-3. Circular FIFO Operation

Table 13-2. Circular FIFO Summary

FIFO Name	PCI Port	Generate PCI Interrupt	Generate Local Interrupt	Head Pointer Maintained	Tail Pointer Maintained by
Inbound Free Queue FIFO	Inbound Queue Port (Host read)	No	No	IOP 480	MU hardware
Inbound Post Queue FIFO	Inbound Queue Port (Host write)	No	Yes, when FIFO is not empty	MU hardware	Local processor
Outbound Post Queue FIFO	Outbound Queue Port (Host read)	Yes, when FIFO is not empty	No	IOP 480	MU hardware
Outbound Free Queue FIFO	Outbound Queue Port (Host write)	No	Yes, when FIFO overflows	MU hardware	Local processor

13.11 PERFORMANCE TUNING

13.11.1 Pull Option

To reduce the number of Host cycles on the PCI Bus, the IOP 480 may optionally provide a *pull option* capability. The Host places inbound messages in the Host system memory instead of the Local shared memory. Then the Host writes a special extended Message Frame Address (XMFA) into the Inbound Queue Port. XMFA contains the message Host system memory address. The IOP 480 then transfers the message from Host system memory to its own memory.

Message frames must be aligned on 16-byte boundaries, thus freeing the least significant four bits of the XMFA. The setting of bit 0 differentiates an XMFA from a standard MFA, while bits [3:1] indicate the number of blocks to transfer. The block size is established with the PullBlockSize parameter in the executive parameter group 0001h, and defaults to 16 bytes.

Extended message frames are released by the IOP 480 by writing to the HostFreeList FIFO structure in Host memory. Direct Master cycles can be used to perform these writes.

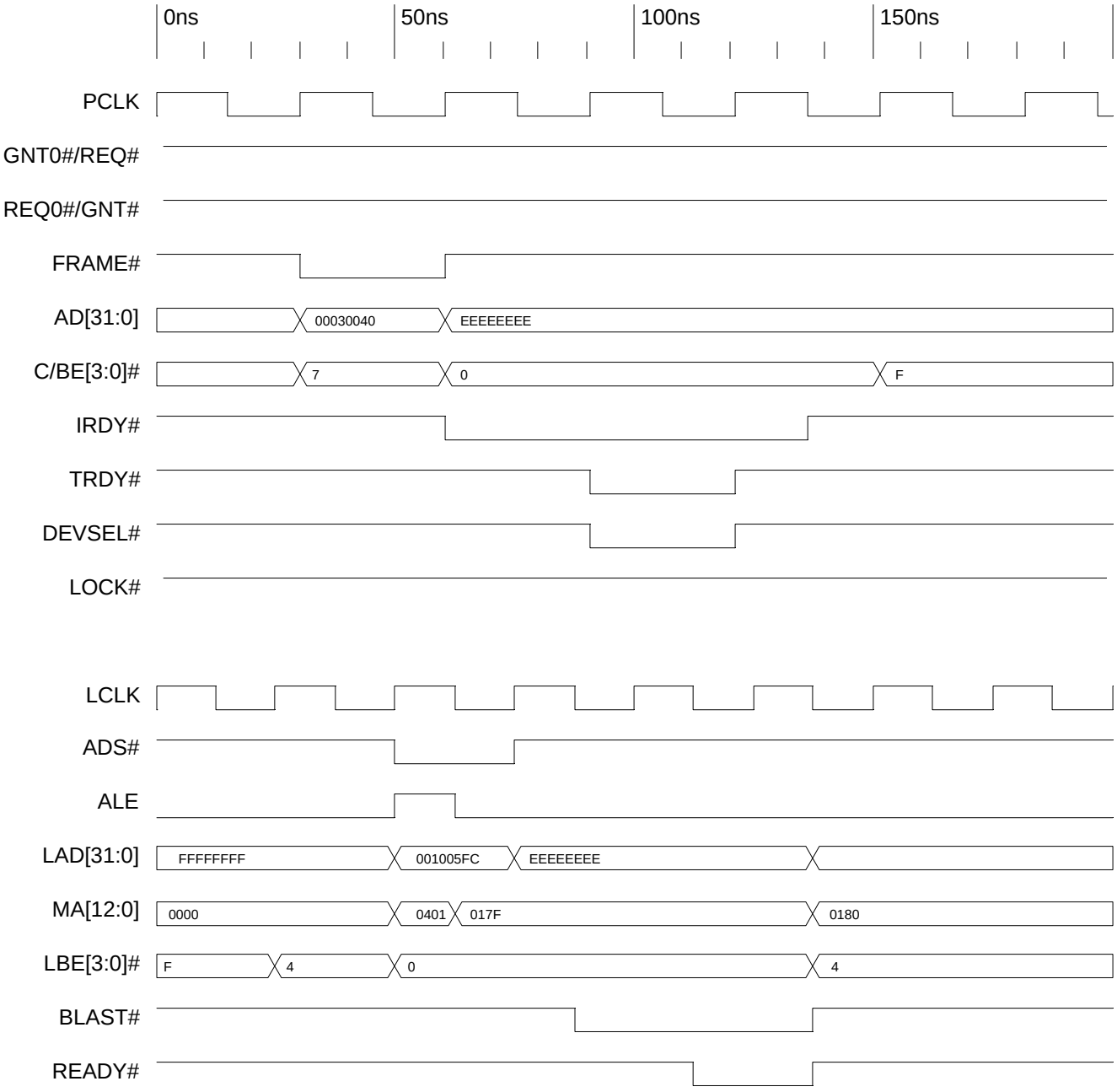
13.11.2 Outbound Option

This feature provides an enhanced mechanism for posting messages from the IOP 480 to the Host. Normally, the IOP 480 writes the MFA to the Outbound Post Queue, and the Host reads the message from local shared memory. Using the outbound option feature, the IOP 480 copies the message to the Host system memory and then posts the message by writing to a single location in the Host memory.

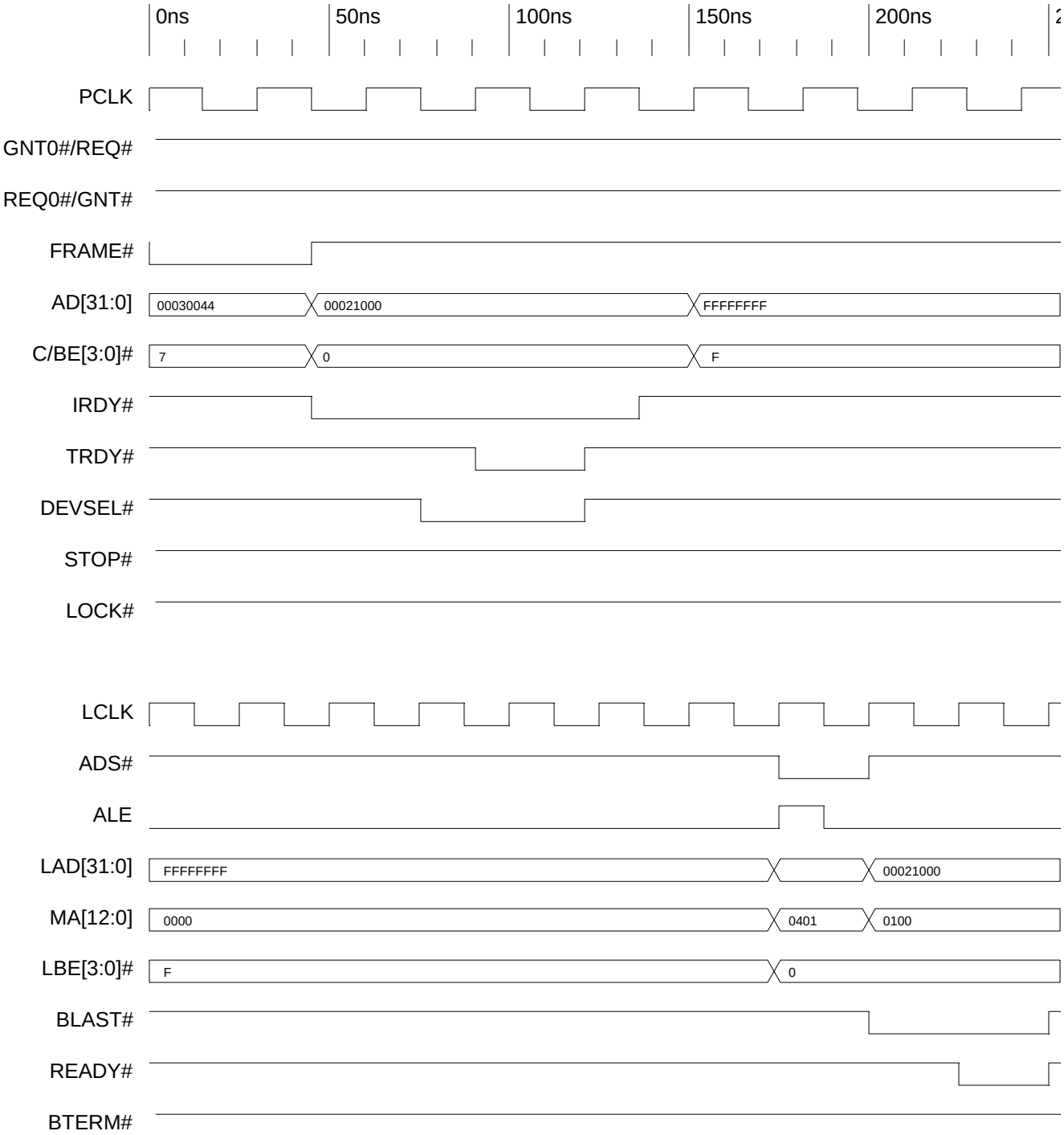
The IOP 480 provides two registers called IopOutboundIndex and HostOutboundIndex. The IOP 480 uses the IopOutboundIndex to keep track of where it writes the next MFA into the HostPostList FIFO in Host memory. This index register is initialized to zero, and is incremented by four each time the IOP 480 writes an MFA to the HostPostList FIFO.

The HostOutboundIndex register is written by the Host, and indicates to the IOP 480 how many of the posted messages have been processed by the Host. As long as the IopOutboundIndex does not equal the HostOutboundIndex, the Outbound Post Queue PCI interrupt should be generated, indicating that the Host needs to process more outbound messages from the IOP 480.

13.12 TIMING DIAGRAMS



Timing Diagram 13-1. I_2O Inbound Write Cycle



Timing Diagram 13-2. I_2O Outbound Write Cycle

14 COMPACTPCI HOT SWAP

The IOP 480 is a CompactPCI Hot Swap-*Friendly*-compliant device.

14.1 OVERVIEW

Hot Swap is used for CompactPCI applications. Hot Swap functionality allows the orderly insertion and removal of boards without adversely affecting system operation. This is done for repair of faulty boards or system reconfiguration. Additionally, Hot Swap provides access to Hot Swap services, allowing system reconfiguration and fault recovery to occur with no system downtime and minimum operator interaction. Adapter insertion/removal logic control resides on individual adapters. The IOP 480 uses two pins, ENUM# and LEDon/LEDin, to implement the hardware aspects of Hot Swap functionality. The IOP 480 uses the Hot Swap Capabilities register to implement the software aspects of Hot Swap.

To avoid confusion in the industry, Hot Swap defines three levels of compatibility:

- Hot Swap-*Capable* devices contain the minimum requirements to operate in a Hot Swap environment
- Hot Swap-*Friendly* devices contain additional functions to ease the designer's job
- Hot Swap-*Ready* devices contain all necessary functions for Hot Swap

Hot Swap-*Capable* requirements are mandatory for a device to be used in a Hot Swap environment. These requirements are attributes for which a system user must compensate using external circuitry, as follows:

- *PCI Local Bus Specification r2.1* compliance
- Tolerate Vcc from early power
- Tolerate asynchronous reset
- Tolerate precharge voltage
- I/O Buffers must meet modified V/I requirements
- Limited I/O pin leakage at precharge voltage

Hot Swap-*Friendly* silicon includes all required *Capable* functions and adds others from the following list. The IOP 480 integrated these functions into the PCI silicon, thereby reducing the amount and cost of required external circuitry.

- **Incorporates Hot Swap Control/Status register (HS_CSR)**—Contained within the configuration space.
- **Incorporates an Extended Capability Pointer (ECP) mechanism**—It is required that Software retain a standard method of determining if a specific function is designed in accordance with the specification. The Capabilities Pointer is located within standard CSR space, using a bit in the PCI Status register (offset 04h).
- Incorporates remaining software connection control resources. Provides ENUM#, Hot Swap switch, and the blue LED.

The IOP 480 is a Hot Swap-*Friendly* PCI silicon device. The IOP 480 incorporates all compliant functions defined by the CompactPCI Hot Swap specification. The IOP 480 incorporates ENUM#, LEDon/LEDin, and Hot Swap Capabilities registers—HSCAPID, HSNEXT, and HSCSR.

14.2 CONTROLLING CONNECTION PROCESSES

The following sections are excerpts from the *PICMG 2.1, CompactPCI Hot Swap Specification, r1.0*. Refer to the specification for more details.

14.2.1 Hardware Connection Control

Hardware Control provides a means for the platform to control the hardware connection process. The signals listed in the following sections must be supported on all Hot Swap boards for interoperability. Implementations on different platforms may vary.

14.2.1.1 Board Slot Control

BD_SEL# is one of the shortest pins. It is driven low to enable power-on. For systems not implementing hardware control, it is grounded on the backplane.

Systems implementing hardware control radially connect BD_SEL# to a Hot Swap Controller (HSC). The controller terminates the signal with a weak pull-down. The controller can detect board present when the board pull-up overrides the pull-down. HSC can then control the power-on process by driving BD_SEL# low.

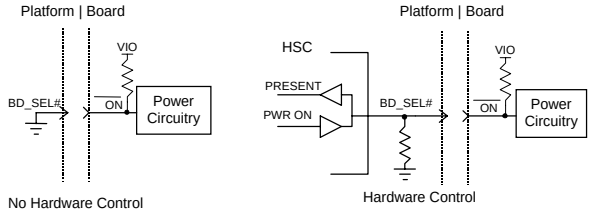


Figure 14-1. Redirection of BD_SEL#

14.2.1.2 Board Healthy

A second radial signal is used to acknowledge board health. It signals that a board is suitable to be released from reset and allowed onto the PCI Bus.

Minimally, this signal must be connected to the card's power controller "power good" status line. Use of HEALTHY# can be expanded for applications requiring additional conditions to be met for the board to be considered healthy.

On platforms that do not use Hardware Connection Control, this line is not monitored. Platforms implementing this signaling route these signals radially to a Hot Swap Controller.

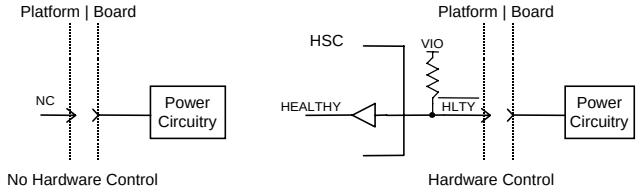


Figure 14-2. Board Healthy

14.2.1.3 Platform Reset

Reset (PCI_RST#), as defined by *CompactPCI Specification*, is a bused signal on the backplane, driven by the Host. Platforms may implement this signal as a radial signal from the Hot Swap Controller to further control the electrical connection process. To maintain function of the bused signal, platforms that do this must *OR* the Host reset signal with the slot-specific signal.

Locally, boards must not come out of reset until the H1 State is reached (healthy), but they must also honor the backplane reset. The Local board reset (Local_PCI_RST#) must be the logical *OR* of these two conditions. Local_PCI_RST# is connected to the IOP 480 RST# input pin.

During a BIOS voltage precharge and platform reset, in insertion and extraction procedures, all PCI I/O buffers must be in a High Impedance state. The IOP 480 supports this condition any time the Host RST# is asserted (*PCI r2.1* or better). To protect the Local board components from early power, the IOP 480 floats the Local Bus I/Os. The TEST pin can be used to perform the High Impedance condition on the Local Bus. Both the RST# and TEST signals can be simultaneously asserted. The TEST signal is de-asserted some time before the Host RST# is de-asserted to ensure the IOP 480 asserts the LRESETo# signal to complete a reset task of the Local board.

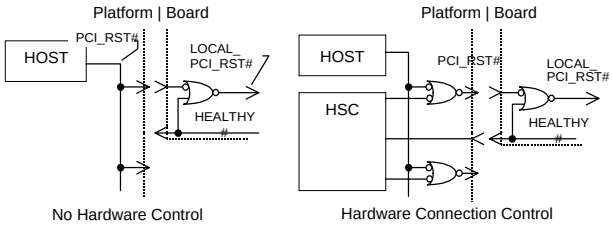


Figure 14-3. PCI Reset

14.2.2 Software Connection Control

Software Connection Control provides a means to control the Software Connection Process. Resources on the hot swap board facilitate software Connection Control. Access to these resources occurs by way of the bus, using PCI protocol transfers (in-band).

These resources consist of four elements:

- ENUM# driven active indicates the need to change the state of the Hot Swap board
- A switch, tied to the ejector, indicates the intent to remove a board
- LED indicates the status of the software connection process
- Control/Status register allows the software to interact with these resources

14.2.2.1 Ejector Switch and Blue LED

A microswitch (switch), located in the card-ejector mechanism of the Hot Swap CompactPCI board, is used to signal the impending removal of a board. This signal asserts ENUM#. The operator normally activates the switch, waits for the LED illumination to indicate it is okay to remove the board, and then removes the board. The IOP 480 implements control logic for both the microswitch and the Blue LED in one pin (LEDon/LEDin).

When the ejector is opened or closed, the switch bounces for a time. The IOP 480 uses internal debounce circuitry to clean the signal before the remainder of Hot Swap logic acknowledges it. The state of the switch is acknowledged six times, at 1 ms intervals, before it is assumed closed or open.

The Blue “Status” LED, located on the front of the Hot Swap CompactPCI board, is illuminated when it is permissible to remove a board. The hardware connection layer provides protection for the system during all insertions and extractions. This LED indicates the system software is in a state that tolerates board extraction.

Upon insertion, the LED is automatically illuminated by the hardware until the hardware connection process completes. The LED remains *OFF* until the software uses it to indicate extraction is once again permitted.

The IOP 480 uses a tri-state I/O pin to drive the external LED. This pin is Time-Division Multiplexed (TDM) for input and output functionality. When an output, it drives the external LED. The LED state is driven from the LED Software On/Off Switch bit (HSCSR[3]). When used as an input, it acknowledges the state of the ejector handle. With the implementation of TDM, this pin is usually driving the LED. A small portion of time is dedicated to acknowledging ejector status.

14.2.2.2 ENUM#

ENUM# is provided to notify the Host CPU that a board has been freshly inserted or is about to be removed. This signal informs the CPU that configuration of the system has changed. The CPU then performs any necessary maintenance such as installing a device driver upon board insertion, or quiescing a device driver prior to board extraction.

ENUM# is an open collector bused signal with a pull-up on the Host. It may drive an interrupt (preferred) or be polled by the system software at regular intervals. The CompactPCI Hot-Plug System Driver on the system Host manages the ENUM# sensing. Full Hot Swap Boards assert ENUM# until serviced by the Hot-Plug system driver.

When a board is inserted into the system and reset, the IOP 480 acknowledges the state of the ejector switch. If this switch is open (ejector closed) the IOP 480 asserts ENUM# interrupt and sets the ENUM# Status Indicator for Board Insertion bit (HSCSR[7]). Once the Host CPU has installed the proper drivers, it can logically include this board by clearing the interrupt.

When a board is about to be removed, the IOP 480 acknowledges the ejector switch is closed (ejector open), asserts ENUM# interrupt and sets the ENUM# Status Indicator for Board Removal bit (HSCSR[6]). The Host then logically removes the board and turns on the LED. The operator can then remove the board completely.

14.2.2.3 Hot Swap Control/Status Register (HSCSR)

The IOP 480 supports Hot Swap directly, as a Control/Status register is provided in Configuration space. This register is accessed through the PCI Extended Capabilities Pointer (ECP) mechanism.

The Hot Swap Control/Status register (HSCSR) provides status read-back for the Hot-Plug System Driver to determine which board is driving ENUM#. This register is also used to control the Hot Swap Status LED on the front panel of the board, and to de-assert ENUM#.

14.2.2.3.1 Hot Swap Capabilities Register

Register 14-1. Hot Swap Capabilities Register

31 24	23 16	15 8	7 0
Reserved	Hot Swap Control (06h)	Next_Cap Pointer	Hot Swap ID (00h)

Hot Swap ID. Bits [7:0] (HSCAPID[7:0]; PCI:54h, LOC:354h). These bits are set to a default value of 00h. User must set the bits to 06h for the bits to be valid.

Next_Cap Pointer. Bits [15:8] (HSNEXT[7:0]; PCI:55h, LOC:355h). These bits point to the next New Capability structure or are set to 0, if this is the last capability in the structure.

Hot Swap Control. Bits [23:16] (HSCSR[7:0]; PCI:56h, LOC:356h). This eight-bit control register is defined as follows (refer to Table 14-1).

Table 14-1. Hot Swap Control

Bit	Description
23	ENUM# status—Insertion (1 = board is inserted).
22	ENUM# status—Removal (1 = board is being removed).
21	Not used.
20	Not used.
19	LED state (1 = LED on, 0 = LED off).
18	Not used.
17	ENUM# interrupt enable (1 = de-assert, 0 = enable interrupt).
16	Not used.

15 VITAL PRODUCT DATA

15.1 OVERVIEW

The Vital Product Data (VPD) function in *PCI Specification r2.2* defines a new location and access method. It also defines the Read Only and Read/Write bits. Currently a Device ID, Vendor ID, Revision ID, Class Code, Subsystem ID, and Subsystem Vendor ID are required in the Configuration Space Header and are used for basic identification of the device and device configuration. Although this information allows device configuration, it is not sufficient to allow a device to be uniquely identified. With the addition of VPD, optional information is provided allowing devices to be uniquely identified and tracked. These additional bits enable current and/or future support tools and reduce total PC and system cost of ownership.

This provides an alternate access method other than Expansion ROM for VPD. VPD is stored in an external serial EEPROM, which is accessed using the Capabilities List function.

15.2 VPD REGISTERS

- **VPD ID.** (VPD_CAP[7:0]; PCI:58h, LOC:358h). Value of 3h is assigned by the PCI SIG. The VPD ID is hardcoded.
- **Next_Cap Pointer.** (VPD_CAP[15:8]; PCI:58h, LOC:358h). Either points to next New Capability structure, or set to 0 if this is the last one. The IOP 480 Defaults to 0h. Value can be overwritten from the Local Bus.
- **VPD Address.** (VPD_CAP[30:16]; PCI:58h, LOC:358h). VPD Byte address accessed. All accesses are 32-bit wide; bits [17:16] must be set to 0, with the maximum serial EEPROM size being 4K bits. Bits [30:28] are ignored.

Note: *Address size allows for a 4K bit serial EEPROM.*

- **F Flag.** (VPD_CAP[31]; PCI:58h, LOC:358h). Flag used to indicate when a serial EEPROM data operation is complete. For Write cycles, the four bytes of data are first written into the VPD Data bits, after which the VPD Address is written at the same time the F flag is set to 1. The F flag clears when a serial EEPROM data transfer completes. For Read cycles, the VPD Address is written at the same time the F flag is cleared to 0. The F flag is set when four bytes of data are read from the serial EEPROM.
- **VPD Data.** (VPD_DATA[31:0]; PCI:5Ch, LOC:35Ch). VPD data is written or read through this register. Four bytes are always transferred between this register and the serial EEPROM.
  - For VPD_DATA[31:16], these bits go into address N (specified in VPD_CAP[30:16]).
  - For VPD_DATA[15:0], these bits go into address N+1.

Note: *VPD Data bits [31:16] correspond to VPD word at the address specified by the VPD Address register.*

15.2.1 VPD Registers

Register 15-1. VPD Registers

31	30 16	15 8	7 0
F Flag	VPD Address	Next_Cap Pointer (0h)	VPD ID (3h)
VPD Data			

15.3 SERIAL EEPROM VPD PARTITIONING

To support VPD, the serial EEPROM is partitioned into read only and read/write sections.

15.4 SEQUENTIAL READ AREA

At power-up, the first 2048 bits, 256 bytes of the serial EEPROM contain read-only information. The read-only portion of the serial EEPROM is loaded into the IOP 480. This occurs once after power-on, or by writing to the Reload Configuration Registers bit (DEVINIT[29]) using a sequential read command to the serial EEPROM.

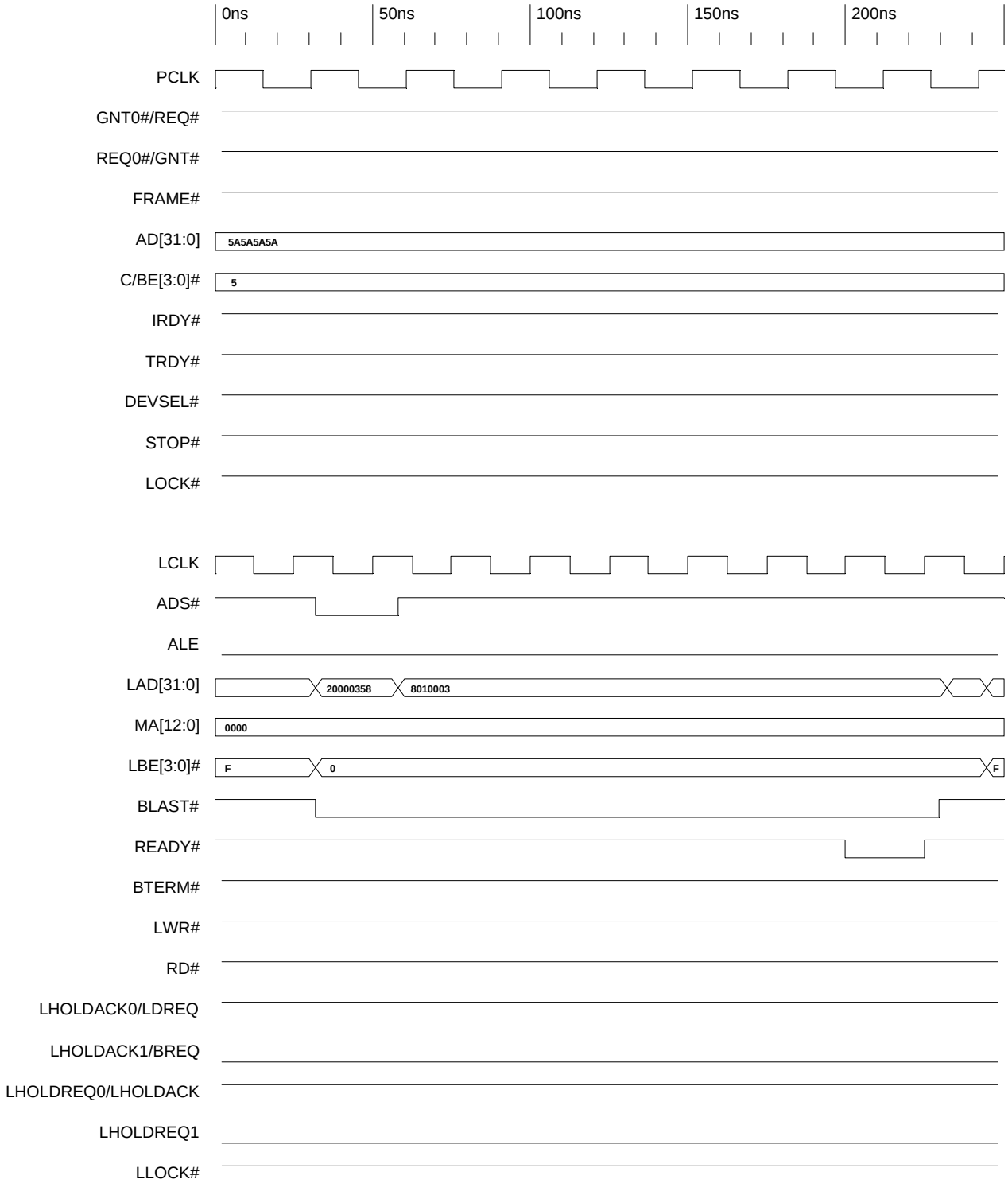
15.5 RANDOM READ AND WRITE AREA

The IOP 480 can read and write the read/write portion of the serial EEPROM. Serial EEPROM locations starting at the address specified by the Serial EEPROM Protected Area bits (DEVINIT[14:8]) are read/write. This register is loaded upon power-on and can be written with any value starting at location 0. This provides the capability of writing the entire serial EEPROM. Writes to the serial EEPROM are comprised of the following three commands:

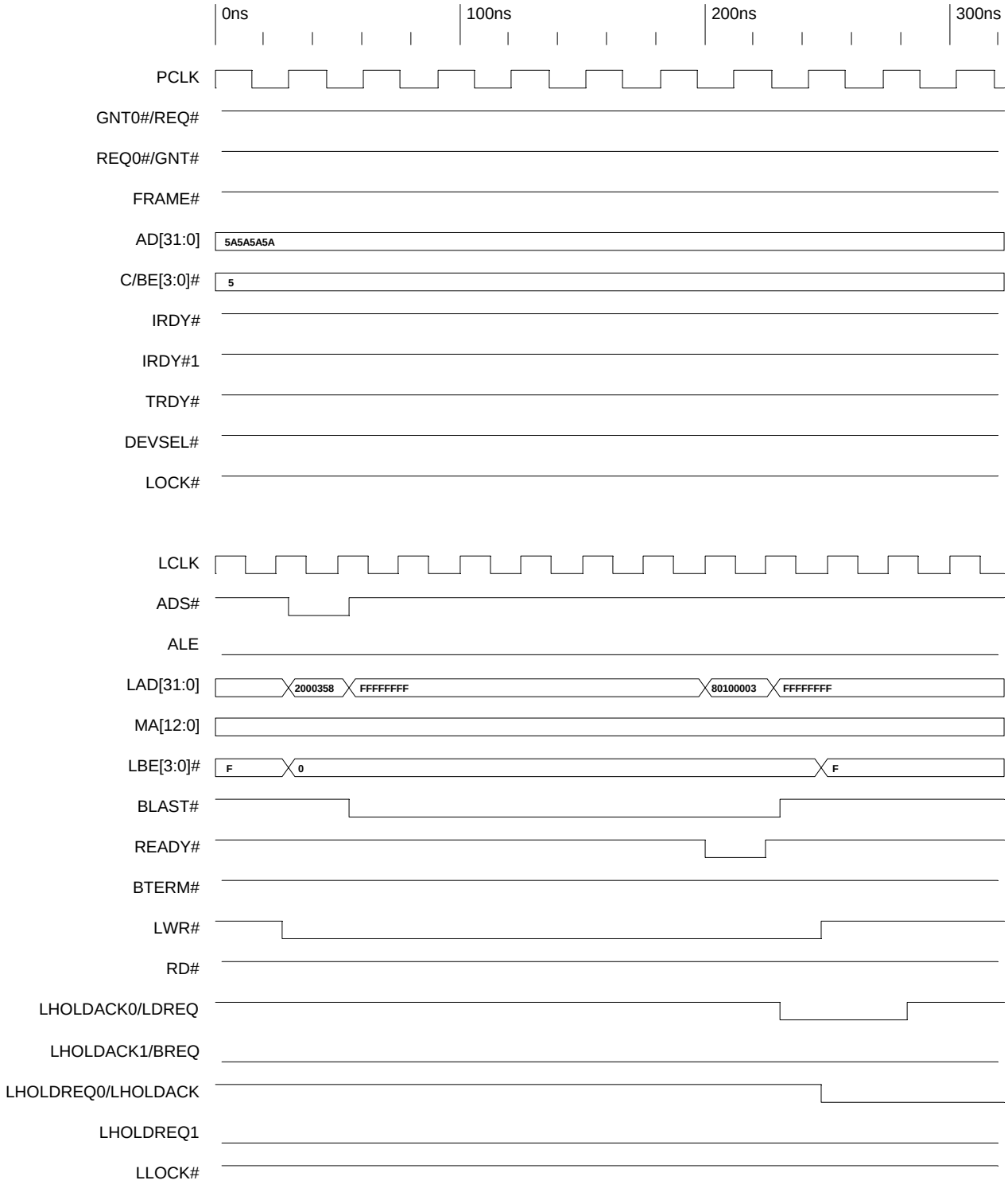
- Write Enable
- Write, followed by Write data
- Write Disable

This is done to ensure against accidental writing to the serial EEPROM. Random cycles allow VPD information to be written and read at any time.

15.6 TIMING DIAGRAMS



Timing Diagram 15-1. Register Write to Start VPD Write



Timing Diagram 15-2. Register Read to Show Completion of VPD Read

16 POWER MANAGEMENT

16.1 OVERVIEW

The *PCI Bus Power Management Interface Specification, r1.1* defines four power states for PCI functions—D_0, D_1, D_2, and D_3. States D_0 and D_3 are required, while states D_1 and D_2 are optional. State D_0 represents the highest power consumption and state D_3 the least. While the IOP 480 supports the message passing of these power management states, it is the responsibility of software to implement the actual power savings.

- **D_0 (Uninitialized)**—Enters this state from a power-on reset or from state D_3. Must be initialized to function properly.
- **D_0 (Active)**—All functions active.
- **D_1**—Uses less power than state D_0, and more than state D_2. No PCI Bus Mastering cycles allowed.
- **D_2**—Uses less power than state D_1 and more power than state D_3. PCI Bus Mastering cycles are not allowed. Clock may be stopped.
- **D_{3hot}**—Uses lower power than any other state. Supports PCI Configuration cycles to function if a clock is running. Supports Wakeup Event from function, but not standard PCI interrupts. When programmed for state D_0, an internal soft reset occurs. All content is lost in this state.
- **D_{3cold}**—No power. Supports bus reset only. All context is lost in this state.

16.2 FUNCTIONAL DESCRIPTION

The PCI Status register (PCISR) and the Capability List Pointer register (CAP_PTR) indicate whether a new capability (the Power Management function) is available. The Capabilities List Status bit (PCISR[4]) enables the PCI BIOS to determine whether the IOP 480 supports Capabilities List Features. This bit is writeable from the Local Bus, and can be read from the Local and PCI Buses. CAP_PTR provides an offset into PCI Configuration space, indicating the starting location of the first item in a capabilities list.

The Power Management Capability ID register (PMCAPID) specifies the Power Management Capability ID, 01h, assigned by the PCI SIG. The Power Management Next Capability Pointer register (PMNEXT) points to the first location of the next item in the Capabilities Linked list. If Power Management is the last item in the list, then this register should be set to 0. The default value for the IOP 480 is 54h (Hot Swap).

For the IOP 480 to change the power state and assert PME#, the Local Host or PCI Host should set the PME_En bit (PMCSR[8]=1). The Local Host then determines to which power state the backplane should change by reading the Power_State bits (PMCSR[1:0]). The Local Host can either be the internal IOP 480 CPU or an external mastering device.

The Local Host sets up the following:

- D_2_Support and D_1_Support bits (PMC[10:9]) are used by the Local Host to identify power state support
- PME_Support bits (PMC[14:11]) are used by the IOP 480 to identify the PME# Support corresponding to a given power state (PMCSR[1:0])

The Local Host then sets the PME_Status bit (PMCSR[15]=1) and the IOP 480 asserts PME#. To clear the PME# Interrupt Status bit, the PCI Host must write a 1 to the PME_Status bit (PMCSR[15]=1). To disable the PME# Interrupt signal, either host can write a 0 to the PME_En bit (PMCSR[8]=0).

INTO is asserted each time the power state in the PMCSR register changes. It is up to the Local Host to implement the Power Savings mode request. A transition from state 11 (D_{3hot}) to state 00 (D_0) causes a soft reset. This should be initiated only from the PCI Bus because the Local Bus interface is reset during a soft reset (PMCSR[1:0]). All context is lost in this state.

The Data_Scale bits (PMCSR[14:13]) indicate the scaling factor to use when interpreting the value of the Power Management Data bits (PMDATA[7:0]). The value and meaning of these bits depend upon the data value specified in the Data_Select bits (PMCSR[12:9]). The Data_Scale bit value is unique for each Data_Select bit. For Data_Select values from 8 to 15, (PMCSR[12:9]=1000 to PMCSR[12:9]=1111), the Data_Scale bits always return a zero (PMCSR[14:13]=0). This data is optional.

PMDATA provides operating data, such as power consumed or heat dissipation. This data is optional.

16.3 SYSTEM CHANGES POWER MODE EXAMPLE

1. The Host writes to the IOP 480 PMCSR register to change the power states.

2. The IOP 480 sends a Local interrupt (INTO) to a Local CPU (LCPU).

3. The LCPU has 200 ms to read the power management information from the IOP 480 PMCSR register to implement the power saving function.

4. After the LCPU implements the power saving function, the IOP 480 disables all Direct Slave accesses and PCI Interrupt output (INTA#). In addition, the BIOS disables the IOP 480 Master Enable bit (PCICR[2]).

Notes: *In Power Saving mode, all PCI and Local Configuration cycles are granted.*

The IOP 480 automatically performs a soft reset to a Local Bus on D_3-to-D_0 transitions.

16.4 WAKE-UP REQUEST EXAMPLE

1. The add-in card (with a IOP 480 chip installed) is in a powered-down state.

2. The Local CPU performs a write to the IOP 480 Power Management Control/Status register (PMCSR[1:0]) to request a change in power state.

3. When the request is detected, the IOP 480 drives PME# out to the PCI Bus.

4. The PCI Host accesses the IOP 480 PMCSR register to disable the PME# output signal and restores the IOP 480 to the D_0 power state.

5. The IOP 480 completes the power management task by issuing the Local interrupt (INTO) to the Local CPU, indicating that the power mode has changed.

16.5 POWER CONSUMPTION

Power Consumption features include:

- Typical full operational power at 66 MHz Local and 33 MHz PCI—600 mW
- Power with clock, but not with other ongoing operations—150 mW
- Power without clock—6 mW

17 REGISTER SUMMARY

17.1 INTERNAL REGISTER ACCESS

The IOP 480 provides several internal registers, allowing for maximum flexibility in the bus interface design and performance. These registers are accessible from the PCI Bus, the internal IOP 480 CPU, or the Local Bus. They include the following:

- PCI Configuration registers
- Local Configuration registers
- Memory Controller registers
- DMA registers
- Mailbox registers
- PCI-to-Local and Local-to-PCI Doorbell registers
- Messaging Queue registers (I_2O)
- Power Management registers
- Hot Swap registers
- VPD registers
- Runtime registers

Figure 17-1 illustrates how these registers are accessed.

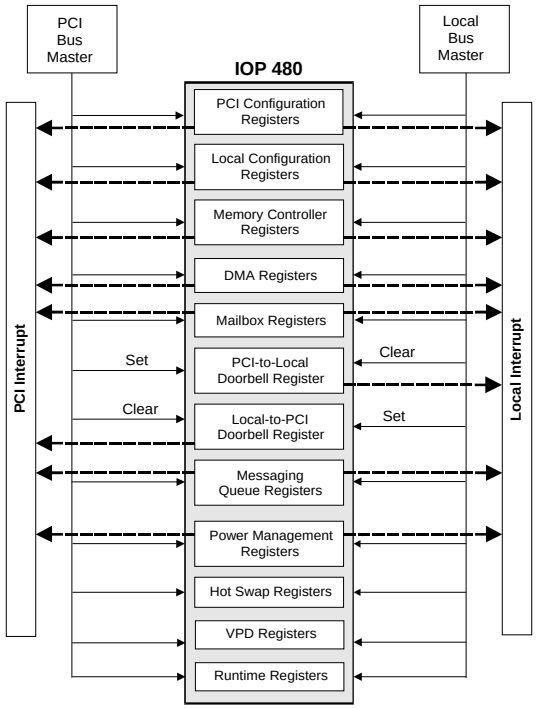


Figure 17-1. IOP 480 Internal Register Access

The IOP 480 PCI Configuration registers can be accessed from the PCI Bus with a Configuration Type 0 or Type 1 cycle.

All other IOP 480 internal registers can be accessed from PCI by hitting space zero. Space zero begins at the address in the PCI Base Address 0 (PCIBAR0[31:10]) for the IOP 480 Memory-Mapped Configuration register. The internal registers take up 1K of Space 0.

PCI read or write accesses to IOP 480 registers can be Byte, Word, or Lword accesses. All PCI Memory accesses to IOP 480 registers can be Burst or Non-Burst accesses.

All IOP 480 internal registers can be accessed from the Local Bus (or the IOP 480 CPU) by hitting the Local Configuration Space. This space starts at the value programmed into CFGBA which, by default, is 0x5000-0000.

Accessing reserved registers returns 0 for reads and has no affect on writes.

Table 17-1 lists the symbols used for identifying Read and Write information in the following register listings.

Table 17-1. Read/Write Symbols Used in Register Listings

Symbol	Description
P	PCI Bus
L	Local Bus
E	Serial EEPROM

Note: *Unused bits should always be written with 0.*

17.2 REGISTER ADDRESS MAPPING

17.2.1 Configuration Register Base Addresses

The IOP 480 contains six groups of Configuration registers. Table 17-2 lists the base address of each group.

Table 17-2. Configuration Register Base Addresses

Register Set	PCI Configuration Offset	PCI Base Address 0 (PCIBAR0) Offset	Local Address Offset
PCI Configuration Registers	00h	—	300h
Messaging Queue Registers	—	000h	000h
Local Configuration Registers	—	080h	080h
Memory Controller Registers	—	100h	100h
Runtime Registers	—	180h	180h
DMA Registers	—	200h	200h

17.2.2 PCI Configuration Registers

Table 17-3. PCI Configuration Registers

PCI Offset (CFG)	Local Offset from CFGBA	Mnemonic	Description 31 30 24 23 16 15 8 7 0	Write
00h	300h	PCI_PCIID	Device ID / Vendor ID	L, E
04h	304h	PCI_PCICR	Status / Command	P, L
08h	308h	PCI_PCIREV	Class Code / Revision ID	L, E
0Ch	30Ch	PCI_PCICLSR	BIST / Header Type / PCI Latency Timer / Cache Line Size	P, L
10h	310h	PCI_PCIBAR0	PCI Base Address 0 for Memory-Mapped Configuration Registers and Local Address Space 0	P, L
14h	314h	PCI_PCIBAR1	PCI Base Address 1 for Local Address Space 1	P, L
18h	318h	PCI_PCIBAR2	PCI Base Address 2 for Local Address Space 2	P, L
1Ch	31Ch	—	Unused Base Address	—
20h	320h	—	Unused Base Address	—
24h	324h	—	Unused Base Address	—
28h	328h	—	Cardbus CIS Pointer (*Not Supported*)	—
2Ch	32Ch	PCI_PCISUBID	Subsystem ID / Subsystem Vendor ID	L, E
30h	330h	PCI_PCIERBAR	PCI Base Address for Local Expansion ROM	P, L
34h	334h	PCI_PCICAPPTR	***Reserved*** / Cap_Pointer	L, E
38h	338h	—	***Reserved***	—
3Ch	33Ch	PCI_PCIILR	Max_Lat / Min_Gnt / Interrupt Pin / Interrupt Line	P, L, E
40h	340h	PMCAP	Power Management Capabilities / Next_Cap Pointer / Capability ID	L, E
44h	344h	PMCSR	Data / Bridge Extensions / Power Management CSR	P, L
48h	348h	PMSCALE	***Reserved*** / Power Management Scale Values	L, E
4Ch	34Ch	PWRCON	Power Consumed	L, E
50h	350h	PWRDIS	Power Dissipated	L, E
54h	354h	HS0	***Reserved*** / Hot Swap Control / Next_Cap Pointer / Hot Swap ID	L, E
58h	358h	VPDCAP	Flag / VPD Address / Next_Cap Pointer / VPD ID	L
5Ch	35Ch	VPDDATA	VPD Data	L
60h-FFh	360h-3FFh	—	***Reserved***	—

17.2.3 Messaging Queue Registers

Table 17-4. Messaging Queue Registers

PCI Offset from PCIBAR0	Local Offset from CFGBA	Mnemonic	Register Description	Write
000h	000h	MQCR	Messaging Queue Configuration	P, L
004h	004h	QBAR	Queue Base Address	P, L
008h	008h	IFHPR	Inbound Free Head Pointer	P, L
00Ch	00Ch	IFTPR	Inbound Free Tail Pointer	P, L
010h	010h	IPHPR	Inbound Post Head Pointer	P, L
014h	014h	IPTPR	Inbound Post Tail Pointer	P, L
018h	018h	OFHPR	Outbound Free Head Pointer	P, L
01Ch	01Ch	OFTPR	Outbound Free Tail Pointer	P, L
020h	020h	OPHPR	Outbound Post Head Pointer	P, L
024h	024h	OPTPR	Outbound Post Tail Pointer	P, L
028h	028h	QSR	Queue Status/Control	P, L
02Ch	02Ch	—	***Reserved***	—
030h	030h	OPQIS	Outbound Post Queue Interrupt Status	—
034h	034h	OPQIM	Outbound Post Queue Interrupt Mask	P, L
038h	038h	—	***Reserved***	—
03Ch	03Ch	—	***Reserved***	—
040h	—	IQPR	Inbound Queue Port	P
044h	—	OQPR	Outbound Queue Port	P
048h-04Ch	048h-04Ch	—	***Reserved***	—
050h	050h	HOSTOUTIDX	Host Outbound Index	P, L
054h	054h	IOPOUTIDX	IOP Outbound Index	P, L
058h-07Fh	058h-07Fh	—	***Reserved***	—

Notes:

When I_2O messaging is enabled (MQC[0]=1), a PCI Master (Host or another IOP) uses the Inbound Queue Port to read Messaging Frame Addresses (MFAs) from the Inbound Free Queue and to write MFAs to the Inbound Post Queue. It uses the Outbound Queue Port to read MFAs from the Outbound Post Queue and to write MFAs to the Outbound Free Queue.

Each Inbound MFA is specified by I_2O as the offset from the PCI Base Address 0 (programmed in register PCIBAR0 at offset 10H) to the start of the message frame. This means that all inbound message frames should reside in PCI Base Address 0 Memory Space.

Each Outbound MFA is specified by I_2O as the offset from system address 0x00000000h. The Outbound MFA is the physical 32-bit address of the frame in shared PCI system memory.

The Inbound and Outbound Queues reside in the Local Address space.

17.2.4 Local Configuration Registers

Table 17-5. Local Configuration Registers

PCI Offset from PCIBAR0	Local Offset from CFGBA	Mnemonic	Register Description	Write
080h	080h	DEVINIT	Device Initialization	P, L, E
084h	084h	LOCCTL	Local Bus Control	P, L, E
088h	088h	LOCTMO	Local Bus Timeout	P, L, E
08Ch	08Ch	LOCTMR	Local Timers	P, L, E
090h	090h	LARBR	Local/DMA Arbitration	P, L, E
094h	094h	BIGEND	Big/Little Endian	P, L, E
098h	098h	PCICTL	PCI Bus Control	P, L, E
09Ch-09Fh	09Ch-09Fh	—	***Reserved***	—
0A0h	0A0h	LAS0RR	Range for Memory-Mapped Configuration Registers and PCI-to-Local Address Space 0	P, L, E
0A4h	0A4h	LAS0BA	Local Base Address (Remap) for PCI to Local Address Space 0	P, L, E
0A8h	0A8h	LAS1RR	Range for PCI-to-Local Address Space 1	P, L, E
0ACh	0ACh	LAS1BA	Local Base Address (Remap) for PCI to Local Address Space 1	P, L, E
0B0h	0B0h	LAS2RR	Range for PCI-to-Local Address Space 2	P, L, E
0B4h	0B4h	LAS2BA	Local Base Address (Remap) for PCI to Local Address Space 2	P, L, E
0B8h-0BFh	0B8h-0BFh	—	***Reserved***	—
0C0h	0C0h	EROMRR	Range for PCI-to-Local Expansion ROM	P, L, E
0C4h	0C4h	EROMBA	Local Base Address (Remap) for PCI-to-Local Expansion ROM	P, L, E
0C8h	0C8h	DMRR	Range for Direct Master-to-PCI	P, L, E
0CCh	0CCh	DMLBAM	Local Base Address for Direct Master-to-PCI Memory	P, L, E
0D0h	0D0h	DMPBAM	PCI Base Address (Remap) for Direct Master-to-PCI (Lower 32 Bits)	P, L, E
0D4h	0D4h	DMDAC	PCI Base Address (Remap) for Direct Master-to-PCI (Upper 32 Bits)	P, L, E
0D8h	0D8h	DMLBAI	Local Base Address for Direct Master-to-PCI IO/CFG	P, L, E
0DCh	0DCh	DMCFGA	PCI Configuration Address Register for Direct Master-to-PCI IO/CFG	P, L, E
0E0h	0E0h	CFGBA	Local Base Address for Configuration Register Access	P, L, E
0E4h	0E4h	UARTBA	IOP 480 CPU Base Address for UART Access	P, L, E
0E8h	0E8h	PLXID	PLX Device and Vendor ID	—
0ECh	0ECh	PLXREV	PLX Silicon Revision	—
0F0h-0FFh	0F0h-0FFh	—	***Reserved***	—

17.2.5 Memory Controller Registers

Table 17-6. Memory Controller Registers

PCI Offset from PCIBAR0	Local Offset from CFGBA	Mnemonic	Register Description	Write
100h	100h	LCS0BRD	LCS0# Bus Region Descriptor	P, L, E
104h	104h	LCS0WT	LCS0# Write Timing	P, L, E
108h	108h	LCS0RT	LCS0# Read Timing	P, L, E
10Ch	10Ch	LCS0BASE	LCS0# Base Address	P, L, E
110h	110h	LCS0RANGE	LCS0# Range	P, L, E
114h	114h	LCS1BRD	LCS1# Bus Region Descriptor	P, L, E
118h	118h	LCS1WT	LCS1# Write Timing	P, L, E
11Ch	11Ch	LCS1RT	LCS1# Read Timing	P, L, E
120h	120h	LCS1BASE	LCS1# Base Address	P, L, E
124h	124h	LCS1RANGE	LCS1# Range	P, L, E
128h	128h	LCS2BRD	LCS2# Bus Region Descriptor	P, L, E
12Ch	12Ch	LCS2WT	LCS2# Write Timing	P, L, E
130h	130h	LCS2RT	LCS2# Read Timing	P, L, E
134h	134h	LCS2BASE	LCS2# Base Address	P, L, E
138h	138h	LCS2RANGE	LCS2# Range Register	P, L, E
13Ch	13Ch	LCS3BRD	LCS3# Bus Region Descriptor	P, L, E
140h	140h	LCS3WT	LCS3# Write Timing	P, L, E
144h	144h	LCS3RT	LCS3# Read Timing	P, L, E
148h	148h	LCS3BASE	LCS3# Base Address	P, L, E
14Ch	14Ch	LCS3RANGE	LCS3# Range	P, L, E
150h	150h	DRAMBRD	DRAM Bus Region Descriptor	P, L, E
154h	154h	DRAMCTL	DRAM Control	P, L, E
158h	158h	DRAMINIT	DRAM Initialization	P, L, E
15Ch	15Ch	DRAMTIM	DRAM Timing Parameters	P, L, E
160h	160h	DRAMBASE	DRAM Base Address	P, L, E
164h	164h	DRAMRANGE	DRAM Range	P, L, E
168h	168h	DFLTBRD	Default Bus Region Descriptor	P, L, E
16Ch-17Fh	16Ch-17Fh	—	***Reserved***	—

17.2.6 Runtime Registers

Table 17-7. Runtime Registers

PCI Offset from PCIBAR0	Local Offset from CFGBA	Mnemonic	Register Description	Write
180h	180h	MBOX0	Mailbox Register 0	P, L, E
184h	184h	MBOX1	Mailbox Register 1	P, L, E
188h	188h	MBOX2	Mailbox Register 2	P, L
18Ch	18Ch	MBOX3	Mailbox Register 3	P, L
190h	190h	MBOX4	Mailbox Register 4	P, L
194h	194h	MBOX5	Mailbox Register 5	P, L
198h	198h	MBOX6	Mailbox Register 6	P, L
19Ch	19Ch	MBOX7	Mailbox Register 7	P, L
1A0h	1A0h	P2LDBELL	PCI-to-Local Doorbell Register	P, L
1A4h	1A4h	L2PDBELL	Local-to-PCI Doorbell Register	P, L
1A8h	1A8h	—	***Reserved***	—
1ACh	1ACh	—	***Reserved***	—
1B0h	1B0h	PINTSTAT	PCI Interrupt Status	P, L
1B4h	1B4h	PINTENB	PCI Interrupt Enable	P, L
1B8h	1B8h	LINTSTAT	Local Interrupt Status	P, L
1BCh	1BCh	LINTENB	Local Interrupt Enable	P, L
1C0h	1C0h	PABTADR	PCI Abort Address	P, L
1C4h-1FFh	1C4h-1FFh	—	***Reserved***	—

17.2.7 DMA Registers

Table 17-8. DMA Registers

PCI Offset from PCIBAR0	Local Offset from CFGBA	Mnemonic	Register Description	Write
200h	200h	C0MODE	Ch 0 Mode	P, L
204h	204h	C0CSR	Ch 0 Control/Status	P, L
208h	208h	C0COUNT	Ch 0 Transfer Count, Valid Bit	P, L
20Ch	20Ch	C0PCILADR	Ch 0 PCI Address (Lower 32 Bits)	P, L
210h	210h	C0LOCADR	Ch 0 Local Address	P, L
214h	214h	C0DESCPTR	Ch 0 Descriptor Pointer	P, L
218h	218h	C0PCIHADR	Ch 0 PCI Address (Upper 32 Bits)	P, L
21Ch	21Ch	C0THRES	Ch 0 Threshold Register	P, L
220h	220h	C1MODE	Ch 1 Mode	P, L
224h	224h	C1CSR	Ch 1 Control/Status	P, L
228h	228h	C1COUNT	Ch 1 Transfer Count, Valid Bit	P, L
22Ch	22Ch	C1PCILADR	Ch 1 PCI Address (Lower 32 Bits)	P, L
230h	230h	C1LOCADR	Ch 1 Local Address	P, L
234h	234h	C1DESCPTR	Ch 1 Descriptor Pointer	P, L
238h	238h	C1PCIHADR	Ch 1 PCI Address (Upper 32 Bits)	P, L
23Ch	23Ch	C1THRES	Ch 1 Threshold Register	P, L
240h	240h	C2MODE	Ch 2 Mode	P, L
244h	244h	C2CSR	Ch 2 Control/Status	P, L
248h	248h	C2COUNT	Ch 2 Transfer Byte Count	P, L
24Ch	24Ch	C2SRCADR	Ch 2 Source Address	P, L
250h	250h	C2DESTADR	Ch 2 Destination Address	P, L
254h-27Fh	254h-27Fh	—	***Reserved***	—

17.2.8 Serial EEPROM Loading Sequence

Table 17-9. Serial EEPROM Loading Sequence

Local Offset from CFGBA	Registers	Lword(s)
84h-E4h	Local Configuration	24
100h-168h	Memory Controller	27
180h-184h	Mailboxes	2
300h, 308h, 32Ch, 334h, 33Ch, 340h, 348h, 34Ch, 350h, 354h	PCI Configuration	10
80h	First Local Configuration	1
Total = 64 words = 256 bytes = 2048 bits		

Note: *Lword = 32 bits.*

17.3 PCI CONFIGURATION REGISTERS

Registers may be written to or read from in Byte (8-bit), Word (16-bit) or Lword (32-bit) accesses.

Register 17-1. (PCIVID; PCI:00h, LOC:300h) PCI Vendor ID

Bit	Description	Read	Write	Value after Reset
15:0	**Vendor ID.** Identifies the device manufacturer. Defaults to the PCI SIG-issued Vendor ID of PLX (10B5h) if no serial EEPROM is present.	P, L	L, E	10B5h

Register 17-2. (PCIDID; PCI:02h, LOC:302h) PCI Device ID

Bit	Description	Read	Write	Value after Reset
15:0	**Device ID.** Identifies the particular device. Defaults to the PLX part number for the IOP 480 (0480h) if no serial EEPROM is present.	P, L	L, E	0480h

Register 17-3. (PCICR; PCI:04h, LOC:304h) PCI Command

Bit	Description	Read	Write	Value after Reset
15:10	***Reserved.***	P, L	No	0h
9	**Fast Back-to-Back Enable.** Indicates the type of fast back-to-back transfers a Master can perform on the bus. Value of 0 indicates fast back-to-back transfers can occur only to the same agent as the previous cycle.	P, L	No	0
8	**SERR# Enable.** Value of 0 disables the SERR# driver. Value of 1 enables the SERR# driver.	P, L	P, L	0
7	**Step Enable.** Controls whether the device does address/data stepping. Value of 0 indicates the device never does stepping. ***Note:*** *Hardcoded to 0.*	P, L	No	0
6	**Parity Error Response.** Value of 0 indicates a Parity error is ignored and operation continues. Value of 1 indicates a parity checking is enabled.	P, L	P, L	0
5	**VGA Palette Snoop.** *Not supported.*	P, L	No	0
4	**Memory Write/Invalidate.** Allows the IOP 480 to generate Memory Write and Invalidate cycles on the PCI Bus. (Refer to DMPBAM[7], C0MODE[13], and C1MODE[13].)	P, L	P, L	0
3	**Special Cycle.** *Not supported.*	P, L	No	0
2	**Master Enable.** Value of 0 disables the device from generating Bus Master accesses. Value of 1 allows the device to behave as a Bus Master.	P, L	P, L	0
1	**Memory Space.** Value of 0 disables the device from responding to Memory Space accesses. Value of 1 allows the device to respond to Memory Space accesses.	P, L	P, L	0
0	**I/O Space.** Value of 0 disables the device from responding to I/O Space accesses. Value of 1 allows the device to respond to I/O Space accesses.	P, L	P, L	0

Register 17-4. (PCISR; PCI:06h, LOC:306h) PCI Status

Bit	Description	Read	Write	Value after Reset
15	**Parity Error Detected.** Value of 1 indicates the IOP 480 detected a PCI Bus Parity error, although Parity error handling was disabled (PCICR[6]=0). One of three conditions can cause this bit to be set. 1) The IOP 480 detected a Parity error during a PCI Address phase; 2) The IOP 480 detected a Data Parity error when it was the target of a write; 3) The IOP 480 detected a Data Parity error when performing a Master Read operation. Writing a 1 to this bit clears the bit.	P, L	P, L/Clr	0
14	**Signaled System Error.** Value of 1 indicates the IOP 480 reported a system error on the SERR# signal. Writing a 1 to this bit clears the bit.	P, L	P, L/Clr	0
13	**Received Master Abort.** Value of 1 indicates the IOP 480 received a Master Abort signal. Writing a 1 to this bit clears the bit.	P, L	P, L/Clr	0
12	**Received Target Abort.** Value of 1 indicates the IOP 480 received a Target Abort signal. Writing a 1 to this bit clears the bit.	P, L	P, L/Clr	0
11	**Target Abort.** Value of 1 indicates the IOP 480 signaled a Target abort. Writing a 1 to this bit clears the bit.	P, L	P, L/Clr	0
10:9	**DEVSEL Timing.** Indicates the timing for DEVSEL# assertion. Value of 01 is medium.	P, L	No	01
8	**Master Data Parity Error Detected.** Value of 1 indicates a Data Parity error occurred while the IOP 480 was Bus Master. This bit is set when three conditions are met: 1) The IOP 480 asserted PERR# or observed PERR# asserted; 2) The IOP 480 was the Bus Master for the operation in which the error occurred; 3) The Parity Error Response bit in the Command register is set (PCICR[6]=1). Writing a 1 to this bit clears the bit.	P, L	P, L/Clr	0
7	**Fast Back-to-Back Capable.** Value of 1 enables the IOP 480 to accept fast back-to-back transactions.	P, L	No	1
6	**User-Definable Features.** Value of 0 indicates the IOP 480 does not support User-Definable Features. Value of 1 indicates the IOP 480 does support User-Definable Features. Can only be written from the Local Bus and is read-only from the PCI Bus.	P, L	Local	0
5	***Reserved.***	P, L	No	0
4	**Capabilities List Status.** Value of 0 indicates the IOP 480 does not support Capabilities List Features. Value of 1 indicates the IOP 480 does support Capabilities List Features.	P, L	Local	1
3:0	***Reserved.***	P, L	No	0h

Register 17-5. (PCIREV; PCI:08h, LOC:308h) PCI Revision ID

Bit	Description	Read	Write	Value after Reset
7:0	**Revision ID.** The IOP 480 silicon revision.	P, L	L, E	Current Revision

Register 17-6. (PCICCR; PCI:09h-0Bh, LOC:309h-30Bh) PCI Class Code

Bit	Description	Read	Write	Value after Reset
23:16	**Base Class Code.** Values: 06h = Bridge Device 0Eh = Intelligent I/O Controller	P, L	L, E	0Eh
15:8	**Subclass Code.** Values: 00h = I_2O Device 80h = Other Bridge Device	P, L	L, E	00h
7:0	**Register Level Programming Interface.** Value: 00h = Messaging Queue Ports at 40h, 44h.	P, L	L, E	00h

Register 17-7. (PCICLSR; PCI:0Ch, LOC:30Ch) PCI Cache Line Size

Bit	Description	Read	Write	Value after Reset
7:0	**Cache Line Size.** System cache line size, in units of 32-bit Lwords.	P, L	P, L	0h

Register 17-8. (PCILTR; PCI:0Dh, LOC:30Dh) PCI Latency Timer

Bit	Description	Read	Write	Value after Reset
7:0	**PCI Latency Timer.** Specifies, in units of PCI Bus clocks, the amount of time the IOP 480, as a Bus Master, can burst data on the PCI Bus.	P, L	P, L	0h

Register 17-9. (PCIHTR; PCI:0Eh, LOC:30Eh) PCI Header Type

Bit	Description	Read	Write	Value after Reset
7	**Header Type.** Value of 0 indicates a single function. Value of 1 indicates multiple functions.	P, L	L	0
6:0	**Configuration Layout Type.** Specifies the layout of bytes 10h through 3Fh in Configuration space. Only layout 0 is supported. All other encodings are ***reserved***.	P, L	L	0h

Register 17-10. (PCIBISTR; PCI:0Fh, LOC:30Fh) PCI Built-In Self Test (BIST)

Bit	Description	Read	Write	Value after Reset
7	**BIST Support.** Returns 1 if the device supports BIST. Returns 0 if the device is not BIST-compatible.	P, L	L	0
6	**BIST Start.** The PCI Bus writes a 1 to invoke BIST. Generates an interrupt to the Local Bus. Local Bus resets the bit when BIST is complete. Software fails the device if BIST is not complete after two seconds. Refer to the Runtime registers for interrupt control/status.	P, L	P, L	0
5:4	***Reserved.***	P, L	No	00
3:0	**BIST Code.** Value of 0 indicates the device passed its test. Nonzero values indicate the device failed. Device specific failure codes can be encoded in the non-zero value.	P, L	L	0h

Register 17-11. (PCIBAR0; PCI:10h, LOC:310h) PCI Base Address Register for Memory Accesses to Configuration Registers and Local Address Space 0

Bit	Description	Read	Write	Value after Reset
31:10	**Memory Base Address.** Memory Base address for access to Messaging Queue, Local, Memory Controller, Runtime and DMA registers, and to Local Address Space 0. The first 1 KB of this Base Address register accesses the Memory-Mapped Configuration registers. Addresses above 1 KB access Local Address Space 0.	P, L	P, L	0h
9:4	**Memory Base Address.** Memory Base address for access to Messaging Queue, Local, Memory Controller, Runtime and DMA registers (default is 1 KB). ***Note:*** *Hardcoded to 0.*	P, L	No	0h
3	**Prefetchable.** Value of 1 indicates there are no side effects on reads. This bit has no effect on the IOP 480 operation. ***Note:*** *Hardcoded to 0.*	P, L	No	0
2:1	**Location of Register.** Location values: 00 = Locate anywhere in 32-bit Memory Address space 11 = ***Reserved*** ***Note:*** *Hardcoded to 0.*	P, L	No	00
0	**Memory Space Indicator.** Value of 0 indicates the register maps into Memory space. Value of 1 indicates the register maps into I/O space. ***Note:*** *Hardcoded to 0.*	P, L	No	0

Register 17-12. (PCIBAR1; PCI:14h, LOC:314h) PCI Base Address Register for Memory Accesses to Local Address Space 1

Bit	Description	Read	Write	Value after Reset
31:4	**Memory Base Address.** Memory Base address for access to Local Address Space 1.	P, L	P, L	0h
3	**Prefetchable (If Memory Space)**. Value of 1 indicates there are no side effects on reads. Reflects value of LAS1RR[3] and provides only status to the system. This bit has no effect on the IOP 480 operation. Prefetching features of this Address space are controlled by the associated Bus Region Descriptor register. (Specified in LAS1RR register.) If I/O space, bit 3 is included in the Base address.	P, L	Mem: No I/O: P, L	0
2:1	**Location of Register (If Memory Space).** Location values: 00 = Locate anywhere in 32-bit Memory Address space 01 = Locate below 1-MB Memory Address space 10 = Locate anywhere in 64-bit Memory Address space 11 = ***Reserved*** (Specified in LAS1RR register.) If I/O Space, bit 1 is always 0 and bit 2 is included in the Base address.	P, L	Mem: No I/O: bit 1, no bit 2, yes P, L	00
0	**Memory Space Indicator.** Value of 0 indicates the register maps into Memory space. Value of 1 indicates the register maps into I/O space. (Specified in LAS1RR register.)	P, L	No	0

Note: *PCIBAR1 can be enabled or disabled by setting or clearing LAS1BA[0].*

Register 17-13. (PCIBAR2; PCI:18h, LOC:318h) PCI Base Address Register for Memory Accesses to Local Address Space 2

Bit	Description	Read	Write	Value after Reset
31:4	**Memory Base Address.** Memory Base address for access to Local Address Space 2.	P, L	P, L	0h
3	**Prefetchable (If Memory Space).** Value of 1 indicates there are no side effects on reads. Reflects the value of LAS2RR[3] and provides only status to the system. This bit has no effect on the IOP 480 operation. Prefetching features of this Address space are controlled by the associated Bus Region Descriptor register. (Specified in LAS2RR register.) If I/O space, bit 3 is included in the Base address.	P, L	Mem: No I/O: P, L	0
2:1	**Location of Register.** Location values: 00 = Locate anywhere in 32-bit Memory Address space 01 = Locate below 1-MB Memory Address space 10 = Locate anywhere in 64-bit Memory Address space 11 = ***Reserved*** (Specified in LAS2RR register.) If I/O Space, bit 1 is always 0 and bit 2 is included in the Base address.	P, L	Mem: No I/O: bit 1, no bit 2, yes P, L	00
0	**Memory Space Indicator.** Value of 0 indicates the register maps into Memory space. Value of 1 indicates the register maps into I/O space. (Specified in LAS2RR register.)	P, L	No	0

Note: *PCIBAR2 can be enabled or disabled by setting or clearing LAS2BA[0].*

Register 17-14. (PCIBAR3; PCI:1Ch, LOC:31Ch) PCI Base Address Register 3

Bit	Description	Read	Write	Value after Reset
31:0	***Reserved.***	P, L	No	0h

Register 17-15. (PCIBAR4; PCI:20h, LOC:320h) PCI Base Address Register 4

Bit	Description	Read	Write	Value after Reset
31:0	***Reserved.***	P, L	No	0h

Register 17-16. (PCIBAR5; PCI:24h, LOC:324h) PCI Base Address Register 5

Bit	Description	Read	Write	Value after Reset
31:0	***Reserved.***	P, L	No	0h

Register 17-17. (PCICIS; PCI:28h, LOC:328h) PCI Cardbus CIS Pointer

Bit	Description	Read	Write	Value after Reset
31:0	**Cardbus.** Cardbus Information Structure Pointer (CIS) for PCMCIA. *Not supported.*	P, L	No	0h

Register 17-18. (PCISVID; PCI:2Ch, LOC:32Ch) PCI Subsystem Vendor ID

Bit	Description	Read	Write	Value after Reset
15:0	**Subsystem Vendor ID (Unique Add-In Board Vendor ID).**	P, L	L, E	10B5h

Register 17-19. (PCISID; PCI:2Eh, LOC:32Eh) PCI Subsystem ID

Bit	Description	Read	Write	Value after Reset
15:0	**Subsystem ID (Unique Add-In Board Device ID).**	P, L	L, E	0480h

Register 17-20. (PCIERBAR; PCI:30h, LOC:330h) PCI Expansion ROM Base

Bit	Description	Read	Write	Value after Reset
31:11	**Expansion ROM Base Address (Upper 21 Bits).**	P, L	P, L	0h
10:1	***Reserved.***	P, L	No	0h
0	**Address Decode Enable.** Value of 0 indicates the device does not accept accesses to the expansion ROM space. Value of 1 indicates the device accepts accesses to the expansion ROM address. Set this bit to 0 if no Expansion ROM is present.	P, L	P, L	0

Register 17-21. (CAP_PTR; PCI:34h, LOC:334h) Capability List Pointer

Bit	Description	Read	Write	Value after Reset
31:8	***Reserved.***	P, L	No	0h
7:0	**Capability List Pointer.** Provides an offset into the PCI Configuration space for the location of the first item in the Capabilities List, if the Capabilities List Status bit is set (PCISR[4]=1). Bits [1:0] should be written as 00.	P, L	L, E	40h

Register 17-22. (PCIILR; PCI:3Ch, LOC:33Ch) PCI Interrupt Line

Bit	Description	Read	Write	Value after Reset
7:0	**Interrupt Line Routing Value.** Value indicates which System Interrupt (controller(s) input is connected to the Device interrupt.	P, L	P, L	0h

Register 17-23. (PCIIPR; PCI:3Dh, LOC:33Dh) PCI Interrupt Pin

Bit	Description	Read	Write	Value after Reset
7:0	**Interrupt Pin Register.** Indicates which interrupt pin the device uses. The following values are decoded: 0 = No interrupt pin 1 = INTA# 2 = INTB# 3 = INTC# 4 = INTD#	P, L	L, E	01h

Register 17-24. (PCIMGR; PCI:3Eh, LOC:33Eh) PCI Min_Gnt

Bit	Description	Read	Write	Value after Reset
7:0	**Min_Gnt.** Specifies how long a burst period the device needs, assuming a clock rate of 33 MHz. Value is multiple of 1/4 µs increments.	P, L	L, E	0h

Register 17-25. (PCIMLR; PCI:3Fh, LOC:33Fh) PCI Max_Lat

Bit	Description	Read	Write	Value after Reset
7:0	**Max_Lat.** Specifies how often the device must gain access to the PCI Bus. Value is multiple of 1/4 µs increments.	P, L	L, E	0h

Register 17-26. (PMCAPID; PCI:40h, LOC:340h) Power Management Capability ID

Bit	Description	Read	Write	Value after Reset
7:0	**PM ID.** Specifies the Power Management Capability ID.	P, L	L, E	01h

Register 17-27. (PMNEXT; PCI:41h, LOC:341h) Power Management Next Capability Pointer

Bit	Description	Read	Write	Value after Reset
7:0	**Next_Cap Pointer.** Points to the first location of the next item in the Capabilities Linked list. If Power Management is the last item in the list, then this register should be set to 0.	P, L	L, E	54h

Register 17-28. (PMC; PCI:42h, LOC:342h) Power Management Capabilities

Bit	Description	Read	Write	Value after Reset
15	***Reserved.***	P, L	No	0
14:11	**PME_Support.** Indicates the power states in which the IOP 480 may assert PME#. Value Description XXX1 PME# can be asserted from D_0 XX1X PME# can be asserted from D_1 X1XX PME# can be asserted from D_2 1XXX PME# can be asserted from D_{3hot}	P, L	L, E	0h
10	**D_2_Support.** Value of 1 indicates the IOP 480 supports the D_2 power state.	P, L	L, E	0
9	**D_1_Support.** Value of 1 indicates the IOP 480 supports the D_1 power state.	P, L	L, E	0
8:6	***Reserved.***	P, L	No	000
5	**DSI.** Value of 1 indicates the IOP 480 requires special initialization following a transition to the D_0 uninitialized state before the generic class device driver is able to use it.	P, L	L, E	0
4	**Auxiliary Power Source.** Because the IOP 480 does not support PME# while in D_{3cold}, this bit is always set to 0.	P, L	No	0
3	**PME Clock.** Value of 1 indicates that a function relies on the presence of the PCI clock for PME# operation. The IOP 480 does not require the PCI clock for PME#; therefore, this bit should set to 0.	P, L	L, E	0
2:0	**Version.** Value of 1 indicates that this function complies with *PCI Power Management Interface Specification, r1.1*.	P, L	L, E	01

Register 17-29. (PMCSR; PCI:44h, LOC:344h) Power Management Control/Status

Bit	Description	Read	Write	Value after Reset
15	**PME_Status.** Value of 1 indicates the PME# pin is being driven if PME_En is set high. Writing a 1 from the Local Bus causes this bit to be set, and writing a 1 from the PCI Bus clears it. Depending on the current power state, this bit is set only if the appropriate PME_Support bits are set (PMC[14:11]).	P, L	L/Set, P/Clr	0
14:13	**Data_Scale.** Indicates the scaling factor to use when interpreting the Data Register value. Value and meaning of this bit depends on the data value selected by the Data_Select bits. The Data_Scale register bits accessed during a read are selected by the value in the Data_Select bits. For Power Consumed and Power Dissipated data, the following scale factors are used (unit values are in watts; data is read from PMSCALE): Value Scale 00 Unknown 01 0.1x 10 0.01x 11 0.001x	P, L	No	00
12:9	**Data_Select.** Selects which data to access through the PMDATA register and Data_Scale bits.	P, L	P, L	0h
8	**PME_En.** Value of 1 enables the PME# pin to be asserted.	P, L	P, L	0
7:2	***Reserved.***	P, L	No	0h
1:0	**Power State.** Used to determine or change the current power state. Value State 00 D_0 01 D_1 10 D_2 11 D_{3hot} The transition from state 11 to state 00 causes a soft reset to occur. This should only be initiated from the PCI Bus because the Local Bus interface is reset during a soft reset. In state D_{3hot}, PCI Memory and I/O accesses are disabled, as well as the PCI interrupts, and only configuration is allowed. The same is true for state D_2 if the corresponding D_2_Support pin is set.	P, L	P, L	00

Register 17-30. (PMCSR_BSE; PCI:46h, LOC:346h) PMCSR Bridge Support Extensions

Bit	Description	Read	Write	Value after Reset
7:0	***Reserved.***	P, L	No	0h

Register 17-31. (PMDATA; PCI:47h, LOC:347h) Power Management Data

Bit	Description	Read	Write	Value after Reset
7:0	**Power Management Data.** Provides operating data such as power consumed or heat dissipation. Data returned is selected by the Data_Select bits and is scaled by the Data_Scale bits. Data_Select Value — Description 0 — D_0 Power Consumed 1 — D_1 Power Consumed 2 — D_2 Power Consumed 3 — D_3 Power Consumed 4 — D_0 Power Dissipated 5 — D_1 Power Dissipated 6 — D_2 Power Dissipated 7 — D_3 Power Dissipated	P, L	No	0h

Register 17-32. (PMSCALE; PCI:48h, LOC:348h) Power Management Data_Scale Values

Bit	Description	Read	Write	Value after Reset
31:16	***Reserved.***	P, L	No	0h
15:14	**Data_Scale 7.** Provides the data scale value for Power Management Data 7 (Data_Select = 7).	P, L	L, E	00
13:12	**Data_Scale 6.** Provides the data scale value for Power Management Data 6 (Data_Select = 6).	P, L	L, E	00
11:10	**Data_Scale 5.** Provides the data scale value for Power Management Data 5 (Data_Select = 5).	P, L	L, E	00
9:8	**Data_Scale 4.** Provides the data scale value for Power Management Data 4 (Data_Select = 4).	P, L	L, E	00
7:6	**Data_Scale 3.** Provides the data scale value for Power Management Data 3 (Data_Select = 3).	P, L	L, E	00
5:4	**Data_Scale 2.** Provides the data scale value for Power Management Data 2 (Data_Select = 2).	P, L	L, E	00
3:2	**Data_Scale 1.** Provides the data scale value for Power Management Data 1 (Data_Select = 1).	P, L	L, E	00
1:0	**Data_Scale 0.** Provides the data scale value for Power Management Data 0 (Data_Select = 0).	P, L	L, E	00

Register 17-33. (PWRCON; PCI:4Ch, LOC:34Ch) Power Consumed Values

Bit	Description	Read	Write	Value after Reset
31:24	**D_3 Power Consumed.** Provides the power consumed in the D_3 state. (Value read from PMDATA when Data_Select = 3).	P, L	L, E	0h
23:16	**D_2 Power Consumed.** Provides the power consumed in the D_2 state. (Value read from PMDATA when Data_Select = 2).	P, L	L, E	0h
15:8	**D_1 Power Consumed.** Provides the power consumed in the D_1 state. (Value read from PMDATA when Data_Select = 1).	P, L	L, E	0h
7:0	**D_0 Power Consumed.** Provides the power consumed in the D_0 state. (Value read from PMDATA when Data_Select = 0).	P, L	L, E	0h

Register 17-34. (PWRDIS; PCI:50h, LOC:350h) Power Dissipated Values

Bit	Description	Read	Write	Value after Reset
31:24	**D_3 Power Dissipated.** Provides the power dissipated in the D_3 state. (Value read from PMDATA when Data_Select = 7.)	P, L	L, E	0h
23:16	**D_2 Power Dissipated.** Provides the power dissipated in the D_2 state. (Value read from PMDATA when Data_Select = 6.)	P, L	L, E	0h
15:8	**D_1 Power Dissipated.** Provides the power dissipated in the D_1 state. (Value read from PMDATA when Data_Select = 5.)	P, L	L, E	0h
7:0	**D_0 Power Dissipated.** Provides the power dissipated in the D_0 state. (Value read from PMDATA when Data_Select = 4.)	P, L	L, E	0h

Register 17-35. (HSCAPID; PCI:54h, LOC:354h) Hot Swap Capability ID

Bit	Description	Read	Write	Value after Reset
7:0	**Hot Swap ID.** Specifies the Hot Swap Capability ID. Must be set to 06h to be valid.	P, L	L, E	0h

Register 17-36. (HSNEXT; PCI:55h, LOC:355h) Hot Swap Next Capability Pointer

Bit	Description	Read	Write	Value after Reset
7:0	**Next_Cap Pointer.** Points to the first location of the next item in the capabilities linked list. If Hot Swap is the last item in the list, then this register should be set to 0.	P, L	L, E	58h

Register 17-37. (HSCSR; PCI:56h, LOC:356h) Hot Swap Control/Status

Bit	Description	Read	Write	Value after Reset
15:8	***Reserved.***	P, L	No	0h
7	**ENUM# Insertion Status.** Value of 1 indicates that a board was inserted. Writing a 1 clears this status bit.	P, L	P, L/Clr	0
6	**ENUM# Removed Status.** Value of 1 indicates that a board was removed. Writing a 1 clears this status bit.	P, L	P, L/Clr	0
5:4	***Reserved.***	P, L	No	00
3	**LED State.** Value of 1 indicates the external LED is turned on. Value of 0 indicates the external LED is turned off. Reading this bit returns the LEDon/LEDin pin status.	P, L	P, L	0
2	***Reserved.***	P, L	No	0
1	**ENUM# Interrupt Mask.** Value of 0 indicates the ENUM# interrupt output is enabled. Value of 1 indicates the ENUM# interrupt output is disabled.	P, L	P, L	0
0	***Reserved.***	P, L	No	0

Register 17-38. (VPD_CAP; PCI:58h, LOC:358h) VPD Capabilities

Bit	Description	Read	Write	Value after Reset
31	**F Flag.** A flag used to indicate when a data operation to the serial EEPROM is complete. For Write cycles, the four bytes of data are first written into the VPD Data bits. Then, the VPD Address is written at the same time the F Flag is set to 1. When the Data transfer to the serial EEPROM is complete, the F Flag is cleared. For Read cycles, the VPD Address is written at the same time the F Flag is cleared to 0. When the four bytes of data are read from the serial EEPROM, the F Flag is set.	P, L	P, L/Set/Clr	0
30:16	**VPD Address.** VPD Byte Address accessed. All accesses are 32-bit wide; bits [17:16] must be set to 0. With the maximum serial EEPROM size being 4K bits, bits [30:28] are ignored.	P, L	P, L	0h
15:8	**Next_Cap Pointer.** Points to the first location of the next item in the capabilities linked list. If VPD is the last item in the list, then this register should be set to zero.	P, L	L	0h
7:0	**VPD ID.** Specifies the VPD Capability ID.	P, L	L	3h

Register 17-39. (VPD_DATA; PCI:5Ch, LOC:35Ch) VPD Data

Bit	Description	Read	Write	Value after Reset
31:16	**VPD Data.** VPD data is written or read through this register. These bits go into address N, as specified in the VPD Address bits (VPD_CAP[30:16]).	P, L	P, L	0h
15:0	**VPD Data.** VPD data is written or read through this register. These bits go into address N + 1.	P, L	P, L	0h

17.4 MESSAGING QUEUE REGISTERS

Note: *Burst cycles are not supported by the Messaging Queue registers.*

Register 17-40. (MQCR; PCI:00h, LOC:00h) Messaging Queue Configuration

Bit	Description	Read	Write	Value after Reset
31:4	***Reserved.***	P, L	No	0h
3:1	**Queue Size.** Contains the size of one of the circular queues. Each of the four queues are the same size. Queue Size Encoding: 3:1 — Maximum Entries Per Queue — Queue Size — Total Queue Memory 000 — 128 entries — 512 KB — 2 KB 001 — 512 entries — 2 KB — 8 KB 010 — 2 kilobit entries — 8 KB — 32 KB 011 — 4 kilobit entries — 16 KB — 64 KB 100 — 8 kilobit entries — 32 KB — 128 KB 101 — 16 kilobit entries — 64 KB — 256 KB 110 — 32 kilobit entries — 128 KB — 512 KB 111 — 64 kilobit entries — 256 KB — 1 MB	P, L	P, L	000
0	**Queue Enable.** Value of 0 indicates writes are accepted, but ignored, and reads return FFFFFFFFh. Value of 1 allows accesses to the Inbound and Outbound Queue ports. Complete pointer initialization and frame allocation before enabling this bit.	P, L	P, L	0

Register 17-41. (QBAR; PCI:04h, LOC:04h) Queue Base Address

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	P, L	0h
19:0	***Reserved.***	P, L	No	0h

Register 17-42. (IFHPR; PCI:08h, LOC:08h) Inbound Free Head Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Inbound Free Head Pointer.** Local Memory Offset for Inbound Free Queue. Must be initialized to (0*QueueSize) and maintained by software.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-43. (IFTPR; PCI:0Ch, LOC:0Ch) Inbound Free Tail Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Inbound Free Tail Pointer.** Local Memory Offset for Inbound Free Queue. Must be initialized to (0*QueueSize) by software. Maintained by the IOP 480 and incremented with respect to the queue size.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-44. (IPHPR; PCI:10h, LOC:10h) Inbound Post Head Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Inbound Post Head Pointer.** Local Memory Offset for the Inbound Post Queue. Must be initialized to (1*QueueSize) by software. Maintained by the IOP 480 and incremented with respect to the queue size.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-45. (IPTPR; PCI:14h, LOC:14h) Inbound Post Tail Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Inbound Post Tail Pointer.** Local Memory Offset for the Inbound Post Queue. Must be initialized to (1*QueueSize) and maintained by software.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-46. (OFHPR; PCI:18h, LOC:18h) Outbound Free Head Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Outbound Free Head Pointer.** Local Memory Offset for Outbound Free Queue. Must be initialized to (3*QueueSize) by software. Maintained by the IOP 480 and incremented with respect to the queue size.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-47. (OFTPR; PCI:1Ch, LOC:1Ch) Outbound Free Tail Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Outbound Free Tail Pointer.** Local Memory Offset for Outbound Free Queue. Must be initialized to (3*QueueSize) and maintained by software.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-48. (OPHPR; PCI:20h, LOC:20h) Outbound Post Head Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Outbound Post Head Pointer.** Local Memory Offset for Outbound Post Queue. Must be initialized to (2*QueueSize) and maintained by software.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-49. (OPTPR; PCI:24h, LOC:24h) Outbound Post Tail Pointer

Bit	Description	Read	Write	Value after Reset
31:20	**Queue Base Address.** Local Memory Base address of the Inbound and Outbound circular queues. Queue Base address must be aligned on a 1 MB boundary.	P, L	No	0h
19:2	**Outbound Post Tail Pointer.** Local Memory Offset for Outbound Post Queue. Must be initialized to (2*QueueSize) by software. Maintained by the IOP 480 and incremented with respect to the queue size.	P, L	P, L	0h
1:0	***Reserved.***	P, L	No	00

Register 17-50. (QSR; PCI:28h, LOC:28h) Queue Status/Control

Bit	Description	Read	Write	Value after Reset
31:9	***Reserved.***	P, L	No	0h
8	**Outbound Option Interrupt Mask.** Value of 0 enables the Outbound Option to set the Outbound Post Queue Interrupt status bit. Value of 1 masks the Outbound Option interrupt.	P, L	P, L	1
7	**Outbound Free Queue Overflow Interrupt.** Value of 1 indicates the Outbound Free Queue overflowed. A Local interrupt out (INTO) is generated. Writing a Message Frame Address 1 clears the interrupt.	P, L	P, L/Clr	0
6	**Outbound Free Queue Overflow Interrupt Mask**. Value of 0 indicates the Outbound Free Queue Overflow interrupt is enabled. Value of 1 masks the Outbound Free Queue Overflow interrupt.	P, L	P, L	1
5	**Inbound Post Queue Interrupt.** Value of 1 indicates the Inbound Post Queue is not empty. This bit is not effected by the Interrupt Mask bit.	P, L	No	0
4	**Inbound Post Queue Interrupt Mask.** Value of 0 indicates the Inbound Post Queue Interrupt is enabled. Value of 1 masks the Inbound Post Queue Interrupt.	P, L	P, L	1
3	**Inbound Free Queue Prefetch Enable.** Value of 1 indicates prefetching occurs from the Inbound Free Queue if not empty.	P, L	P, L	0
2	**Outbound Post Queue Prefetch Enable.** Value of 1 indicates prefetching occurs from the Outbound Post Queue if not empty.	P, L	P, L	0
1:0	***Reserved.***	P, L	No	00

Register 17-51. (OPQIS; PCI:30h, LOC:30h) Outbound Post Queue Interrupt Status

Bit	Description	Read	Write	Value after Reset
31:4	***Reserved.***	P, L	No	0h
3	**Outbound Post Queue Interrupt Status.** Value of 1 indicates the Outbound Post Queue is not empty. Also, if the Outbound Option interrupt is enabled, this interrupt is set if the HOSTOUTIDX and IOPOUTIDX registers are not equal. This bit is not effected by the Interrupt Mask bit, and causes a PCI interrupt to be generated.	P, L	No	0
2:0	***Reserved.***	P, L	No	000

Register 17-52. (OPQIM; PCI:34h, LOC:34h) Outbound Post Queue Interrupt Mask

Bit	Description	Read	Write	Value after Reset
31:4	***Reserved.***	P, L	No	0h
3	**Outbound Post Queue Interrupt Mask.** Value of 0 indicates the Outbound Post Queue Interrupt is enabled. Value of 1 masks the Outbound Post Queue Interrupt.	P, L	P, L	1
2:0	***Reserved.***	P, L	No	000

Register 17-53. (IQP; PCI:40h) Inbound Queue Port

Bit	Description	Read	Write	Value after Reset
31:0	**Inbound Queue Port.** Value written by the PCI Master is stored into the Inbound Post Queue, which is located in a Local memory at the address pointed to by the Queue Base Address + Queue Size + Inbound Post Head Pointer. From the time of the PCI write until the Local Memory write and update of the Inbound Post Queue Head Pointer, further accesses to this register result in a Retry. A Local interrupt is generated when the Inbound Post Queue is not empty. When the port is read by the PCI Master, the value is read from the Inbound Free Queue, which is located in Local memory at the address pointed to by the Queue Base Address + Inbound Free Tail Pointer. If the queue is empty, a value of FFFFFFFh is returned.	P	P	0h

Register 17-54. (OQP; PCI:44h) Outbound Queue Port

Bit	Description	Read	Write	Value after Reset
31:0	**Outbound Queue Port.** Value written by the PCI Master is stored into the Outbound Free Queue, which is located in a Local memory at the address pointed to by the Queue Base Address + 3*Queue Size + Outbound Free Head Pointer. From the time of the PCI write until the Local Memory write and update of the Outbound Free Queue Head Pointer, further accesses to this register result in a Retry. If the queue fills, a Local NMI interrupt is generated. When the port is read by the PCI Master, the value is read from the Outbound Post Queue, which is located in Local memory at the address pointed to by the Queue Base Address + 2*Queue Size + Outbound Post Tail Pointer. If the queue is empty, a value of FFFFFFFh is returned. A PCI interrupt is generated if the Outbound Post Queue is not empty.	P	P	0h

Register 17-55. (HOSTOUTIDX; PCI:50h, LOC:50h) Host Outbound Index

Bit	Description	Read	Write	Value after Reset
31:2	**Host Outbound Index Register.** Written by the PCI Host. Used for implementing the I_2O Outbound Option. Indicates to the IOP 480 how many of the posted I_2O messages were processed by the Host.	P, L	P, L	0h
1:0	***Reserved*.**	P, L	No	00

Register 17-56. (IOPOUTIDX; PCI:54h, LOC:54h) IOP Outbound Index

Bit	Description	Read	Write	Value after Reset
31:2	**IOP Outbound Index Register.** Maintained by the Local CPU. Used for implementing the I_2O Outbound Option. The Local CPU (IOP) uses this register to keep track of where the next MFA is written into the HostPostList FIFO in Host memory.	P, L	P, L	0h
1:0	***Reserved*.**	P, L	No	00

17.5 LOCAL CONFIGURATION REGISTERS

Register 17-57. (DEVINIT; PCI:80h, LOC:80h) Device Initialization

Bit	Description	Read	Write	Value after Reset
31	**Local Init Status.** Value of 1 indicates Local Init done. Responses to Direct Slave accesses are Retrys until this bit is set.Must be set by either the serial EEPROM (at the end of loading) or by the internal IOP 480 CPU (in Adapter mode).	P, L	P, L, E	0
30	**Software Reset.** Value of 1 activates Software reset. The IOP 480 Host/Adapter Mode pin determines the behavior of the Software Reset.	P, L	P, L, E	0
29	**Reload Configuration Registers.** Changing the value from 0 to 1 causes IOP 480 to reload the Configuration registers from the serial EEPROM.	P, L	P, L, E	0
28	**Serial EEPROM Present.** Value of 1 indicates a non-blank serial EEPROM is present and valid. Works in conjunction with DEVINIT[5].	P, L	No	0
27	**Read Serial EEPROM Data.** For reads, this input bit is the output of the serial EEPROM. Clocked out of the serial EEPROM by the serial EEPROM clock.	P, L	No	—
26	**Write Bit to Serial EEPROM.** For writes, this output bit is the input to the serial EEPROM. Clocked into the serial EEPROM by the serial EEPROM clock.	P, L	P, L, E	0
25	**Serial EEPROM Chip Select.** For Local or PCI Bus reads or writes to the serial EEPROM, setting this bit to 1 provides the serial EEPROM chip select.	P, L	P, L, E	0
24	**Serial EEPROM Clock.** Toggling this bit generates a serial EEPROM clock. (Refer to manufacturer's data sheet for particular serial EEPROM being used.)	P, L	P, L, E	0
23	**Serial EEPROM Data Output Enable.** Value of 1 enables software access (Read/Write) to the serial EEPROM. Value of 0 disables software access (Read/Write) to the serial EEPROM.	P, L	P, L	0
22:15	***Reserved.***	P, L	No	0h
14:8	**Serial EEPROM Protected Area.** Specifies the top of the protected area in the serial EEPROM. Locations below this value cannot be written to. These bits are in units of 32-bit Lwords.	P, L	P, L	40h
7:6	***Reserved*.**	P, L	No	00
5	**Serial EEPROM Physically Present.** Value of 1 indicates the serial EEPROM is blank or programmed. *When the Serial EEPROM Present bit is set to 1 (DEVINIT[28]=1), DEVINIT[5] indicates a non-blank serial EEPROM is present.* Value of 0 indicates that there is no serial EEPROM physically present.	P, L	No	0
4	**RISCTrace Enable.** Value of 1 enables diagnostic RISCTrace mode. Bit 23, the Serial EEPROM Data Output Enable bit, must also be asserted. The UART (SPU) and serial EEPROM interface are disabled when RISCTrace is enabled. The third PCI Arbiter Channel (REQ2# and GNT2#) are disabled while RISCTrace is running. In RISCTrace mode, the IOP 480 makes use of the TS[6:1] output pins.	P, L	P, L, E	0
3	**ID Select.** Value of 1 places the Subsystem and Subsystem Vendor IDs into the PCIVID and PCIDID registers.	P, L	P, L, E	0
2:1	**Mask Revision.** Indicates the silicon mask revision.	P, L	No	Current Mask Revision
0	**IOP 480 CPU Reset.** Value of 0 causes the IOP 480 CPU reset to de-assert, allowing it to begin operation. Value of 1 holds the IOP 480 CPU in a reset state. If a programmed serial EEPROM is not detected at PCI reset, this bit is set to 0, allowing the CPU to start. If a programmed serial EEPROM is detected, then this bit is programmed by the serial EEPROM.	P, L	P, L, E	1

Register 17-58. (LOCCTL; PCI:84h, LOC:84h) Local Bus Control

Bit	Description	Read	Write	Value after Reset
31:26	***Reserved.***	P, L	No	0h
25	**PCI Abort Control.** Value of 1 indicates a PCI Master/Target abort asserts BTERM# as it normally does, and also continues asserting READY# until BLAST# is asserted. Value of 0 sets up BTERM# to indicate the last data transfer for the access. This means that an external Local Master must know that a cycle can end with either BLAST# (by Master) or BTERM# (by Slave).	P, L	P, L	1
24	**IOP 480 CPU Lock.** Value of 1 indicates the Local Bus arbiter to grant only the Local Bus to the internal IOP 480 CPU. Value of 0 indicates the Local Bus arbiter operates normally.	P, L	P, L, E	0
23:22	**Direct Slave Write Delay.** Used to delay writes to the Local Bus, providing time for data to accumulate in the Direct Slave Write FIFO before starting a Local Burst cycle. 00 = No delay; start the cycle immediately 01 = Delay 4 Local clocks 10 = Delay 8 Local clocks 11 = Delay 16 Local clocks	P, L	P, L, E	00
21	**Direct Master FIFO Almost Full.** Value of 1 indicates the Direct Master Write FIFO is almost full. Reflects the inverse of the DMPAF# pin.	P, L	No	0
20	***Reserved*.**	P, L	No	0
19	**BLAST# Timing.** Value of 0 asserts BLAST# during the entire last cycle when the IOP 480 is the Local Bus Master. Value of 1 causes BLAST# to not assert until the internal wait state counters have finished counting.	P, L	P, L, E	0
18	***Reserved.***	P, L	No	0
17	**USER4 Data.** Reading this bit provides the state of the EOT1#/EOT2#/ USER4 pin. When the EOT1#/EOT2#/USER4 pin is not used for DMA, it can be used as a general purpose input pin.	P, L	No	1
16	**USER3 Data.** Reading this bit provides the state of the EOT0#/USER3 pin. When the EOT0#/USER3 pin is not used for DMA, it can be used as a general purpose input pin.	P, L	No	1
15:12	***Reserved.***	P, L	No	0h
11	**DREQ/DACK/EOT Select.** Selects the function of the DREQ1#/DREQ2#, DACK1#/DACK2# and EOT1#/EOT2#/USER4 pins for DMA Channels 1 and 2. Value of 0 indicates use of the DREQ1#, DACK1#, and EOT1# pins. Value of 1 indicates use of the DREQ2#, DACK2#, and EOT2# pins.	P, L	P, L, E	0
10	**USER2 Data.** If programmed as an output, writing a 1 causes the corresponding pin to go high. If programmed as an input, reading this bit provides the state of the CINT/USER2 pin.	P, L	P, L, E	0
9	**USER2 Direction.** Value of 0 indicates the bit is programmed as an input. Value of 1 indicates the bit is programmed as an output. It is always an input if the CINT function is selected.	P, L	P, L, E	0
8	**CINT/USER2 Pin Select.** Selects function of CINT/USER2 pin. Value of 0 identifies this pin as CINT. Value of 1 identifies this pin as USER2.	P, L	P, L, E	1
7	**LCS3#/MA17 Pin Select.** Selects function of LCS3#/MA17 pin. Value of 0 identifies this pin as LCS3#. Value of 1 identifies this pin as MA17.	P, L	P, L, E	0
6	**USER1 Data.** If programmed as an output, writing a 1 causes the corresponding pin to go high. If programmed as an input, reading this bit provides the state of the LCS2#/USER1 pin.	P, L	P, L, E	0

Register 17-58. (LOCCTL; PCI:84h, LOC:84h) Local Bus Control (Continued)

Bit	Description	Read	Write	Value after Reset
5	**USER1 Direction.** Value of 0 indicates the bit is programmed as an input. Value of 1 indicates the bit is programmed as an output. It is always an output if the LCS2# function is selected.	P, L	P, L, E	0
4	**LCS2#/USER1 Pin Select.** Selects function of the LCS2#/USER1 pin. Value of 0 identifies this pin as LCS2#. Value of 1 identifies this pin as USER1.	P, L	P, L, E	1
3	**USER0 Data.** If programmed as an output, writing a 1 causes the corresponding pin to go high. If programmed as an input, reading this bit provides the state of the LCS1#/USER0 pin.	P, L	P, L, E	0
2	**USER0 Direction.** Value of 0 indicates the bit is programmed as an input. Value of 1 indicates the bit is programmed as an output. It is always an input if the LCS1# function is selected.	P, L	P, L, E	0
1	**LCS1#/USER0 Pin Select.** Selects function of LCS1#/USER0 pin. Value of 0 identifies this pin as LCS1#. Value of 1 identifies this pin as USER0.	P, L	P, L, E	1
0	**LCS0#/DMPAF# Pin Select.** Selects function of the LCS0#/DMPAF# pin. Value of 0 identifies this pin as LCS0#. Value of 1 identifies this pin as DMPAF#.	P, L	P, L, E	0

Register 17-59. (LOCTMO; PCI:88h, LOC:88h) Local Bus Timeout

Bit	Description	Read	Write	Value after Reset
31:16	***Reserved.***	P, L	No	0h
15	**Local Bus Timeout Enable.** Value of 1 indicates the Local Bus Timeout Timer is enabled. If an external Local Bus Master controls the Local Bus, then the Timeout Timer runs only if the Timeout Timer Enable bit in the corresponding Memory Controller register is also set.	P, L	P, L, E	0
14:0	**Local Bus Timeout Value.** Value loaded into a timer at the beginning of the Local Bus transfer. Timer is decremented once per Local Bus clock while READY# is de-asserted. If the timer times out, an internal ready and interrupt are generated, signaling a bus timeout. Maximum timeout is about 480 µs, based on a 66 MHz Local Bus clock.	P, L	P, L, E	0h

Register 17-60. (LOCTMR; PCI:8Ch, LOC:8Ch) Local Bus Timers

Bit	Description	Read	Write	Value after Reset
31:25	***Reserved.***	P, L	No	0h
24	**Local Bus Pause Timer Enable.** Value of 1 indicates the Pause Timer is enabled.	P, L	P, L, E	0
23:16	**Local Bus Pause Timer.** Number of Local Bus Clock cycles to occur before requesting the Local Bus after releasing the Local Bus for internal DMA channels.	P, L	P, L, E	0h
15:9	***Reserved.***	P, L	No	0h
8	**Local Bus Latency Timer Enable.** Value of 1 indicates the Latency Timer is enabled.	P, L	P, L, E	0
7:0	**Local Bus Latency Timer.** Number of Local Bus Clock cycles the IOP 480 holds the Local Bus before releasing it to another requester.	P, L	P, L, E	0h

Register 17-61. (LARBR; PCI:90h, LOC:90h) Local/DMA Arbitration

Bit	Description	Read	Write	Value after Reset
31:22	***Reserved.***	P, L	No	0h
21	**BOFF# Timer Resolution.** Value of 0 selects the LSB of the Backoff timer of 8 clocks. Value of 1 selects the LSB of the Backoff timer of 64 clocks.	P, L	P, L, E	0
20:17	Direct Slave **BOFF# Delay Clocks.** Number of Local Bus clocks in which a Direct Slave Bus request is pending and a Local Direct Master access is in progress and not being granted the bus before asserting BOFF#. Once asserted, BOFF# remains asserted until the LHOLDREQ[1:0] pins are de-asserted. (LSB = 8 or 64 clocks.)	P, L	P, L, E	0h
16	**Local Bus BOFF# Enable.** Value of 1 enables the IOP 480 to assert the BOFF# pin.	P, L	P, L, E	0
15:11	***Reserved.***	P, L	No	0h
10	**Latency Timer Gate.** Value of 1 indicates the gate of the Local Bus Latency Timer with LHOLDREQ[1:0]; therefore, the Local Bus Latency Timer is significant only when LHOLDREQ[1:0] input is asserted.	P, L	P, L, E	0
9	**Direct Slave LOCK Enable.** Value of 0 disables Direct Slave locked sequences. Value of 1 enables Direct Slave locked sequences.	P, L	P, L, E	0
8	**Local Bus Direct Slave Release Bus Mode.** Value of 1 indicates the IOP 480 de-asserts HOLD and releases the Local Bus when the Direct Slave Write FIFO becomes empty during a Direct Slave write or when a Direct Slave Read FIFO becomes full during a Direct Slave read.	P, L	P, L, E	0
7	**External Preempt Enable.** Value of 0 indicates the Refresh cycle and Local Latency Timer cannot preempt external Local Masters. Value of 1 indicates that LHOLDACK1/BREQ output is asserted to request an external Local Master to release the Local Bus when there is a Refresh cycle or the Local Latency Timer is expired (internal Local arbiter must be enabled in this case).	P, L	P, L	1
6	**Local Bus Park Enable.** Value of 0 indicates the IOP 480 does not drive the LAD and LBE buses when the Local Bus is idle. In this case, external pull-up resistors should be used to drive the LAD and LBE buses to a known value. Value of 1 indicates the LAD and LBE buses are driven by the IOP 480 when the internal Local Bus arbiter is enabled, and the Local Bus is idle.	P, L	P, L	1
5:4	**DMA Channel Priority.** Value of 00 indicates a rotational priority scheme. Value of 01 indicates Channel 0 has priority. Value of 10 indicates Channel 1 has priority. Value of 11 is ***reserved*** (Local Bus arbiter.)	P, L	P, L, E	00
3:1	**Local Arbitration Priority.** Determines the Local Bus arbiter operation. If an individual requester is given priority, the other requesters still participate in a round-robin arbitration. The prioritized individual requester alternates with the round-robin arbitration winner. 000 Round-robin priority 001 IOP 480 CPU has priority 010 DMA Channels 0 and 1 have priority 011 Direct Slave has priority 100 LHOLDREQ0/LHOLDACK has priority 101 LHOLDREQ1 has priority 110 DMA Channel 2 has priority 111 ***Reserved***	P, L	P, L, E	000
0	**Local Arbiter Enable.** Value of 0 indicates the Local Bus arbiter is disabled, and LHOLDREQ0/LHOLDACK and LHOLDACK0/LDREQ are used by the IOP 480 to acquire Local Bus use. Value of 1 indicates this bit enables the Local Bus arbiter.	P, L	P, L, E	1

Register 17-62. (BIGEND; PCI:94h, LOC:94h) Big/Little Endian

Bit	Description	Read	Write	Value after Reset
31:4	***Reserved.***	P, L	No	0h
3	**Direct Master Big Endian Mode (for internal IOP 480 CPU).** Value of 0 specifies Little Endian ordering. Value of 1 specifies use of Big Endian data ordering for Direct Master accesses.	P, L	P, L, E	0
2	**Configuration Register Big Endian Mode (for internal IOP 480 CPU).** Value of 0 specifies Little Endian ordering. Value of 1 specifies use of Big Endian data ordering for Local accesses to the Configuration registers.	P, L	P, L, E	0
1	**Direct Master Big Endian Mode (for external Local Bus Master).** Value of 0 specifies Little Endian ordering. Value of 1 specifies use of Big Endian data ordering for Direct Master accesses.	P, L	P, L, E	0
0	**Configuration Register Big Endian Mode (for external Local Bus Master).** Value of 0 specifies Little Endian ordering. Value of 1 specifies use of Big Endian data ordering for Local accesses to the Configuration registers.	P, L	P, L, E	0

Register 17-63. (PCICTL; PCI:98h, LOC:98h) PCI Bus Control

Bit	Description	Read	Write	Value after Reset
31:28	Direct Slave **Retry Delay Clocks.** Contains the value (multiplied by 8) of the number of PCI Bus clocks after receiving a PCI Local Read or Write access and not successfully completing a transfer. Pertains only to Direct Slave writes when bit 27 is set to 1 (PCICTL[27]=1).	P, L	P, L, E	4 (32 clocks)
27	**Insert Direct Slave Wait States on Direct Slave Write FIFO Full.** Value of 0 indicates the IOP 480 should disconnect when the Direct Slave Write FIFO is full. Value of 1 indicates the IOP 480 should de-assert TRDY# when the Write FIFO is full.	P, L	P, L, E	0
26	**De-Assert PCI Request at FRAME#.** Value of 0 causes the IOP 480 to leave REQ0# asserted for the entire Bus Master cycle. Value of 1 causes the IOP 480 to de-assert REQ0# when it asserts FRAME# during a Master cycle. Valid only when the IOP 480 PCI arbiter is disabled.	P, L	P, L, E	1
25	**PCI Delayed Read Mode.** Value of 0 indicates the IOP 480 does not return TRDY# to the PCI Host until read data is available. Value of 1 indicates the IOP 480 operates in Delayed Transaction mode for Direct Slave reads. The IOP 480 issues a RETRY and prefetches the Read data. Considered a pending read until the PCI Host returns for the read data.	P, L	P, L, E	1
24	**Retry PCI Writes During Pending Reads.** Value of 0 allows writes to occur while a read is pending. Value of 1 forces a Retry on writes if read is pending.	P, L	P, L, E	0
23	**Flush Pending Reads on PCI Writes.** Value of 1 flushes pending Read cycles when a Write cycle is detected.	P, L	P, L, E	0
22	**PCI Read Ahead Mode.** Value of 0 submits request to flush the Read FIFO when a PCI Read cycle completes. Value of 1 submits a request to not flush the Read FIFO when the PCI Read cycle completes.	P, L	P, L, E	0
21	**Treat 256 PCI Retries as Abort.** Value of 0 enables the IOP 480 to attempt Master Retries indefinitely. Value of 1 enables the IOP 480 to treat 256 Master consecutive Retries from a Direct Slave as a Target abort.	P, L	P, L	0

Register 17-63. (PCICTL; PCI:98h, LOC:98h) PCI Bus Control (Continued)

Bit	Description	Read	Write	Value after Reset
20	**Flush Pending PCI Reads on Disconnect (by the IOP 480).** Value of 1 forces the IOP 480 to flush any pending reads after the IOP 480 issues a disconnect (versus a Retry). Value of 0 makes no distinction between a disconnect or a Retry; therefore, it does not flush pending reads, waiting for the reading Master to return for the Read data.	P, L	P, L	0
19	**PCI Arbiter Parking on IOP 480.** Value of 1 indicates the PCI arbiter parks the grant on the IOP 480. Value of 0 indicates the PCI arbiter parks the grant on the current PCI Master (when using the Internal PCI arbiter).	P, L	P, L	0
18	**Early Grant Release.** Value of 0 indicates the IOP 480 keeps GNT# asserted until another Master requests use of the PCI Bus. Value of 1 indicates the IOP 480 always de-asserts the current GNT# when FRAME# is asserted (when using the Internal PCI arbiter).	P, L	P, L, E	0
17	**IOP 480 High Priority.** Value of 0 indicates the IOP 480 participates in round-robin arbitration with the other PCI Masters. Value of 1 indicates a two-level, round-robin arbitration scheme is enabled. The other PCI Bus Masters participate in their own round-robin arbitration. The winner of this arbitration then arbitrates for the PCI Bus with the IOP 480 (when using the Internal PCI arbiter).	P, L	P, L, E	0
16	**PCI Arbiter Enable.** Value of 0 indicates the PCI arbiter is disabled and REQ0# and GNT0# are used by the IOP 480 to acquire PCI Bus use. Value of 1 indicates the PCI arbiter is enabled.	P, L	P, L, E	0
15:12	**PCI Memory Write Command Code for Direct Master.**	P, L	P, L, E	0111
11:8	**PCI Memory Read Command Code for Direct Master.**	P, L	P, L, E	0110
7:4	**PCI Write Command Code for DMA.**	P, L	P, L, E	0111
3:0	**PCI Read Command Code for DMA.**	P, L	P, L, E	1110

Register 17-64. (LAS0RR; PCI:A0h, LOC:A0h) Memory-Mapped Configuration Register and Local Address Space 0 Range Register for PCI-to-Local Bus

Bit	Description	Read	Write	Value after Reset
31:10	**Local Space 0 Range.** Specifies which PCI address bits to use for decoding a PCI access to the Memory-Mapped Configuration registers and Local Address Space 0. Each bit corresponds to a PCI address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in the decoding. Write 0 to all other bits (used in conjunction with PCIBAR0). The default is 1 KB, which corresponds to the Configuration register space.	P, L	P, L, E	FFFFC0h
9:0	***Reserved.***	P, L	No	0h

Register 17-65. (LAS0BA; PCI:A4h, LOC:A4h) Local Address Space 0 Base Address (Remap) for PCI-to-Local Bus

Bit	Description	Read	Write	Value after Reset
31:10	**Local Space 0 Remap.** Remap of PCI Address Space to Local Address Space 0 into a Local Address Space. Bits in this register remap (replace) the PCI Address bits used for decoding as the Local Address bits.	P, L	P, L, E	0h
9:0	***Reserved.***	P, L	No	0h

Register 17-66. (LAS1RR; PCI:A8h, LOC:A8h) Local Address Space 1 Range Register for PCI-to-Local Bus

Bit	Description	Read	Write	Value after Reset
31:4	**Local Space 1 Range.** Specifies which PCI Address bits to use for decoding a PCI access to Local Address Space 1. Each bit corresponds to a PCI Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in the decoding. Write 0 to all other bits (used in conjunction with PCIBAR1). Default is 1 MB.	P, L	P, L, E	FFF0000h
3	**Prefetchable.** If mapped into Memory space, a value of 1 indicates reads are prefetchable (bit has no effect on the operation of IOP 480, but is for system status). If mapped into I/O space, the bit is included with bits [31:2] to indicate the decoding range.	P, L	P, L, E	0
2:1	**Address Location.** If mapped into Memory Space, encoding is as follows: 2/1 Meaning 0 0 Locate anywhere in 32-bit PCI Address space 0 1 Locate below 1 MB in PCI Address space 1 0 Locate anywhere in 64-bit PCI Address space 1 1 ***Reserved*** If mapped into I/O space, bit 1 must be set to 0. Bit 2 is included with bits [31:3] to indicate decoding range.	P, L	P, L, E	00
0	**Memory Space Indicator.** Value of 0 indicates the Local Address Space 1 maps into PCI Memory space. Value of 1 indicates the Local Address Space 1 maps into PCI I/O space.	P, L	P, L, E	0

Register 17-67. (LAS1BA; PCI:ACh, LOC:ACh) Local Address Space 1 Base Address (Remap) for PCI-to-Local Bus

Bit	Description	Read	Write	Value after Reset
31:4	**Local Space 1 Remap.** Remap of PCI Address Space to Local Address Space 1 into a Local Address Space. Bits in this register remap (replace) the PCI Address bits used for decoding as the Local Address bits.	P, L	P, L, E	0h
3:2	**Local Space 1 Remap.** If Local Address Space 1 is mapped into Memory space, bits are not used. If mapped into I/O space, bits are included with bits [31:4] for remapping.	P, L	P, L, E	00
1	***Reserved.***	P, L	No	0
0	**Space 1 Enable.** Value of 0 disables decoding of this Address space. Value of 1 enables decoding of PCI addresses for Direct Slave access to Local Address Space 1.	P, L	P, L, E	0

Register 17-68. (LAS2RR; PCI:B0h, LOC:B0h) Local Address Space 2 Range Register for PCI-to-Local Bus

Bit	Description	Read	Write	Value after Reset
31:4	**Local Space 2 Range.** Specifies which PCI Address bits to use for decoding a PCI access to Local Address Space 2. Each bit corresponds to a PCI Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in the decoding. Write 0 to all other bits (used in conjunction with PCIBAR2). Default is 1 MB.	P, L	P, L, E	FFF0000h
3	**Prefetchable.** If mapped into Memory space, a value of 1 indicates reads are prefetchable (bit has no effect on the operation of IOP 480, but is for system status). If mapped into I/O space, bit is included with bits [31:2] to indicate decoding range.	P, L	P, L, E	0
2:1	**Address Location.** If mapped into Memory Space, encoding is as follows: 2/1 Meaning 0 0 Locate anywhere in 32-bit PCI Address space 0 1 Locate below 1 MB in PCI Address space 1 0 Locate anywhere in 64-bit PCI Address space 1 1 ***Reserved*** If mapped into I/O space, bit 1 must be set to 0. Bit 2 is included with bits [31:3] to indicate decoding range.	P, L	P, L, E	00
0	**Memory Space Indicator.** Value of 0 indicates the Local Address Space 2 maps into PCI Memory space. Value of 1 indicates the Local Address Space 2 maps into the PCI I/O space.	P, L	P, L, E	0

Register 17-69. (LAS2BA; PCI:B4h, LOC:B4h) Local Address Space 2 Base Address (Remap) for PCI-to-Local Bus

Bit	Description	Read	Write	Value after Reset
31:4	**Local Space 2 Remap.** Remap of PCI Address Space to Local Address Space 2 into a Local Address space. Bits in this register remap (replace) the PCI Address bits used for decoding as the Local Address bits.	P, L	P, L, E	0h
3:2	**Local Space 2 Remap**. If Local Address Space 2 is mapped into Memory space, bits are not used. If mapped into I/O space, bits are included with bits [31:4] for remapping.	P, L	P, L, E	00
1	***Reserved.***	P, L	No	0
0	**Space 2 Enable.** Value of 0 disables decoding of this Address space. Value of 1 enables decoding of PCI addresses for Direct Slave access to Local Address Space 2.	P, L	P, L, E	0

Register 17-70. (EROMRR; PCI:C0h, LOC:C0h) Expansion ROM Range

Bit	Description	Read	Write	Value after Reset
31:11	**EROM Range.** Specifies which PCI Address bits to use for decoding a PCI-to-Local Bus Expansion ROM. Each bit corresponds to a PCI Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in decode. Write 0 to all other bits (used in conjunction with PCIERBAR). Default is 64 KB.	P, L	P, L, E	FFFF00h
10:0	***Reserved.***	P, L	No	0h

Register 17-71. (EROMBA; PCI:C4h, LOC:C4h) Expansion ROM Local Base Address (Remap)

Bit	Description	Read	Write	Value after Reset
31:11	**EROM Remap.** Remap of PCI Expansion ROM space into a Local Address space. Bits in this register remap (replace) the PCI Address bits for decoding as the Local Address bits.	P, L	P, L, E	0h
10:0	***Reserved.***	P, L	No	0h

Register 17-72. (DMRR; PCI:C8h, LOC:C8h) Local Range Register for Direct Master-to-PCI

Bit	Description	Read	Write	Value after Reset
31:16	**Direct Master Range.** Specifies which Local Address bits to use for decoding a Local-to-PCI Bus access. Each bit corresponds to a PCI Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in decoding. Write 0 to all other bits. Changes the range, in increments of 64 KB.	P, L	P, L, E	0h
15:0	***Reserved.***	P, L	No	0h

Register 17-73. (DMLBAM; PCI:CCh, LOC:CCh) Local Bus Base Address Register for Direct Master-to-PCI Memory

Bit	Description	Read	Write	Value after Reset
31:16	**Local Direct Master Base Address.** Assigns a value to the bits to use for decoding Local-to-PCI Memory access.	P, L	P, L, E	0h
15:0	***Reserved.***	P, L	No	0h

Register 17-74. (DMPBAM; PCI:D0h, LOC:D0h) PCI Base Address (Remap) Register for Direct Master-to-PCI Memory (Lower 32 Bits)

Bit	Description	Read	Write	Value after Reset
31:16	**Direct Master Remap.** Remap of Local-to-PCI space into PCI Address space. Bits in this register remap (replace) the Local Address bits used in decode as the PCI Address bits.	P, L	P, L, E	0h
15	**I/O Remap Select.** Value of 0 uses bits [31:16] as PCI Address bits [31:16]. Value of 1 forces PCI Address bits [31:16] to all zeros on Direct Master I/O cycles.	P, L	P, L, E	0
14:13	**Direct Master Write Delay.** Delays the PCI Bus request after a Direct Master Burst Write cycle started. Values: 00 = No delay; start the cycle immediately 01 = Delay 4 PCI clocks 10 = Delay 8 PCI clocks 11 = Delay 16 PCI clocks	P, L	P, L, E	00
12:8	**Programmable Almost Full Flag.** When the number of entries in the 32-Lword Direct Master Write FIFO exceed this value, the output pin DMPAF# is asserted low.	P, L	P, L, E	0h
7	**Write and Invalidate Mode.** Value of 1 indicates the IOP 480 waits for 8 or 16 Lwords that must be written from the Local Bus before starting a PCI access. Value of 1 indicates that all Local Direct Master-to-PCI Write accesses must be 8- or 16-Lword bursts.	P, L	P, L, E	0
6	**Direct Master PCI Read Mode.** Value of 0 indicates the IOP 480 should release the PCI Bus when the Read FIFO becomes full. Value of 1 indicates the IOP 480 should keep the PCI Bus and de-assert IRDY# when Read FIFO becomes full.	P, L	P, L, E	0
5	**Direct Master Prefetch Limit.** Value of 1 indicates that prefetching is terminated at 4 KB boundaries.	P, L	P, L, E	0
4:3	**Direct Master Read Prefetch Size Control.** Values: 00 = IOP 480 continues to prefetch the Read data from the PCI Bus until the Direct Master access is finished. May result in an additional four unneeded words being prefetched from the PCI Bus. 01 = Prefetch up to 4 words from the PCI Bus 10 = Prefetch up to 8 words from the PCI Bus 11 = Prefetch up to 16 words from the PCI Bus Direct Master Burst reads should not exceed the programmed limit.	P, L	P, L, E	00
2	**LLOCK# Input Enable.** Value of 0 disables LLOCK# input. Value of 1 enables LLOCK# input, enabling PCI-locked sequences.	P, L	P, L, E	0
1	**Direct Master I/O Access Enable.** Value of 0 disables decode of Direct Master I/O accesses. Value of 1 enables decode of Direct Master I/O accesses.	P, L	P, L, E	0
0	**Direct Master Memory Access Enable.** Value of 0 disables decode of Direct Master Memory accesses. Value of 1 enables decode of Direct Master Memory accesses.	P, L	P, L, E	0

Register 17-75. (DMDAC; PCI:D4h, LOC:D4h) Direct Master Dual Address Cycle Upper Address

Bit	Description	Read	Write	Value after Reset
31:0	**Direct Master PCI Dual Address Cycle Upper Address.** During PCI Dual Address Cycles provides the upper 32 bits of the PCI Address for Memory cycles. Used for 64-bit PCI addressing on devices that do not support the 64-bit addressing bus (PCI Single Address cycle), but support 64-bit addressing with an additional Address phase (PCI Dual Address cycle). Direct Master PCI Dual Address cycles are enabled only when this register is a value other than 0.	P, L	P, L, E	0h

Register 17-76. (DMLBAI; PCI:D8h, LOC:D8h) Local Base Address Register for Direct Master-to-PCI IO/CFG

Bit	Description	Read	Write	Value after Reset
31:16	**Direct Master IO/CFG Local Base Address.** Assigns a value to the bits to use for decoding a Local-to-PCI I/O or Configuration access.	P, L	P, L, E	0h
15:0	***Reserved.***	P, L	No	0h

Register 17-77. (DMCFGA; PCI:DCh, LOC:DCh) PCI Configuration Address Register for Direct Master-to-PCI IO/CFG

Bit	Description	Read	Write	Value after Reset
31	**Configuration Enable.** Value of 1 allows Local-to-PCI I/O accesses converted to a PCI Configuration cycle. Parameters in this table are used to generate the PCI Configuration address.	P, L	P, L, E	0
30:24	***Reserved.***	P, L	No	0h
23:16	**Bus Number.**	P, L	P, L, E	0h
15:11	**Device Number.**	P, L	P, L, E	0h
10:8	**Function Number.**	P, L	P, L, E	000
7:2	**Register Number.**	P, L	P, L, E	0h
1:0	**Configuration Type.** Values: 00 = Type 0 01 = Type 1	P, L	P, L, E	00

Register 17-78. (CFGBA; PCI:E0h, LOC:E0h) Configuration Base Address

Bit	Description	Read	Write	Value after Reset
31:10	**Configuration Base Address.** Defines the Base address of the 1 KB Address space used to access the Configuration registers from the IOP 480 CPU or the Local Bus Master.	P, L	P, L, E	0101_0000_0000_0000_0000_00
9:0	***Reserved.***	P, L	No	0h

Register 17-79. (UARTBA; PCI:E4h, LOC:E4h) UART Base Address

Bit	Description	Read	Write	Value after Reset
31:24	**UART Base Address.** Defines the Base address of the 32-byte Address space used to access the UART from the IOP 480 CPU or the Local Bus Master.	P, L	P, L, E	0100_0000
23:0	***Reserved.***	P, L	No	0h

Register 17-80. (PLXID; PCI:E8h, LOC:E8h) PLX Hardcoded Configuration ID

Bit	Description	Read	Write	Value after Reset
31:16	**PLX Device ID.** Identifies the particular device. Hardcoded to the PLX part number for the IOP 480 (0480h).	P, L	No	0480h
15:0	**PLX Vendor ID.** Identifies the device manufacturer. Hardcoded to the PCI SIG-issued Vendor ID of PLX (10B5h).	P, L	No	10B5h

Register 17-81. (PLXREV; PCI:ECh, LOC:ECh) PLX Hardcoded Revision ID

Bit	Description	Read	Write	Value after Reset
31:8	***Reserved.***	P, L	No	0h
7:0	**PLX Revision ID.** Hardcoded silicon revision of the IOP 480.	P, L	No	Current Revision

17.6 MEMORY CONTROLLER REGISTERS

Note: *This Bus region is used for the boot ROM if no programmed serial EEPROM is detected.*

Register 17-82. (LCS0BRD; PCI:100h, LOC:100h) LCS0 Bus Region Descriptor

Bit	Description	Read	Write	Value after Reset
31:20	***Reserved.***	P, L	No	0h
19	**Ready/Recover Enable.** Value of 0 indicates the READY# pin is driven only with the IOP 480 internal ready status. Value of 1 indicates the READY# pin is also driven active during recovery states. Can be used to prevent external Local Bus Masters from starting a new cycle until the recovery period expires.	P, L	P, L, E	0
18:17	**Read Prefetch Count.** Number of words to prefetch when the the Direct Slave controller or the DMA controller is reading this memory. Used only when the Prefetch Count is enabled (LCS0BRD[16]=1). Values: 00 = Don't prefetch. Only read bytes specified by C/BE lines. 01 = Prefetch 4 words if bit 16 is set. 10 = Prefetch 8 words if bit 16 is set. 11 = Prefetch 16 words if bit 16 is set.	P, L	P, L, E	00
16	**Read Prefetch Count Enable.** Value of 0 indicates the count is ignored and prefetching continues until it is terminated by the PCI Bus or a page boundary is reached. Value of 1 indicates the IOP 480 prefetches up to the number of words specified by the Read Prefetch Count bits [18:17].	P, L	P, L, E	0
15	***Reserved.***	P, L	P, L, E	0
14	**Timeout Enable.** Value of 0 indicates the Local Bus Timeout Timer is disabled if an external Master controls the Local Bus. Value of 1 indicates the Local Bus Timeout Timer is enabled if an external Master controls the Local Bus.	P, L	P, L, E	0
13	**Parity Select.** Value of 0 indicates even parity is selected. Value of 1 indicates odd parity is selected.	P, L	P, L, E	0
12	**Parity Checking.** Value of 0 indicates parity checking is disabled. Value of 1 indicates parity checking is enabled. When parity is disabled, MA[16:13] are driven out onto the DP[3:0]/MA[16:13] pins if this region is accessed, and a 0 is written to this bit by the serial EEPROM or a CPU.	P, L	P, L, E	0
11	**Memory Write Protect.** Value of 0 indicates the MDQM[3:0]# and MWE# signals are asserted during Write cycles. Value of 1 indicates the MDQM[3:0]# and MWE# signals are not asserted during Write cycles (write protected).	P, L	P, L, E	0
10	**BTERM# Input Enable.** Value of 0 indicates BTERM# input is disabled and the burst length is limited to four words. Value of 1 indicates BTERM# input is enabled, and bursts continue until BTERM# is asserted or the corresponding FIFO becomes empty or full. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer.	P, L	P, L, E	0
9	**READY# Input Enable.** Value of 0 indicates READY# input is disabled, and the number of wait states is determined by the internal wait state generator. Value of 1 indicates READY# input is enabled, and determines when a Read or Write transfer occurs. Set to 1 when the Enable Target device is a peripheral that requires an unknown number of wait states for access.	P, L	P, L, E	0

Register 17-82. (LCS0BRD; PCI:100h, LOC:100h) LCS0 Bus Region Descriptor (Continued)

Bit	Description	Read	Write	Value after Reset
8	**Burst Enable.** Value of 0 indicates bursting is disabled. Value of 1 indicates bursting is enabled and the Target device must be able to accept Burst transfers. Clear bit if the Target device can accept only Single-Cycle transfers. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer. Burst cycles from the PCI Direct Slave or DMA controllers are terminated when a page boundary is reached. The internal IOP 480 CPU and external Local Bus masters should burst only to devices that can accept Burst cycles, and must prevent bursting across page boundaries.	P, L	P, L, E	0
7:5	***Reserved.***	P, L	No	000
4	**IOP 480 CPU Byte Ordering.** Determines byte ordering when IOP 480 CPU is the Local Bus Master. Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
3	**Big Endian Byte Lane Mode.** Value of 0 specifies that in Big Endian mode byte lanes [15:0] are used for a 16-bit bus, and byte lanes [7:0] for an 8-bit bus. Value of 1 specifies that in Big Endian mode byte lanes [31:16] be used for a 16-bit bus, and byte lanes [31:24] for an 8-bit bus.	P, L	P, L, E	0
2	**Direct Slave Byte Ordering.** Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
1:0	**Local Bus Width.** Specifies Data Bus width to the devices using this chip select. Values: 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = ***Reserved***	P, L	P, L, E	00

Register 17-83. (LCS0WT; PCI:104h, LOC:104h) LCS0 Write Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**WRCV.** Number of Write Recovery states (1–8). Recovery states = WRCV+1 (*for example*, WRCV=001, Recovery states = 2).	P, L	P, L, E	000
15:14	***Reserved.***	P, L	No	00
13:11	**WHLD.** Number of Write Hold (MDQM#, MWE#) states (0–7).	P, L	P, L, E	000
10:8	**WDLY.** Number of Write Enable (MDQM# and MWE#) Delay states (0–7).	P, L	P, L, E	000
7:4	**WDD.** Number of Write Data-to-Data wait states (0–15).	P, L	P, L, E	0000
3:0	**WAD.** Number of Write Address-to-Data wait states (0–15).	P, L	P, L, E	0000

Register 17-84. (LCS0RT; PCI:108h, LOC:108h) LCS0 Read Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**RRCV.** Number of Read Recovery states (1–8: value + 1).	P, L	P, L, E	111
15:14	***Reserved.***	P, L	No	00
13:11	**RDLYD.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	000
10:8	**RDLYA.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	001
7:4	**RDD.** Number of Read Data-to-Data wait states (0–15).	P, L	P, L, E	0111
3:0	**RAD.** Number of Read Address-to-Data wait states (0–15).	P, L	P, L, E	0111

Register 17-85. (LCS0BASE; PCI:10Ch, LOC:10Ch) LCS0 Base Address

Bit	Description	Read	Write	Value after Reset
31:8	**LCS0# Local Base Address.** Defines the LCS0 Memory Region Base address. The resolution is 256 bytes. The Default Base address is 0xFFF00000, which allows the internal IOP 480 CPU to boot from a serial EEPROM at location 0xFFFFFFFC.	P, L	P, L, E	1111_1111_1111_0000_0000_0000
7:1	***Reserved.***	P, L	No	0h
0	**LCS0# Enable.** Value of 0 indicates that LCS0# is disabled. Value of 1 indicates that LCS0# is enabled. Because LCS0# is output on a multiplexed pin, the LCS0# function must be selected in the LOCCTL Configuration register before using this Bus region (LOCCTL[0]).	P, L	P, L, E	1

Register 17-86. (LCS0RANGE; PCI:110h, LOC:110h) LCS0 Range

Bit	Description	Read	Write	Value after Reset
31:8	**LCS0 Range.** Specifies which Local Address bits to use for decoding accesses to the LCS0 Memory region. Each bit corresponds to a Local Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in decoding. Write 0 to all other bits. Default is 1 MB.	P, L	P, L, E	1111_1111_1111_0000_0000_0000
7:0	***Reserved.***	P, L	No	0h

Register 17-87. (LCS1BRD; PCI:114h, LOC:114h) LCS1 Bus Region Descriptor

Bit	Description	Read	Write	Value after Reset
31:20	***Reserved.***	P, L	No	0h
19	**Ready/Recover Enable.** Value of 0 indicates the READY# pin is driven only with the IOP 480 internal ready status. Value of 1 indicates the READY# pin is also driven active during recovery states. Can be used to prevent external Local Bus Masters from starting a new cycle until the recovery period expires.	P, L	P, L, E	0
18:17	**Read Prefetch Count.** Number of words to prefetch when the Direct Slave controller or DMA controller is reading this memory. Only used if Prefetch Count is enabled (LCS1BRD[1]=1). Values: 00 = Don't prefetch. Only read bytes specified by C/BE lines. 01 = Prefetch 4 words if bit 16 is set. 10 = Prefetch 8 words if bit 16 is set. 11 = Prefetch 16 words if bit 16 is set.	P, L	P, L, E	00
16	**Read Prefetch Count Enable.** Value of 0 indicates the count is ignored and prefetching continues until it is terminated by the PCI Bus or a page boundary is reached. Value of 1 indicates the IOP 480 prefetches up to the number of words specified by the Read Prefetch Count bits [18:17].	P, L	P, L, E	0
15	***Reserved.***	P, L	No	0
14	**Timeout Enable.** Value of 0 indicates that the Local Bus Timeout Timer is disabled if an external Master controls the Local Bus. Value of 1 indicates the Local Bus Timeout Timer is enabled if an external Master controls the Local Bus.	P, L	P, L, E	0
13	**Parity Select.** Value of 0 indicates even parity is selected. Value of 1 indicates odd parity is selected.	P, L	P, L, E	0
12	**Parity Checking.** Value of 0 indicates that parity checking is disabled. Value of 1 indicates that parity checking is enabled. When parity is disabled, MA[16:13] are driven out onto the DP[3:0]/MA[16:13] pins if this region is accessed, and a 0 is written to this bit by the serial EEPROM or a CPU.	P, L	P, L, E	0
11	**Memory Write Protect.** Value of 0 indicates the MDQM[3:0]# and MWE# signals are asserted during Write cycles. Value of 1 indicates the MDQM[3:0]# and MWE# signals are not asserted during Write cycles (write-protected).	P, L	P, L, E	0
10	**BTERM# Input Enable.** Value of 0 indicates that BTERM# input is disabled and the burst length is limited to 4 words. Value of 1 indicates that BTERM# input is enabled, and bursts continue until BTERM# is asserted or the corresponding FIFO becomes empty or full. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer.	P, L	P, L, E	0
9	**READY# Input Enable.** Value of 0 indicates that READY# input is disabled, and the number of wait states is determined by the internal wait state generator. Value of 1 indicates that READY# input is enabled, and determines when a Read or Write transfer occurs. Enable when the Target device is a peripheral that requires an unknown number of wait states for access.	P, L	P, L, E	0
8	**Burst Enable.** Value of 0 indicates bursting is disabled. Value of 1 indicates bursting is enabled and the Target device must be able to accept Burst transfers. This bit should be cleared if the Target device can accept only Single-Cycle transfers. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer. Burst cycles from PCI Direct Slave or DMA controllers are terminated when a page boundary is reached. The internal IOP 480 CPU and external Local Bus masters should burst only to devices that can accept Burst cycles, and must prevent bursting across page boundaries.	P, L	P, L, E	0
7:5	***Reserved.***	P, L	No	000

Register 17-87. (LCS1BRD; PCI:114h, LOC:114h) LCS1 Bus Region Descriptor (Continued)

Bit	Description	Read	Write	Value after Reset
4	**IOP 480 CPU Byte Ordering.** Determines byte ordering when IOP 480 CPU is the Local Bus Master. Value of 0 indicates Little Endian byte ordering is selected. Value of 1 indicates Big Endian byte ordering is selected.	P, L	P, L, E	0
3	**Big Endian Byte Lane Mode.** Value of 0 specifies that in Big Endian mode byte lanes [15:0] are used for a 16-bit bus, and byte lanes [7:0] for an 8-bit bus. Value of 1 specifies that in Big Endian mode byte lanes [31:16] be used for a 16-bit bus, and byte lanes [31:24] for an 8-bit bus.	P, L	P, L, E	0
2	**Direct Slave Byte Ordering.** Value of 0 indicates Little Endian byte ordering is selected. Value of 1 indicates Big Endian byte ordering is selected.	P, L	P, L, E	0
1:0	**Local Bus Width.** Specifies Data Bus width to the devices using this chip select. Values: 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = ***Reserved***	P, L	P, L, E	00

Register 17-88. (LCS1WT; PCI:118h, LOC:118h) LCS1 Write Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**WRCV.** Number of Write Recovery states (1–8: value + 1).	P, L	P, L, E	000
15:14	***Reserved.***	P, L	No	00
13:11	**WHLD.** Number of Write Hold (MDQM#, MWE#) states (0–7).	P, L	P, L, E	000
10:8	**WDLY.** Number of Write Enable (MDQM#, MWE#) Delay states (0–7).	P, L	P, L, E	000
7:4	**WDD.** Number of Write Data-to-Data wait states (0–15).	P, L	P, L, E	0000
3:0	**WAD.** Number of Write Address-to-Data wait states (0–15).	P, L	P, L, E	0000

Register 17-89. (LCS1RT; PCI:11Ch, LOC:11Ch) LCS1 Read Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**RRCV.** Number of Read Recovery states (1–8: value + 1).	P, L	P, L, E	000
15:14	***Reserved.***	P, L	No	00
13:11	**RDLYD.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	000
10:8	**RDLYA.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	001
7:4	**RDD.** Number of Read Data-to-Data wait states (0–15).	P, L	P, L, E	0000
3:0	**RAD.** Number of Read Address-to-Data wait states (0–15).	P, L	P, L, E	0001

Register 17-90. (LCS1BASE; PCI:120Ch, LOC:120Ch) LCS1 Base Address

Bit	Description	Read	Write	Value after Reset
31:8	**LCS1# Local Base Address.** Defines the LCS1 Memory Region Base address. The resolution is 256 bytes.	P, L	P, L, E	0h
7:1	***Reserved.***	P, L	No	0h
0	**LCS1# Enable.** Value of 0 indicates LCS1# is disabled. Value of 1 indicates LCS1# is enabled. Because LCS1# is output on a multiplexed pin, the LCS1# function must be selected in the LOCCTL Configuration register (LOCCTL[1]=0) before using this Bus region.	P, L	P, L, E	0

Register 17-91. (LCS1RANGE; PCI:124h, LOC:124h) LCS1 Range

Bit	Description	Read	Write	Value after Reset
31:8	**LCS1 Range.** Specifies which Local Address bits to use for decoding accesses to the LCS1 Memory region. Each bit corresponds to a Local Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in decoding. Write 0 to all other bits. Default is 1 MB.	P, L	P, L, E	1111_1111_ 1111_0000_ 0000_0000
7:0	***Reserved.***	P, L	No	0h

Register 17-92. (LCS2BRD; PCI:128h, LOC:128h) LCS2 Bus Region Descriptor

Bit	Description	Read	Write	Value after Reset
31:20	***Reserved.***	P, L	No	0h
19	**Ready/Recover Enable.** Value of 0 indicates the READY# pin is driven only with the IOP 480 internal ready status. Value of 1 indicates the READY# pin is also driven active during recovery states.Can be used to prevent external Local Bus Masters from starting a new cycle until the recovery period expires.	P, L	P, L, E	0
18:17	**Read Prefetch Count.** Number of words to prefetch when the Direct Slave controller or DMA controller is reading this memory. Used only when Prefetch Count is enabled (LCS2BRD[16]=1). Values: 00 = Don't prefetch. Only read bytes specified by C/BE lines. 01 = Prefetch 4 words if bit 16 is set. 10 = Prefetch 8 words if bit 16 is set. 11 = Prefetch 16 words if bit 16 is set.	P, L	P, L, E	00
16	**Read Prefetch Count Enable.** Value of 0 indicates the count is ignored and prefetching continues until it is terminated by the PCI Bus or a page boundary is reached. Value of 1 indicates the IOP 480 prefetches up to the number of words specified by the Read Prefetch Count bits [18:17].	P, L	P, L, E	0
15	***Reserved.***	P, L	No	0
14	**Timeout Enable.** Value of 0 indicates that the Local Bus Timeout Timer is disabled if an external Master controls the Local Bus. Value of 1 indicates the Local Bus Timeout Timer is enabled if an external Master controls the Local Bus.	P, L	P, L, E	0
13	**Parity Select.** Value of 0 indicates even parity is selected. Value of 1 indicates odd parity is selected.	P, L	P, L, E	0
12	**Parity Checking.** Value of 0 indicates parity checking is disabled. Value of 1 indicates parity checking is enabled. When parity is disabled, MA[16:13] is driven out on the DP[3:0]/MA[16:13] pins if this region is accessed, and a 0 is written to this bit by the serial EEPROM or a CPU.	P, L	P, L, E	0

Register 17-92. (LCS2BRD; PCI:128h, LOC:128h) LCS2 Bus Region Descriptor (Continued)

Bit	Description	Read	Write	Value after Reset
11	**Memory Write Protect.** Value of 0 indicates the MDQM[3:0]# and MWE# signals are asserted during Write cycles. Value of 1 indicates the MDQM[3:0]# and MWE# signals are not asserted during Write cycles (write-protected).	P, L	P, L, E	0
10	**BTERM# Input Enable.** Value of 0 indicates BTERM# input is disabled and the burst length is limited to 4 words. Value of 1 indicates BTERM# input is enabled, and bursts continue until BTERM# is asserted or the corresponding FIFO becomes empty or full. Used only when the IOP 480 PCI Direct Slave interface or one of the three internal DMA controllers are performing the transfer.	P, L	P, L, E	0
9	**READY# Input Enable.** Value of 0 indicates READY# input is disabled, and the number of wait states is determined by the internal wait state generator. Value of 1 indicates READY# input is enabled, and determines when a Read or Write transfer occurs. Enable when the Target device is a peripheral that requires an unknown number of wait states for access.	P, L	P, L, E	0
8	**Burst Enable.** Value of 0 indicates that bursting is disabled. Value of 1 indicates that bursting is enabled and the Target device must be able to accept Burst transfers. This bit should be cleared if the Target device can accept only Single-Cycle transfers. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer. Burst cycles from the PCI Direct Slave or DMA controllers are terminated when a page boundary is reached. The internal IOP 480 CPU and external Local Bus masters should burst only to devices that can accept Burst cycles, and must prevent bursting across page boundaries.	P, L	P, L, E	0
7:5	***Reserved.***	P, L	No	000
4	**IOP 480 CPU Byte Ordering.** Determines byte ordering when IOP 480 CPU is the Local Bus Master. Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
3	**Big Endian Byte Lane Mode.** Value of 0 specifies that in Big Endian mode byte lanes [15:0] are used for a 16-bit bus, and byte lanes [7:0] for an 8-bit bus. Value of 1 specifies that in Big Endian mode byte lanes [31:16] be used for a 16-bit bus, and byte lanes [31:24] for an 8-bit bus.	P, L	P, L, E	0
2	**Direct Slave Byte Ordering.** Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
1:0	**Local Bus Width.** Specifies Data Bus width to the devices using this chip select. Values: 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = ***Reserved***	P, L	P, L, E	00

Register 17-93. (LCS2WT; PCI:12Ch, LOC:12Ch) LCS2 Write Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**WRCV.** Number of Write Recovery states (1–8: value + 1).	P, L	P, L, E	000
15:14	***Reserved.***	P, L	No	00
13:11	**WHLD.** Number of Write Hold (MDQM#, MWE#) states (0–7).	P, L	P, L, E	000
10:8	**WDLY.** Number of Write Enable (MDQM#, MWE#) Delay states (0–7).	P, L	P, L, E	000
7:4	**WDD.** Number of Write Data-to-Data wait states (0–15).	P, L	P, L, E	0000
3:0	**WAD.** Number of Write Address-to-Data wait states (0–15).	P, L	P, L, E	0000

Register 17-94. (LCS2RT; PCI:130h, LOC:130h) LCS2 Read Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**RRCV.** Number of Read Recovery states (1–8: value + 1).	P, L	P, L, E	000
15:14	***Reserved.***	P, L	No	00
13:11	**RDLYD.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	000
10:8	**RDLYA.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	001
7:4	**RDD.** Number of Read Data-to-Data wait states (0–15).	P, L	P, L, E	0000
3:0	**RAD.** Number of Read Address-to-Data wait states (0–15).	P, L	P, L, E	0001

Register 17-95. (LCS2BASE; PCI:134h, LOC:134h) LCS2 Base Address

Bit	Description	Read	Write	Value after Reset
31:8	**LCS2# Local Base Address.** Defines the LCS2 Memory Region Base address. The resolution is 256 bytes.	P, L	P, L, E	0h
7:1	***Reserved.***	P, L	No	0h
0	**LCS2# Enable.** Value of 0 indicates LCS2# is disabled. Value of 1 indicates LCS2# is enabled. Because LCS2# is output on a multiplexed pin, the LCS2# function must be selected in the LOCCTL Configuration register (LOCCTL[4]=0) before using this Bus region.	P, L	P, L, E	0

Register 17-96. (LCS2RANGE; PCI:138h, LOC:138h) LCS2 Range

Bit	Description	Read	Write	Value after Reset
31:8	**LCS2 Range.** Specifies which Local Address bits to use for decoding accesses to the LCS2 Memory region. Each bit corresponds to a Local Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in decoding. Write 0 to all other bits. Default is 1 MB.	P, L	P, L, E	1111_1111_1111_0000_0000_0000
7:0	***Reserved.***	P, L	No	0h

Register 17-97. (LCS3BRD; PCI:13Ch, LOC:13Ch) LCS3 Bus Region Descriptor

Bit	Description	Read	Write	Value after Reset
31:20	***Reserved.***	P, L	No	0h
19	**Ready/Recover Enable.** Value of 0 indicates the READY# pin is driven only with the IOP 480 internal ready status. Value of 1 indicates the READY# pin is also driven active during recovery states. Can be used to prevent external Local Bus Masters from starting a new cycle until the recovery period expires.	P, L	P, L, E	0
18:17	**Read Prefetch Count.** Number of words to prefetch when the Direct Slave controller or DMA controller is reading this memory. Used only when Prefetch Count is enabled (LCS3BRD[16]=1). Values: 00 = Don't prefetch. Only read bytes specified by C/BE lines. 01 = Prefetch 4 words if bit 16 is set. 10 = Prefetch 8 words if bit 16 is set. 11 = Prefetch 16 words if bit 16 is set.	P, L	P, L, E	0
16	**Read Prefetch Count Enable.** Value of 0 indicates the count is ignored and prefetching continues until it is terminated by the PCI Bus or a page boundary is reached. Value of 1 indicates the IOP 480 prefetches up to the number of words specified by the Read Prefetch Count bits [18:17].	P, L	P, L, E	0
15	***Reserved.***	P, L	No	0
14	**Timeout Enable.** Value of 0 indicates the Local Bus Timeout Timer is disabled if an external Master controls the Local Bus. Value of 1 indicates the Local Bus Timeout Timer is enabled if an external Master controls the Local Bus.	P, L	P, L, E	0
13	**Parity Select.** Value of 0 indicates even parity is selected. Value of 1 indicates odd parity is selected.	P, L	P, L, E	0
12	**Parity Checking.** Value of 0 indicates parity checking is disabled. Value of 1 indicates parity checking is enabled. When parity is disabled, MA[16:13] is driven out on the DP[3:0]/MA[16:13] pins if this region is accessed, and a 0 is written to this bit by the serial EEPROM or a CPU.	P, L	P, L, E	0
11	**Memory Write Protect.** Value of 0 indicates the MDQM[3:0]# and MWE# signals are asserted during Write cycles. Value of 1 indicates the MDQM[3:0]# and MWE# signals are not asserted during Write cycles (write-protected).	P, L	P, L, E	0
10	**BTERM# Input Enable.** Value of 0 indicates that BTERM# input is disabled and the burst length is limited to 4 words. Value of 1 indicates that BTERM# input is enabled, and bursts continue until BTERM# is asserted or the corresponding FIFO becomes empty or full. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer.	P, L	P, L, E	0
9	**READY# Input Enable.** Value of 0 indicates the READY# input is disabled, and the number of wait states is determined by the internal wait state generator. Value of 1 indicates the READY# input is enabled, and determines when a Read or Write transfer occurs. Enable this bit when the Target device is a peripheral that requires an unknown number of wait states for accesses.	P, L	P, L, E	0
8	**Burst Enable.** Value of 0 indicates that bursting is disabled. Value of 1 indicates that bursting is enabled and the Target device must be able to accept Burst transfers. Clear this bit if the Target device can only accept Single-Cycle transfers. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer. Burst cycles from the PCI Direct Slave or DMA controllers are terminated when a page boundary is reached. The internal IOP 480 CPU and external Local Bus masters should burst only to devices that can accept Burst cycles, and must prevent bursting across page boundaries.	P, L	P, L, E	0
7:5	***Reserved.***	P, L	No	000

Register 17-97. (LCS3BRD; PCI:13Ch, LOC:13Ch) LCS3 Bus Region Descriptor (Continued)

Bit	Description	Read	Write	Value after Reset
4	**IOP 480 CPU Byte Ordering.** Determines byte ordering when IOP 480 CPU is the Local Bus Master. Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
3	**Big Endian Byte Lane Mode.** Value of 0 specifies that in Big Endian mode byte lanes [15:0] are used for a 16-bit bus, and byte lanes [7:0] for an 8-bit bus. Value of 1 specifies that in Big Endian mode byte lanes [31:16] be used for a 16-bit bus, and byte lanes [31:24] for an 8-bit bus.	P, L	P, L, E	0
2	**Direct Slave Byte Ordering.** Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
1:0	**Local Bus Width.** Specifies Data Bus width to the devices using this chip select. Values: 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = ***Reserved***	P, L	P, L, E	00

Register 17-98. (LCS3WT; PCI:140h, LOC:140h) LCS3 Write Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**WRCV.** Number of Write Recovery states (1–8: value + 1).	P, L	P, L, E	000
15:14	***Reserved.***	P, L	No	00
13:11	**WHLD.** Number of Write Hold (MDQM#, MWE#) states (0–7).	P, L	P, L, E	000
10:8	**WDLY.** Number of Write Enable (MDQM#, MWE#) Delay states (0–7).	P, L	P, L, E	000
7:4	**WDD.** Number of Write Data-to-Data wait states (0–15).	P, L	P, L, E	0000
3:0	**WAD.** Number of Write Address-to-Data wait states (0–15).	P, L	P, L, E	0000

Register 17-99. (LCS3RT; PCI:144h, LOC:144h) LCS3 Read Timing

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:16	**RRCV.** Number of Read Recovery states (1–8: value + 1).	P, L	P, L, E	000
15:14	***Reserved.***	P, L	No	00
13:11	**RDLYD.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	000
10:8	**RDLYA.** Number of Read Enable (RD# and MOE#) Delay states (0–7).	P, L	P, L, E	001
7:4	**RDD.** Number of Read Data-to-Data wait states (0–15).	P, L	P, L, E	0000
3:0	**RAD.** Number of Read Address-to-Data wait states (0–15).	P, L	P, L, E	0001

Register 17-100. (LCS3BASE; PCI:148h, LOC:148h) LCS3 Base Address

Bit	Description	Read	Write	Value after Reset
31:8	**LCS3# Local Base Address.** Defines the LCS3 Memory Region Base address. The resolution is 256 bytes.	P, L	P, L, E	0h
7:1	***Reserved.***	P, L	No	0h
0	**LCS3# Enable.** Value of 0 indicates that LCS3# is disabled. Value of 1 indicates that LCS3# is enabled. Because LCS3# is output on a multiplexed pin, the LCS3# function must be selected in the LOCCTL Configuration register (LOCCTL[7]=0) before using this Bus region.	P, L	P, L, E	0

Register 17-101. (LCS3RANGE; PCI:14Ch, LOC:14Ch) LCS3 Range

Bit	Description	Read	Write	Value after Reset
31:8	**LCS3 Range.** Specifies which Local Address bits to use for decoding accesses to the LCS3 Memory region. Each bit corresponds to a Local Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in decoding. Write 0 to all other bits. Default is 1 MB.	P, L	P, L, E	1111_1111_ 1111_0000_ 0000_0000
7:0	***Reserved.***	P, L	No	0h

Register 17-102. (DRAMBRD; PCI:150h, LOC:150h) DRAM Bus Region Descriptor

Bit	Description	Read	Write	Value after Reset
31:20	***Reserved.***	P, L	No	0h
19	**Ready/Recover Enable.** Value of 0 indicates the READY# pin is driven only with the IOP 480 internal ready status. Value of 1 indicates the READY# pin is also driven active during recovery states. Can be used to prevent external Local Bus Masters from starting a new cycle until the recovery period expires.	P, L	P, L, E	0
18:17	**Read Prefetch Count.** Number of words to prefetch when the Direct Slave controller or DMA controller is reading this memory. Used only when Prefetch Count is enabled (DRAMBRD[16]=1). Values: 00 = Don't prefetch. Only read bytes specified by C/BE lines. 01 = Prefetch 4 words if bit 16 is set. 10 = Prefetch 8 words if bit 16 is set. 11 = Prefetch 16 words if bit 16 is set.	P, L	P, L, E	00
16	**Read Prefetch Count Enable.** Value of 0 indicates the count is ignored and prefetching continues until it is terminated by the PCI Bus or a page boundary is reached. Value of 1 indicates the IOP 480 prefetches up to the number of words specified by the Read Prefetch Count bits [18:17].	P, L	P, L, E	0
15:14	***Reserved.***	P, L	No	00
13	**Parity Select.** Value of 0 indicates even parity is selected. Value of 1 indicates odd parity is selected.	P, L	P, L, E	0
12	**Parity Checking.** Value of 0 indicates that parity checking is disabled. Value of 1 indicates that parity checking is enabled. When parity is disabled, zeros are driven out on the DP[3:0]/MA[16:13] pins if this region is accessed, and a 0 is written to this bit by the serial EEPROM or a CPU.	P, L	P, L, E	0
11	**Memory Write Protect.** Value of 0 indicates the MDQM[3:0]# (for SDRAM) and MWE# (for EDO) signals are asserted during Write cycles. Value of 1 indicates the MDQM[3:0]# (for SDRAM) and MWE# (for EDO) signals are not asserted during Write cycles (write-protected).	P, L	P, L, E	0

Register 17-102. (DRAMBRD; PCI:150h, LOC:150h) DRAM Bus Region Descriptor (Continued)

Bit	Description	Read	Write	Value after Reset
10	**BTERM# Input Enable.** Value of 0 indicates that BTERM# input is disabled and the burst length is limited to 4 words. Value of 1 indicates that BTERM# input is enabled, and bursts continue until BTERM# is asserted or the corresponding FIFO becomes empty or full. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer.	P, L	P, L, E	0
9	***Reserved.***	P, L	No	0
8	**Burst Enable.** Value of 0 indicates that bursting is disabled. Value of 1 indicates that bursting is enabled and the Target device must be able to accept Burst transfers. This bit should be cleared if the Target device can accept only Single-Cycle transfers. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer. Burst cycles from PCI Direct Slave or DMA controllers are terminated when a page boundary is reached. The internal IOP 480 CPU and external Local Bus masters should burst only to devices that can accept Burst cycles, and must prevent bursting across page boundaries.	P, L	P, L, E	0
7:5	***Reserved.***	P, L	No	000
4	**IOP 480 CPU Byte Ordering.** Determines byte ordering when IOP 480 CPU is the Local Bus Master. Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
3	**Big Endian Byte Lane Mode.** Value of 0 specifies that in Big Endian mode byte lanes [15:0] are used for a 16-bit bus, and byte lanes [7:0] for an 8-bit bus. Value of 1 specifies that in Big Endian mode byte lanes [31:16] be used for a 16-bit bus, and byte lanes [31:24] for an 8-bit bus.	P, L	P, L, E	0
2	**Direct Slave Byte Ordering.** Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
1:0	**Local Bus Width.** Specifies Data Bus width to the devices using this chip select. Values: 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = ***Reserved***	P, L	P, L, E	00

Register 17-103. (DRAMCTL; PCI:154h, LOC:154h) DRAM Control

Bit	Description	Read	Write	Value after Reset
31:24	***Reserved.***	P, L	No	0h
23	**Refresh Enable.** Value of 0 indicates that Refresh cycles are disabled. Value of 1 indicates that Refresh cycles are enabled.	P, L	P, L, E	1
22:12	**Refresh Interval.** Determines the interval between Refresh cycles in terms of LCLK Clock cycles. The value can be calculated as follows: reg value = refresh rate x LCLK clock frequency. Typically: reg value = 15.625 µs x 66.666 MHz = 1041 = 0x411.	P, L	P, L, E	100_0000_0111
11:10	**Self-Refresh Level.** Determines the power management level at which an SDRAM initiates Low-Power Auto-Refresh mode. Values: 00 = No Self-Refresh (fully operational) 01 = Self-Refresh when in power management state D_1 or higher 10 = Self-Refresh when in power management state D_2 or higher 11 = Self-Refresh when in power management state D_3	P, L	P, L, E	10
9	**Page Mode Enable.** Value of 0 indicates Page mode is disabled. Value of 1 indicates Page mode is enabled. If Page mode is enabled, the CAS_MDQM[3:0]# pins are not asserted during SRAM cycles.	P, L	P, L, E	0
8:6	**Number of Columns.** Values: 000 8 columns 001 9 columns 010 10 columns 011 11 columns 100 12 columns 101-111 ***Reserved***	P, L	P, L, E	000
5:4	***Reserved.***	P, L	No	00
3:1	**Bank Size.** These bits set the maximum addressable memory size for each of the four banks of the IOP 480. Total addressable memory is four times the chosen bank size. Values: 000 4 MB 001 8 MB 010 16 MB 011 32 MB 100 64 MB 101-111 ***Reserved***	P, L	P, L, E	000
0	**DRAM Type.** Value of 0 indicates that EDO DRAMs are selected. Value of 1 indicates that SDRAMs are selected.	P, L	P, L, E	0

Note: *This register cannot be written to until all other DRAM registers have been set up, including enabling the SDRAM Reload Initialization bit (DRAM Initialization [14]) and enabling the DRAM Enable bit (DRAM Base Address [0]). Writing to this register prior to this crashes the Memory Controller.*

Register 17-104. (DRAMINIT; PCI:158h, LOC:158h) DRAM Initialization

Bit	Description	Read	Write	Value after Reset
31:17	***Reserved.***	P, L	No	0h
16	**SDRAM Initialization Status.** Initialized to 1 at reset, and automatically cleared when an SDRAM initialization is complete	P, L	No	1
15	**SDRAM Initialization Order.** Value of 0 indicates that "load mode" command is followed by eight Refresh cycles during initialization. Value of 1 indicates that eight Refresh cycles are followed by "load mode" command during initialization. Cleared when SDRAM initialization completes.	P, L	P, L, E	0
14	**Reload SDRAM Initialization Word.** Write this bit to reload the SDRAM initialization word with bits [13:0]. Automatically cleared when initialization completes.	P, L	Set	0
13:0	**SDRAM Initialization Word.** Described in the IOP 480 Memory Controller Specification. Determines the SDRAM operating modes. The value in these bits is automatically loaded into the SDRAM at power-on. Also used by the SRAM Memory Controller for determining the CAS latency value programmed into the SDRAMs. This bit must not be changed to a value other than that present when the SDRAM initialization occurs.	P, L	P, L, E	00_0000_0010_0111

Register 17-105. (DRAMTIM; PCI:15Ch, LOC:15Ch) DRAM Timing

Bit	Description	Read	Write	Value after Reset
31:22	***Reserved.***	P, L	No	0h
21:20	**TWR.** Delay between the last word written during a Single or Burst Write cycle and a Precharge command. Values Clock Period(s) 00 0 01 1 10 2 11 3	P, L	P, L, E	00
19:18	**W2W.** Delay between words of a Burst Write for an SDRAM Write cycle. Values Clock Period(s) 00 0 01 1 10 2 11 3	P, L	P, L, E	00
17:16	**A2C.** Active to Read/Write command delay for an SDRAM Read or Write cycle. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	01
15:14	***Reserved.***	P, L	No	0
13:12	**RRCV.** Number of recovery states after the end of a Single or Burst EDO or an SDRAM Read cycle. Used to provide the device with time to float its outputs. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	01
11:10	**PRCG.** RAS precharge delay for EDO or SDRAMs. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	10
9:8	**WCW.** CAS pulse width for an EDO Write cycle. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	00

Register 17-105. (DRAMTIM; PCI:15Ch, LOC:15Ch) DRAM Timing (Continued)

7:6	**RCW.** CAS pulse width for EDO Read cycle. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	00
5:4	**C2C.** Delay from column address to CAS for EDO cycle. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	00
3:2	**R2C.** Delay from RAS to column address for EDO cycle. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	01
1:0	**R2R.** Delay from row address to RAS for EDO cycle. Values Clock Period(s) 00 1 01 2 10 3 11 4	P, L	P, L, E	00

Register 17-106. (DRAMBASE; PCI:160h, LOC:160h) DRAM Base Address

Bit	Description	Read	Write	Value after Reset
31:22	**DRAM Local Base Address.** Defines the DRAM Memory Region Base address. The resolution is 4 MB.	P, L	P, L, E	0h
21:1	***Reserved.***	P, L	No	0h
0	**DRAM Enable.** Value of 0 indicates the DRAM controller is disabled. Value of 1 indicates the DRAM controller is enabled.	P, L	P, L, E	0

Register 17-107. (DRAMRANGE; PCI:164h, LOC:164h) DRAM Range

Bit	Description	Read	Write	Value after Reset
31:22	**DRAM Range.** Specifies which Local Address bits to use for decoding accesses to the DRAM Memory region. Each bit corresponds to a Local Address bit. Bit 31 corresponds to address bit 31. Write 1 to all bits included in decoding. Write 0 to all other bits. Default is 4 MB.	P, L	P, L, E	1111_1111_11
21:0	***Reserved.***	P, L	No	0h

Note: *The following Bus parameters are used when accesses to the Local Bus by the IOP 480 (PCI, DMA, or internal IOP 480 CPU) do not fall into one of the five Memory spaces. There are no programmable timing parameters for this space, and all cycles are terminated with BTERM# or READY#.*

Register 17-108. (DFLTBRD; PCI:168h, LOC:168h) Default Bus Region Descriptor

Bit	Description	Read	Write	Value after Reset
31:19	***Reserved.***	P, L	No	0h
18:17	**Read Prefetch Count.** Number of words to prefetch when the Direct Slave controller or DMA controller is reading this memory. Used only when the Prefetch Count is enabled (DFLTBRD[16]=1). Values: 00 = Don't prefetch. Only read bytes specified by C/BE lines. 01 = Prefetch 4 words if bit 16 is set. 10 = Prefetch 8 words if bit 16 is set. 11 = Prefetch 16 words if bit 16 is set.	P, L	P, L, E	00
16	**Read Prefetch Count Enable.** Value of 0 indicates the count is ignored and prefetching continues until it is terminated by the PCI Bus or a page boundary is reached. Value of 1 indicates the IOP 480 prefetches up to the number of words specified by the Read Prefetch Count bits [18:17].	P, L	P, L, E	0
15	**Timeout Ready Out Enable.** Value of 0 indicates the READY# pin is not driven when a timeout during accesses to a device in the Default region is detected. Value of 1 indicates the READY# pin is driven when a timeout is detected during accesses to a device in the Default region.	P, L	P, L	0
14	**Timeout Enable.** Value of 0 indicates the Local Bus Timeout Timer is disabled if an external Master controls the Local Bus. Value of 1 indicates the Local Bus Timeout Timer is enabled if an external Master controls the Local Bus.	P, L	P, L, E	0
13	**Parity Select.** Value of 0 indicates even parity is selected. Value of 1 indicates odd parity is selected.	P, L	P, L, E	0
12	**Parity Checking.** Value of 0 indicates parity checking is disabled. Value of 1 indicates parity checking is enabled. When parity is disabled, MA[16:13] is driven out on the DP[3:0]/MA[16:13] pins if this region is accessed, and a 0 is written to this bit by the serial EEPROM or a CPU.	P, L	P, L, E	0
11	**Memory Write Protect.** Value of 0 indicates the MDQM[3:0]# and MWE# signals are asserted during Write cycles. Value of 1 indicates the MDQM[3:0]# and MWE# signals are not asserted during Write cycles (write-protected).	P, L	P, L, E	0
10	**BTERM# Input Enable.** Value of 0 indicates BTERM# input is disabled and the burst length is limited to four words. Value of 1 indicates BTERM# input is enabled, and bursts continue until BTERM# is asserted or the corresponding FIFO becomes empty or full. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer.	P, L	P, L, E	0
9	***Reserved.***	P, L	No	0
8	**Burst Enable.** Value of 0 indicates that bursting is disabled. Value of 1 indicates bursting is enabled and that the Target device must be able to accept Burst transfers. Clear this bit if the Target device can only accept Single-Cycle transfers. Used only when the IOP 480 Direct Slave interface or one of the three internal DMA controllers are performing the transfer. Burst cycles from the PCI Direct Slave or DMA controllers are terminated when a page boundary is reached. The internal IOP 480 CPU and external Local Bus masters should burst only to devices that can accept Burst cycles, and must prevent bursting across page boundaries.	P, L	P, L, E	0
7:5	***Reserved.***	P, L	No	000

Register 17-108. (DFLTBRD; PCI:168h, LOC:168h) Default Bus Region Descriptor (Continued)

Bit	Description	Read	Write	Value after Reset
4	**IOP 480 CPU Byte Ordering.** Determines byte ordering when IOP 480 CPU is the Local Bus Master. Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
3	**Big Endian Byte Lane Mode.** Value of 0 specifies that in Big Endian mode byte lanes [15:0] are used for a 16-bit bus, and byte lanes [7:0] for an 8-bit bus. Value of 1 specifies that in Big Endian mode byte lanes [31:16] be used for a 16-bit bus, and byte lanes [31:24] for an 8-bit bus.	P, L	P, L, E	0
2	**Direct Slave Byte Ordering.** Value of 0 indicates that Little Endian byte ordering is selected. Value of 1 indicates that Big Endian byte ordering is selected.	P, L	P, L, E	0
1:0	**Local Bus Width.** Specifies the Data Bus width to the default devices. Values: 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = ***Reserved***	P, L	P, L, E	10

17.7 RUNTIME REGISTERS

Register 17-109. (MBOX0; PCI:180h, LOC:180h) Mailbox 0

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L, E	0h

Register 17-110. (MBOX1; PCI:184h, LOC:184h) Mailbox 1

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L, E	0h

Register 17-111. (MBOX2; PCI:188h, LOC:188h) Mailbox 2

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L	0h

Register 17-112. (MBOX3; PCI:18Ch, LOC:18Ch) Mailbox 3

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L	0h

Register 17-113. (MBOX4; PCI:190h, LOC:190h) Mailbox 4

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L	0h

Register 17-114. (MBOX5; PCI:194h, LOC:194h) Mailbox 5

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L	0h

Register 17-115. (MBOX6; PCI:198h, LOC:198h) Mailbox 6

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L	0h

Register 17-116. (MBOX7; PCI:19Ch, LOC:19Ch) Mailbox 7

Bit	Description	Read	Write	Value after Reset
31:0	**32-Bit Mailbox Register.**	P, L	P, L	0h

Register 17-117. (P2LDBELL; PCI:1A0h, LOC:1A0h) PCI-to-Local Doorbell

Bit	Description	Read	Write	Value after Reset
31:0	**Doorbell Register.** A PCI Master can write to this register and generate a Local interrupt to the Local processor. The Local processor can then read this register to determine which Doorbell bit was asserted. The PCI Master sets a doorbell by writing 1 to a particular bit. The Local processor can clear a Doorbell bit by writing 1 to that bit position.	P, L	P, L/Clr	0h

Register 17-118. (L2PDBELL; PCI:1A4h, LOC:1A4h) Local-to-PCI Doorbell

Bit	Description	Read	Write	Value after Reset
31:0	**Doorbell Register.** The Local processor can write to this register and generate a PCI interrupt. the PCI Master can then read this register to determine which Doorbell bit was asserted. The Local processor sets a doorbell by writing 1 to a particular bit. The PCI Master can clear a Doorbell bit by writing 1 to that bit position.	P, L	P, L/Clr	0h

Note: *The following bits determine when INTA# is asserted when enabled by PINTENB[0].*

Register 17-119. (PINTSTAT; PCI:1B0h, LOC:1B0h) PCI Interrupt Status

Bit	Description	Read	Write	Value after Reset
31:20	***Reserved.***	P, L	No	0h
19	**Target Abort Generated.** Value of 1 indicates a Target Abort was generated by the IOP 480 after 256 consecutive Master Retries to a target.	P, L	No	0
18	**DMA Channel 1—Master or Target Abort Detected.** Value of 1 indicates DMA Channel 1 was the Bus Master during a Master or Target abort.	P, L	No	0
17	**DMA Channel 0—Master or Target Abort Detected.** Value of 1 indicates DMA Channel 0 was the Bus Master during a Master or Target abort.	P, L	No	0
16	**Direct Master—Master or Target Abort Detected.** Value of 1 indicates a Direct Master was the Bus Master during a Master or Target abort.	P, L	No	0
15:14	***Reserved.***	P, L	No	00
13	**PCI Abort Interrupt.** Value of 1 indicates the PCI Abort interrupt is active.	P, L	No	0
12	**Local Interrupt.** Value of 1 indicates the Local interrupt is active.	P, L	No	0
11	**PCI Doorbell Interrupt.** Value of 1 indicates the PCI Doorbell interrupt is active.	P, L	No	0
10	**DMA Channel 2 Interrupt.** Value of 1 indicates the DMA CH 2 interrupt is active.	P, L	No	0
9	**DMA Channel 1 Interrupt.** Value of 1 indicates the DMA CH 1 interrupt is active.	P, L	No	0
8	**DMA Channel 0 Interrupt.** Value of 1 indicates the DMA CH 0 interrupt is active.	P, L	No	0
7:1	***Reserved.***	P, L	No	0h
0	**Generate PCI Bus SERR#.** Changing the value from 0 to 1 generates a PCI Bus SERR#.	P, L	P, L	0

Register 17-120. (PINTENB; PCI:1B4h, LOC:1B4h) PCI Interrupt Enable

Bit	Description	Read	Write	Value after Reset
31:14	***Reserved.***	P, L	No	0h
13	**PCI Abort Interrupt Enable.** Value of 1 enables a Master abort or Master detect of a Target abort to generate a PCI interrupt. Used in conjunction with PCI interrupt enable. Clearing the Abort Status bit PCISR[13] also clears the PCI interrupt.	P, L	P, L	0
12	**PCI Local Interrupt Enable.** Value of 1 enables a Local interrupt to generate a PCI interrupt. Use in conjunction with the PCI Interrupt Enable bit PINTENB[13]. Clearing the Local Bus cause of the interrupt also clears the interrupt.	P, L	P, L	0
11	**PCI Doorbell Interrupt Enable.** Value of 1 enables Doorbell interrupts. Used in conjunction with PCI interrupt enable. Clearing the Doorbell interrupt bits that caused the interrupt also clears the interrupt.	P, L	P, L	0
10	**DMA CH 2 Interrupt Enable.** Value of 1 enables a PCI interrupt to generate when the DMA Controller Channel 2 is done or has reached a terminal count.	P, L	P, L	0
9	**DMA CH 1 Interrupt Enable.** Value of 1 enables a PCI interrupt to generate when the DMA Controller Channel 1 is done or has reached a terminal count.	P, L	P, L	0
8	**DMA CH 0 Interrupt Enable.** Value of 1 enables a PCI interrupt to generate when the DMA Controller Channel 0 is done or has reached a terminal count.	P, L	P, L	0
7:1	***Reserved.***	P, L	No	0h
0	**PCI Interrupt Output Enable.** Value of 1 enables PCI interrupts.	P, L	P, L	1

Note: *The following bits determine when INTO is asserted, assuming they are enabled in the LINTENB register. INTO also drives the IOP 480 CPU External Interrupt input.*

Register 17-121. (LINTSTAT; PCI:1B8h, LOC:1B8h) Local Interrupt Status

Bit	Description	Read	Write	Value after Reset
31	**Mailbox 7 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 7. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
30	**Mailbox 6 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 6. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
29	**Mailbox 5 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 5. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
28	**Mailbox 4 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 4. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
27	**Mailbox 3 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 3. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
26	**Mailbox 2 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 2. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
25	**Mailbox 1 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 1. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
24	**Mailbox 0 Interrupt.** Value of 1 indicates the PCI wrote data to Mailbox 0. To clear the Local interrupt, the Local Master must read the mailbox.	P, L	No	0
23:22	***Reserved*.**	P, L	No	00
21	**Refresh Local Interrupt Status.** Value of 1 indicates a Refresh Local Interrupt occurs. Writing a 1 clears the interrupt. The Refresh INT is set when two Refresh requests occur without a grant occurring between the two requests. Value of 0 indicates that no Refresh Local Interrupt occurs.	P, L	P/Clr, L/Clr	0
20	**PCI SERR Interrupt.** Value of 1 indicates that an interrupt was received on the SERR# pin. Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
19:17	***Reserved*.**	P, L	No	000
16	**PCI ENUM# Interrupt.** Value of 1 indicates the ENUM# pin received an interrupt. Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
15	**PCI PME# Interrupt.** Value of 1 indicates the PME# pin received an interrupt. Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
14	**PCI INTA# Interrupt.** Value of 1 indicates the INTA# pin received an interrupt. Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
13	**Power Management Interrupt.** Value of 1 indicates a Power Management interrupt is pending. A Power Management interrupt is caused by a change in the Power State bits (PMCSR[1:0]). Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
12	**BIST Interrupt.** Value of 1 indicates BIST interrupt is active. A BIST (built-in self test) interrupt is generated by writing 1 to the BIST Interrupt bit (PCIBIST[6]=1). Clearing (PCIBIST[6]), clears the interrupt. Refer to the PCIBISTR register for a description of self test.	P, L	No	0
11	**Local Doorbell Interrupt.** Value of 1 indicates Local Doorbell interrupt is active. Clearing the Local Doorbell Interrupt clears the interrupt and, as a result, the Local Doorbell interrupt is de-activated.	P, L	No	0
10	**Channel 2 DMA Interrupt.** Value of 1 indicates the DMA CH 2 interrupt is active. Clearing the DMA status bits also clears the interrupt.	P, L	No	0
9	**Channel 1 DMA Interrupt.** Value of 1 indicates the DMA CH 1 interrupt is active. Clearing the DMA status bits also clears the interrupt.	P, L	No	0
8	**Channel 0 DMA Interrupt.** Value of 1 indicates the DMA CH 0 interrupt is active. Clearing the DMA status bits also clears the interrupt.	P, L	No	0

Register 17-121. (LINTSTAT; PCI:1B8h, LOC:1B8h) Local Interrupt Status (Continued)

Bit	Description	Read	Write	Value after Reset
7	**INTI Interrupt.** Value of 1 indicates an interrupt is pending on the INTI pin. Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
6	**Local Bus Parity Error.** Value of 1 indicates that a Local Bus Parity error has occurred. Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
5	**Serial Port Interrupt 2.** Value of 1 indicates the Serial Port Interrupt 2 is active.	P, L	No	0
4	**Serial Port Interrupt 1.** Value of 1 indicates the Serial Port Interrupt 1 is active.	P, L	No	0
3	**Local Bus Timeout Interrupt.** Value of 1 indicates a Local Bus timeout has occurred. Writing a 1 clears the interrupt.	P, L	P, L/Clr	0
2	**PCI Error Interrupt.** Value of 1 indicates a PCI Parity error or Messaging queue Outbound Free Queue Overflow interrupt is active.	P, L	No	0
1	**PCI Abort Interrupt.** Value of 1 indicates a PCI Master or Target Abort status is active.	P, L	No	0
0	***Reserved.***	P, L	No	0

Register 17-122. (LINTENB; PCI:1BCh, LOC:1BCh) Local Interrupt Enable

Bit	Description	Read	Write	Value after Reset
31	**Mailbox 7 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 7.	P, L	P, L	0
30	**Mailbox 6 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 6.	P, L	P, L	0
29	**Mailbox 5 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 5.	P, L	P, L	0
28	**Mailbox 4 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 4.	P, L	P, L	0
27	**Mailbox 3 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 3.	P, L	P, L	0
26	**Mailbox 2 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 2.	P, L	P, L	0
25	**Mailbox 1 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 1.	P, L	P, L	0
24	**Mailbox 0 Interrupt Enable.** Value of 1 enables a Local interrupt generated when the PCI Bus writes to Mailbox 0.	P, L	P, L	0
23:22	***Reserved.***	P, L	No	00
21	**Refresh Local Interrupt Enable.** Value of 1 indicates the Refresh Local Interrupt is enabled. The Refresh INT is set when two Refresh requests occur without a grant occurring between the two requests. Value of 0 indicates the Refresh Local Interrupt is disabled.	P, L	P, L	0
20	**PCI SERR Interrupt Enable Reserved.** Value of 1 enables a Local interrupt to generate when the PCI SERR# pin is asserted.	P, L	P, L	0
19	**CINT Polarity.** Value of 1 indicates active high polarity for the CINT interrupt input. Value of 0 indicates active low polarity for the CINT interrupt input.	P, L	P, L	0
18	**INTI Polarity.** Value of 1 indicates active high polarity for the INTI interrupt input. Value of 0 indicates active low polarity for the INTI interrupt input.	P, L	P, L	0

Register 17-122. (LINTENB; PCI:1BCh, LOC:1BCh) Local Interrupt Enable (Continued)

Bit	Description	Read	Write	Value after Reset
17	**INTO Polarity.** Value of 1 indicates active high polarity for the INTO interrupt output. Value of 0 indicates active low polarity for the INTO interrupt output.	P, L	P, L	0
16	**PCI ENUM# Interrupt Enable.** Value of 1 enables a Local interrupt to generate when the PCI ENUM# pin is asserted.	P, L	P, L	0
15	**PCI PME Interrupt Enable.** Value of 1 enables a Local interrupt to generate when the PCI PME# pin is asserted.	P, L	P, L	0
14	**PCI INTA Interrupt Enable.** Value of 1 enables a Local interrupt to generate when the PCI INTA# pin is asserted.	P, L	P, L	0
13	**Power Management Interrupt Enable.** Value of 1 enables a Local interrupt to generate when the Power Management Power State changes (PMCSR[1:0]).	P, L	P, L	0
12	**BIST Interrupt Enable.** Value of 1 enables a Local interrupt to generate when the Built-In Self Test Interrupt bit is set (PCIBIST[6]=1) by the PCI Host.	P, L	P, L	0
11	**Local Doorbell Interrupt Enable.** Value of 1 enables Doorbell interrupts. Used in conjunction with Local interrupt enable.	P, L	P, L	0
10	**Local DMA Channel 2 Interrupt Enable.** Value of 1 enables DMA Channel 2 interrupts. Used in conjunction with Local interrupt enable.	P, L	P, L	0
9	**Local DMA Channel 1 Interrupt Enable.** Value of 1 enables DMA Channel 1 interrupts. Used in conjunction with Local interrupt enable.	P, L	P, L	0
8	**Local DMA Channel 0 Interrupt Enable.** Value of 1 enables DMA Channel 0 interrupts. Used in conjunction with Local interrupt enable. Clearing the DMA status bits also clears the interrupt.	P, L	P, L	0
7	**INTI Interrupt Enable.** Value of 1 enables an interrupt to generate if the INTI pin is asserted.	P, L	P, L	0
6	**Local Parity Error Interrupt Enable.** Value of 1 enables a Local interrupt to generate if a Local Parity error is detected.	P, L	P, L	0
5	**Serial Port Interrupt 2 Enable.** Value of 1 enables a Local interrupt when Serial Port Interrupt 2 is active.	P, L	P, L	0
4	**Serial Port Interrupt 1 Enable.** Value of 1 enables a Local interrupt when Serial Port Interrupt 1 is active.	P, L	P, L	0
3	**Local Timeout Interrupt Enable.** Value of 1 enables a Local interrupt when a Local Bus timeout occurs.	P, L	P, L	0
2	**PCI Bus Parity Error Interrupt Enable.** Value of 1 enables a Local interrupt when a PCI Parity error is detected or the Outbound Free Messaging Queue overflows.	P, L	P, L	0
1	**PCI Abort Interrupt Enable.** Value of 1 enables a Local interrupt output when a Direct Slave bit (PCISR[12]) or Master Abort Status bit (PCISR[13]) is set in the PCI Status Configuration register.	P, L	P, L	0
0	**Local Interrupt Output Enable.** Value of 1 enables Local interrupt output.	P, L	P, L	1

Register 17-123. (PABTADR; PCI:1C0h, LOC:1C0h) PCI Abort Address

Bit	Description	Read	Write	Value after Reset
31:2	**PCI Abort Address.** When a PCI Master/Target abort occurs, the starting address of the current access is returned to this register.	P, L	No	32'h0000_0000
1:0	***Reserved.***	No	No	0

17.8 DMA REGISTERS

Register 17-124. (C0MODE; PCI:200h, LOC:200h) Channel 0 Mode

Bit	Description	Read	Write	Value after Reset
31:21	***Reserved.***	P, L	No	0h
20	**EOT0# End Link.** Used only for DMA Scatter/Gather transfers. When EOT0# is asserted, value of 1 indicates the DMA transfer ends the current Scatter/Gather link and continues with the remaining Scatter/Gather transfers. When EOT0# is asserted, value of 0 indicates the DMA transfer ends the current Scatter/Gather transfer and does not continue with the remaining Scatter/Gather transfers.	P, L	P, L	0
19	**Valid Stop Control.** Value of 0 indicates the DMA Chaining controller continuously polls a descriptor with the Valid bit set to 0 (invalid descriptor) if the Valid Mode Enable bit (C0MODE[18]=1) is set. Value of 1 indicates the Chaining controller stops polling when the Valid bit with a value of 0 is detected (C0COUNT[31]=0). In this case, the CPU must restart the DMA controller by setting the Start bit (C0CSR[1]=1). A pause sets the DMA Done register.	P, L	P, L	0
18	**Valid Mode Enable.** Value of 0 indicates the Valid bit (C0COUNT[31]) is ignored. Value of 1 indicates the DMA descriptors are processed only when the Valid bit is set (C0COUNT[31]). If the Valid bit is set, the transfer count is 0, and the descriptor is not the last descriptor in the chain. The DMA controller then moves to the next descriptor in the chain.	P, L	P, L	0
17	**PCI Dual Address Scatter/Gather.** Value of 0 indicates that normal Address cycles are used during descriptor Scatter/Gather operations. Value of 1 indicates that PCI Dual Address cycles are used during descriptor Scatter/Gather operations (load C0PCIHADR[31:0]).	P, L	P, L	0
16	**DMA Clear Count Mode.** Value of 1 indicates the byte count in each Scatter/Gather descriptor is cleared when the corresponding DMA transfer is complete. If this bit is enabled and Scatter/Gather DMA has been stopped by EOT, the Writeback cycle writes back to the current descriptor the number of bytes remaining to be transferred.	P, L	P, L	0
15	**DMA Fast/Slow Terminate Mode Select.** Value of 0 asserts BLAST# to terminate a DMA transfer. Value of 1 indicates EOT0# asserted or DREQ0# de-asserted during Demand Mode DMA immediately terminates a DMA transfer (with or without BLAST#).	P, L	P, L	0
14	**DMA EOT Enable.** Value of 0 disables EOT0# input pin. Value of 1 enables the EOT0# input pin.	P, L	P, L	0
13	**Write and Invalidate Mode for DMA Transfers.** Value of 1 indicates the IOP 480 performs Write and Invalidate cycles to the PCI Bus. The IOP 480 supports Write and Invalidate sizes of 8 or 16 words. Size is specified in the PCI Cache Line Size register. If a size other than 8 or 16 is specified, the IOP 480 performs Write transfers rather than Write and Invalidate transfers. Transfers must start and end at the cache-line boundaries.	P, L	P, L	0
12	**Demand Mode.** Value of 1 causes the DMA controller to operate in Demand mode. In Demand mode, the DMA controller transfers data when its DREQ0# input is asserted. It asserts DACK0# to indicate the current Local Bus transfer is in response to the DREQ0# input. DMA controller transfers words (32 bits) of data. May result in multiple transfers for an 8- or 16-bit bus.	P, L	P, L	0

Register 17-124. (C0MODE; PCI:200h, LOC:200h) Channel 0 Mode (Continued)

Bit	Description	Read	Write	Value after Reset
11	**Local Addressing Mode.** Value of 0 indicates the Local address is incremented. Value of 1 indicates the Local address MA[17:0] must be held constant.	P, L	P, L	0
10	**Done Interrupt Enable.** Value of 1 enables the DMA Done interrupt. When DMA Clear Count mode (C0MODE[16]=1)is enabled, the interrupt does not occur until byte count is cleared.	P, L	P, L	0
9	**Scatter/Gather.** Value of 0 indicates Block mode is enabled. Value of 1 indicates Scatter/Gather mode is enabled. For Scatter/Gather mode, the DMA source address, destination address, and byte count are loaded from memory into PCI or Local Address spaces.	P, L	P, L	0
8:0	***Reserved.***	P, L	No	0h

Register 17-125. (C0CSR; PCI:204h, LOC:204h) Channel 0 Control/Status

Bit	Description	Read	Write	Value after Reset
31:5	***Reserved.***	P, L	No	0h
4	**DMA Done.** Value of 0 indicates the channel transfer is not complete. Value of 1 indicates the channel transfer is complete, which occurs at the end of a single DMA transfer if Scatter/Gather is disabled (C0MODE[9]=0), or at the completion of a group of Scatter/Gather DMA transfers if Scatter/Gather is enabled (C0MODE[9]=1).	P, L	No	1
3	**Clear Interrupts.** Writing a value of 1 to this bit clears Channel 0 interrupts.	No	P, L/Clr	0
2	**Abort.** Writing a value of 1 to this bit causes the channel to abort the current transfer. The Channel Enable bit must be cleared (C0CSR[0]=0). The DMA Done bit is set when the abort is complete (C0CSR[4]=1).	No	P, L/Set	0
1	**Start.** Writing a value of 1 to this bit causes the channel to start transferring data if the channel is enabled (C0CSR[0]=1).	No	P, L/Set	0
0	**Enable.** Value of 0 disables the channel from starting a DMA transfer and if in the process of transferring data, to suspend the transfer (pause). Value of 1 enables the channel to transfer data.	P, L	P, L	0

Register 17-126. (C0COUNT; PCI:208h, LOC:208h) Channel 0 Transfer Count

Bit	Description	Read	Write	Value after Reset
31	**Valid.** When the Valid Mode bit is enabled (C0MODE[18]=1), indicates the validity of this DMA descriptor.	P, L	P, L	0
30:23	***Reserved.***	P, L	No	0h
22:0	**DMA Transfer Size (Bytes).** Indicates the number of bytes to transfer during a DMA operation. However, if DMA Clear Count Mode is enabled and the DMA channel had not been stopped by EOT, the Writeback cycle clears this register's bits to zero. However, if DMA Clear Count Mode is enabled and the DMA channel had been stopped by EOT, the Writeback cycle writes back the number of bytes remaining to transfer.	P, L	P, L	0h

Register 17-127. (C0PCILADR; PCI:20Ch, LOC:20Ch) Channel 0 PCI Lower Address

Bit	Description	Read	Write	Value after Reset
31:0	**PCI Address Register.** Indicates PCI Memory space location in which the DMA transfers (reads or writes) start.	P, L	P, L	0h

Register 17-128. (C0LOCADR; PCI:210h, LOC:210h) Channel 0 Local Address

Bit	Description	Read	Write	Value after Reset
31:0	**Local Address Register.** Indicates the Local Memory space location in which the DMA transfers (reads or writes) start.	P, L	P, L	0h

Register 17-129. (C0DESCPTR; PCI:214h, LOC:214h) Channel 0 Descriptor Pointer

Bit	Description	Read	Write	Value after Reset
31:4	**Next Descriptor Address.** Quad-word aligned (Bits [3:0] = 0000).	P, L	P, L	0h
3	Direction of Transfer. Value of 1 indicates transfers from the Local Bus to the PCI Bus. Value of 0 indicates transfers from the PCI Bus to the Local Bus.	P, L	P, L	0
2	**Interrupt after Terminal Count.** Value of 1 generates an interrupt after reaching the terminal count for this descriptor. Value of 0 disables interrupts from being generated.	P, L	P, L	0
1	**End of Chain.** Value of 1 indicates end of chain. Value of 0 indicates not end of chain descriptor. (Same as Block mode.)	P, L	P, L	0
0	**Descriptor Location.** Value of 1 indicates PCI Address space. Value of 0 indicates Local Address space.	P, L	P, L	0

Register 17-130. (C0PCIHADR; PCI:218h, LOC:218h) Channel 0 Dual Address Cycle Upper Address

Bit	Description	Read	Write	Value after Reset
31:0	**Dual Address Cycle Upper Address.** Upper 32-bits of the Dual Address Cycle PCI address during DMA Channel 0 cycles. Used for 64-bit PCI addressing on devices that don't support the 64-bit addressing bus (Single Address Cycle), but support 64-bit addressing with an additional Address phase (Dual Address Cycle). DMA Channel 0 Dual Address cycles are enabled when this register is non-zero.	P, L	P, L	0h

Register 17-131. (C0THRES; PCI:21Ch, LOC:21Ch) Channel 0 Threshold

Bit	Description	Read	Write	Value after Reset
31:16	***Reserved.***	P, L	No	0h
15:12	**PCI-to-Local Almost Empty (C0PLAE).** Number of empty entries (minus 1) in the FIFO before requesting the PCI Bus for reads.	P, L	P, L	0h
11:8	**Local-to-PCI Almost Full (C0LPAF).** Number of pairs of full entries (minus 1) in the FIFO before requesting the PCI Bus for writes.	P, L	P, L	0h
7:4	**Local-to-PCI Almost Empty (C0LPAE).** Number of empty entries (minus 1) in the FIFO before requesting the Local Bus for reads. (C0LPAF+1) + (C0LPAE+1) should be ≤ FIFO depth of 32.	P, L	P, L	0h
3:0	**PCI-to-Local Almost Full (C0PLAF).** Number of pairs of full entries (minus 1) in the FIFO before requesting the Local Bus for writes. (C0PLAF+1) + (C0PLAE+1) should be ≤ FIFO depth of 32.	P, L	P, L	0h

Register 17-132. (C1MODE; PCI:220h, LOC:220h) Channel 1 Mode

Bit	Description	Read	Write	Value after Reset
31:21	***Reserved.***	P, L	No	0h
20	**EOT1# End Link.** Used only for DMA Scatter/Gather transfers. When EOT1# is asserted, value of 1 indicates the DMA transfer ends the current Scatter/Gather link and continues with the remaining Scatter/Gather transfers. When EOT1# is asserted, value of 0 indicates the DMA transfer ends the current Scatter/Gather transfer and does not continue with the remaining Scatter/Gather transfers.	P, L	P, L	0
19	**Valid Stop Control.** Value of 0 indicates the DMA Chaining controller continuously polls a descriptor with the Valid bit set to 0 (invalid descriptor) if the Valid Mode Enable bit (C1MODE[18]=1) is set. Value of 1 indicates the Chaining controller stops polling when the Valid bit with a value of 0 is detected (C1COUNT[31]). In this case, the CPU must restart the DMA controller by setting the Start bit (C1CSR[1]=1). A pause sets the DMA Done register.	P, L	P, L	0
18	**Valid Mode Enable.** Value of 0 indicates the Valid bit (C1COUNT[31]) is ignored. Value of 1 indicates that DMA descriptors are processed only when the Valid bit is set (C1COUNT[31]). If the Valid bit is set, the transfer count is 0, and the descriptor is not the last descriptor in the chain. The DMA controller then moves to the next descriptor in the chain.	P, L	P, L	0
17	**PCI Dual Address Scatter/Gather.** Value of 0 indicates that normal Address cycles are used during descriptor Scatter/Gather operations. Value of 1 indicates that PCI Dual Address cycles are used during descriptor Scatter/Gather operations (load C1PCIHADR[31:0]).	P, L	P, L	0
16	**DMA Clear Count Mode.** Value of 1 indicates that byte count in each Scatter/Gather descriptor is cleared when the corresponding DMA transfer is complete. If this bit is enabled and Scatter/Gather DMA had been stopped by EOT, the Writeback cycle writes back to the current descriptor the number of bytes remaining to be transferred.	P, L	P, L	0
15	**DMA Fast/Slow Terminate Mode Select.** Value of 0 asserts BLAST# to terminate a DMA transfer. Value of 1 indicates EOT1# asserted or DREQ1# de-asserted during Demand Mode DMA immediately terminates a DMA transfer (with or without BLAST#).	P, L	P, L	0
14	**DMA EOT Enable.** Value of 0 disables EOT1# input pin. Value of 1 enables EOT1# input pin.	P, L	P, L	0
13	**Write and Invalidate Mode for DMA Transfers.** Value of 1 indicates the IOP 480 performs Write and Invalidate cycles to the PCI Bus. The IOP 480 supports Write and Invalidate sizes of 8 or 16 words. Size is specified in the PCI Cache Line Size register. If a size other than 8 or 16 is specified, the IOP 480 performs Write transfers rather than Write and Invalidate transfers. Transfers must start and end at the cache-line boundaries.	P, L	P, L	0
12	**Demand Mode.** Value of 1 causes DMA controller to operate in Demand mode. In Demand mode, the DMA controller transfers data when its DREQ1# input is asserted. It asserts DACK1# to indicate the current Local Bus transfer is in response to the DREQ1# input. DMA controller transfers words (32 bits) of data. May result in multiple transfers for an 8- or 16-bit bus.	P, L	P, L	0

Register 17-132. (C1MODE; PCI:220h, LOC:220h) Channel 1 Mode (Continued)

Bit	Description	Read	Write	Value after Reset
11	**Local Addressing Mode.** Value of 0 indicates Local Address is incremented. Value of 1 indicates Local Address MA[17:0] must be held constant.	P, L	P, L	0
10	**Done Interrupt Enable.** Value of 1 enables DMA Done interrupt. When DMA Clear Count mode (C1MODE[16]=1) is enabled, the interrupt does not occur until byte count is cleared.	P, L	P, L	0
9	**Scatter/Gather.** Value of 0 indicates Block mode is enabled. Value of 1 indicates Scatter/Gather mode is enabled. For Scatter/Gather mode, the DMA source address, destination address and byte count are loaded from memory in PCI or Local Address spaces.	P, L	P, L	0
8:0	***Reserved.***	P, L	No	0h

Register 17-133. (C1CSR; PCI:224h, LOC:224h) Channel 1 Control/Status

Bit	Description	Read	Write	Value after Reset
31:5	***Reserved.***	P, L	No	0h
4	**DMA Done.** Value of 1 indicates the Channel transfer is complete, which occurs at the end of a single DMA transfer if Scatter/Gather is disabled (C1MODE[9]=1), or at the completion of a group of Scatter/Gather DMA transfers if Scatter/Gather is enabled. Value of 0 indicates Channel transfer is not complete (C1MODE[9]=0).	P, L	No	1
3	**Clear Interrupts.** Writing a 1 to this bit clears Channel 1 interrupts.	No	P, L/Clr	0
2	**Abort.** Writing a 1 to this bit causes the channel to abort the current transfer. The Channel Enable bit must be cleared (C1CSR[0]=0). The DMA Done bit is set when the abort is complete (C1CSR[4]=1).	No	P, L/Set	0
1	**Start.** Value of 1 to this bit causes the channel to start transferring data if the channel is enabled (C1CSR[0]=1).	No	P, L/Set	0
0	**Enable.** Value of 0 disables the channel from starting a DMA transfer and if in the process of transferring data, to suspend the transfer (pause). Value of 1 enables the channel to transfer data.	P, L	P, L	0

Register 17-134. (C1COUNT; PCI:228h, LOC:228h) Channel 1 Transfer Count

Bit	Description	Read	Write	Value after Reset
31	**Valid.** When the valid mode is enabled (C1MODE[18]=1), indicates the validity of this DMA descriptor.	P, L	P, L	0
30:23	***Reserved.***	P, L	No	0h
22:0	**DMA Transfer Size (Bytes).** Indicates number of bytes to transfer during a DMA operation. However, if DMA Clear Count Mode is enabled and the DMA channel had not been stopped by EOT, the Writeback cycle clears this register's bits to zero. However, if DMA Clear Count Mode is enabled and the DMA channel had been stopped by EOT, the Writeback cycle writes back the number of bytes remaining to transfer.	P, L	P, L	0h

Register 17-135. (C1PCILADR; PCI:22Ch, LOC:22Ch) Channel 1 PCI Lower Address

Bit	Description	Read	Write	Value after Reset
31:0	**PCI Address Register.** Indicates the PCI Memory space location in which the DMA transfers (reads or writes) start.	P, L	P, L	0h

Register 17-136. (C1LOCADR; PCI:230h, LOC:230h) Channel 1 Local Address

Bit	Description	Read	Write	Value after Reset
31:0	**Local Address Register.** Indicates the Local Memory space location in which the DMA transfers (reads or writes) start.	P, L	P, L	0h

Register 17-137. (C1DESCPTR; PCI:234h, LOC:234h) Channel 1 Descriptor Pointer

Bit	Description	Read	Write	Value after Reset
31:4	**Next Descriptor Address.** Quad-word aligned (Bits [3:0] = 0000).	P, L	P, L	0h
3	**Direction of Transfer.** Value of 1 indicates transfers from the Local Bus to the PCI Bus. Value of 0 indicates transfers from the PCI Bus to the Local Bus.	P, L	P, L	0
2	**Interrupt after Terminal Count.** Value of 1 generates an interrupt after reaching the terminal count for this descriptor. Value of 0 disables interrupts from being generated.	P, L	P, L	0
1	**End of Chain.** Value of 1 indicates end of chain. Value of 0 indicates not end of chain descriptor. (Same as Block mode.)	P, L	P, L	0
0	**Descriptor Location.** Value of 1 indicates PCI Address space. Value of 0 indicates Local Address space.	P, L	P, L	0

Register 17-138. (C1PCIHADR; PCI:238h, LOC:238h) Channel 1 Dual Address Cycle Upper Address

Bit	Description	Read	Write	Value after Reset
31:0	**Dual Address Cycle Upper Address.** Upper 32-bits of the Dual Address Cycle PCI address during DMA Channel 1 cycles. Used for 64-bit PCI addressing on devices that don't support the 64-bit addressing bus (Single Address cycle), but support 64-bit addressing with an additional Address phase (Dual Address cycle). DMA Channel 1 Dual Address cycles are enabled when this register is non-zero.	P, L	P, L	0h

Register 17-139. (C1THRES; PCI:23Ch, LOC:23Ch) Channel 1 Threshold

Bit	Description	Read	Write	Value after Reset
31:16	***Reserved.***	P, L	No	0h
15:12	**PCI-to-Local Almost Empty (C1PLAE).** Number of empty entries (minus 1) in the FIFO before requesting the PCI Bus for reads.	P, L	P, L	0h
11:8	**Local-to-PCI Almost Full (C1LPAF).** Number of pairs of full entries (minus 1) in the FIFO before requesting the PCI Bus for writes.	P, L	P, L	0h
7:4	**Local-to-PCI Almost Empty (C1LPAE).** Number of empty entries (minus 1) in the FIFO before requesting Local Bus for reads. (C1LPAF+1) + (C1LPAE+1) should be ≤ FIFO depth of 16.	P, L	P, L	0h
3:0	**PCI-to-Local Almost Full (C1PLAF).** Number of pairs of full entries (minus 1) in the FIFO before requesting Local Bus for writes. (C1PLAF+1) + (C1PLAE+1) should be ≤ FIFO depth of 16.	P, L	P, L	0h

Register 17-140. (C2MODE; PCI:240h, LOC:240h) Channel 2 Mode

Bit	Description	Read	Write	Value after Reset
31:16	***Reserved.***	P, L	No	0h
15	**DMA Fast/Slow Terminate Mode Select.** Value of 0 indicates when EOT# is asserted or DREQ2# is de-asserted, the DMA controller completes the current word's transfer and completes the next word's transfer with BLAST# asserted. When set to 1, the DMA controller stops transferring data after the current word is transferred, although BLAST# output is not asserted.	P, L	P, L	0
14	**DMA EOT Enable.** Value of 0 disables the EOT2# input pin. Value of 1 enables the EOT2# input pin.	P, L	P, L	0
13	**Done Interrupt Enable.** Value of 1 enables the DMA Done interrupt.	P, L	P, L	0
12	**Source Demand Mode.** Value of 1 causes the DMA controller to operate in Demand mode while reading Source data. In Demand mode, the DMA controller transfers data when its DREQ2# input is asserted. It asserts DACK2# to indicate the current Local Bus transfer is in response to the DREQ2# input. The DMA controller transfers words (32 bits) of data. May result in multiple transfers for an 8- or 16-bit bus. For Flyby mode, this bit is ignored.	P, L	P, L	0
11	**Destination Demand Mode.** Value of 1 causes the DMA controller to operate in Demand mode while writing Destination data. In Demand mode, the DMA controller transfers data when its DREQ2# input is asserted. It asserts DACK2# to indicate the current Local Bus transfer is in response to the DREQ2# input. The DMA controller transfers words (32 bits) of data. This may result in multiple transfers for an 8- or 16-bit bus. For Flyby mode, this bit is used for reads or writes.	P, L	P, L	0
10	**Flyby Mode.** Value of 0 indicates normal DMA addressing, with the Local Source address used for reads and the Destination address used for writes. Value of 1 indicates that a Flyby DMA is performed, where the destination information is used for reads or writes.	P, L	P, L	0
9	**Flyby Direction.** When Flyby mode is enabled (C2MODE[10]=1), a value of 0 indicates the Destination address is written (C2MODE[10]=1). Value of 1 indicates the Destination address is read. ***Note:*** *Destination address is used for both modes.*	P, L	P, L	0
8	***Reserved.***	P, L	No	0
7	**Local Destination Addressing Mode.** Value of 0 indicates the Local Destination address is incremented. Value of 1 indicates the Local Address MA[17:0] must be held constant. Used during all Flyby mode DMA operations.	P, L	P, L	0
6	**Local Source Addressing Mode.** Value of 0 indicates Local Source address is incremented. Value of 1 indicates Local Address MA[17:0] must be held constant. This bit is ignored during all Flyby mode DMA operations.	P, L	P, L	0
5:0	***Reserved.***	P, L	No	0h

Register 17-141. (C2CSR; PCI:244h, LOC:244h) Channel 2 Control/Status

Bit	Description	Read	Write	Value after Reset
31:5	***Reserved.***	P, L	No	0h
4	**DMA Done.** A value of 1 indicates the channel transfer is complete, which occurs at the end of a single DMA transfer. A value of 0 indicates channel transfer is not complete.	P, L	No	1
3	**Clear Interrupts.** Writing a 1 to this bit clears Channel 2 interrupts.	No	P, L/Clr	0
2	**Abort.** Value of 1 to this bit causes the channel to abort the current transfer. The Channel Enable bit must be cleared (C2CSR[0]=0). The DMA Done bit is set when the abort is complete (C2CSR[4]=1).	No	P, L/Set	0
1	**Start.** Value of 1 causes the channel to start transferring data if the channel is enabled (C2CSR[0]=1).	No	P, L/Set	0
0	**Enable.** Value of 1 enables the channel to transfer data. Value of 0 disables the channel from starting a DMA transfer and if in the process of transferring data suspend the transfer (pause).	P, L	P, L	0

Register 17-142. (C2COUNT; PCI:248h, LOC:248h) Channel 2 Transfer Count

Bit	Description	Read	Write	Value after Reset
31:0	**DMA Transfers Size (Bytes).** Indicates number of bytes to transfer during a DMA operation.	P, L	P, L	0h

Register 17-143. (C2SRCADR; PCI:24Ch, LOC:24Ch) Channel 2 Source Address

Bit	Description	Read	Write	Value after Reset
31:0	**Source Address Register.** Indicates the Local Memory space location in which the DMA transfers (reads or writes) start.	P, L	P, L	0h

Register 17-144. (C2DESTADR; PCI:250h, LOC:250h) Channel 2 Destination Address

Bit	Description	Read	Write	Value after Reset
31:0	**Destination Address Register.** Indicates the Local Memory space location in which the DMA transfers (reads or writes) start.	P, L	P, L	0h

18 IOP 480 PIN DESCRIPTION

18.1 PIN DESCRIPTIONS

Table 18-1. I/O Pin Summary

Functional Block	Number of External I/O		Comments
	PQFP	PBGA	
PCI Bus Controller with I_2O Interface	50	50	PLX PCI 9054 core
PCI Arbiter	6	6	Up to three external PCI Masters
Serial EEPROM Interface	3	3	For controller configuration
Hot Swap	2	2	CompactPCI Hot Swap support
16450 Compatible Serial Port (UART)	2	2	Serial port
Memory Controller	25	25	Supports EDO DRAM/SDRAM devices
IOP 480 CPU Clock and Test/Debug	11	11	IOP 480 CPU reset, clock, test, and debug
Power, Ground, and Unused	43	59	Power and ground pins (PBGA package includes one extra common ground, six additional ground pins, and nine unused pins)
Local Bus Interface (IOP 480 CPU Type)	59	60	Local Bus 32-bit multiplexed interface (PBGA package includes one extra pad for LLOCK# signal)
Local Bus Arbiter	4	4	Up to two external Local Bus Masters
Local Bus Interrupts	3	3	General-purpose Local Bus interrupts
Total	208	225	

Table 18-2 lists the abbreviations used in the tables that follow. Table 18-3 describes the I/O buffer types.

For a visual view of the chip pinout, refer to Section 20, “Package.”

Table 18-2. Pin Type Abbreviations

Abbreviation	Pin Type
I/O	Input and output pin
I	Input-only pin
O	Output-only pin
TS	Three-state pin
OD	Open drain pin
TP	Totem pole pin
STS	Sustained three-state pin, driven high for one CLK before float
PU	Pull-up resistor on pin
PD	Pull-down resistor on pin

Note: *PU or PD indicates that the pin buffer has either an internal pull-up or pull-down resistor. This guarantees only that the internal chip pin signal is being pulled up or pulled down. It does not guarantee that the pin signal external to the chip will be pulled up or down. Therefore, external pull-up or pull-down resistors are recommended, even when internal pull-up or pull-down resistors are provided.*

Table 18-3. I/O Buffer Types

Buffer	Buffer Type
BT520 BT535 BT550 BT565	Local Bus Buffers
BPCI5 BPCI5T BPCI5PU BPCI5PUT	PCI Bus Buffers

Table 18-4. Pin Configuration Control Pins

Pin Name	Pin Configuration	Configuration Register and Bit	Default after Reset
ENUM#	PCI Hot Swap—selectable as input or output	HMODE pin	—
INTA#	PCI Interrupt A—output enable (always input)	PCIIPR[7:0]	Disabled
PME#	PCI Power Management Event—output enable (always input)	PMCSR[8]	Disabled
RST#	PCI Reset—input or output	HMODE pin	—
GNT0#/REQ#	PCI Grant 0—input or output	(determined by PCICTL[16])	Input
	PCI Arbiter Enable	PCICTL[16]	Disabled
REQ0#/GNT#	PCI Request 0—input or output	(determined by PCICTL[16])	Output
	PCI Arbiter Enable	PCICTL[16]	Disabled
GNT2#/TS1	PCI Grant 2 or RISCTrace 1 output	DEVINIT[4]	GNT2
REQ2#/TS2	PCI Request 2 or RISCTrace 2 output	DEVINIT[4]	REQ2#
EEDATA/TS3	Serial EEPROM Data or RISCTrace 3 output	DEVINIT[4]	EEDATA
EESK/TS4	Serial EEPROM Serial Clock or RISCTrace 4 output	DEVINIT[4]	EESK
DP[3:0]/MA[16:13]	Data Parity or Memory Addresses [16:13]	LCSnBRD[12]	Disabled
EOT0#/USER3	DMA End of Transfer or User 3 input	C0MODE[14]	USER3
	USER3 Data (input)	LOCCTL[16]	—
EOT1#/USER4	DMA End of Transfer or User 4 input	C1MODE[14]	USER4
	USER4 Data (input)	LOCCTL[17]	—
LCS0#/DMPAF#	Local Bus Chip Select or Direct Master Programmable Almost Full signal	LOCCTL[0]	LCS0#
	LCS0# Enable	LCS0BASE[0]	Enabled
LCS1#/USER0	Local Bus Chip Select output or USER0 signal	LOCCTL[1]	USER0
	LCS1# Enable	LCS1BASE[0]	Disabled
	USER0 Direction	LOCCTL[2]	Input (pull-up)
	USER0 Data	LOCCTL[3]	0
LCS2#/USER1	Local Bus Chip Select output or USER1 signal	LOCCTL[4]	USER1
	LCS2# Enable	LCS2BASE[0]	Disabled
	USER1 Direction	LOCCTL[5]	Input (pull-up)
	USER1 Data	LOCCTL[6]	0
LCS3#/MA17	Local Bus Chip Select or Memory Address 17	LOCCTL[7]	LCS3#
	LCS3# Enable	LCS3BASE[0]	Disabled
LHOLDACK0	Local Bus Hold Acknowledge 0—input or output	(determined by LARBR[0])	Input
	Local Arbiter Enable	LARBR[0]	Enabled
LHOLDREQ0	Local Bus Hold Request 0—input or output	(determined by LARBR[0])	Output
	Local Arbiter Enable	LARBR[0]	Enabled

Table 18-4. Pin Configuration Control Pins (Continued)

Pin Name	Pin Configuration	Configuration Register and Bit	Default after Reset
CINT/USER2	Critical Interrupt input or USER2 Signal	LOCCTL[8]	USER2
	USER2 Direction	LOCCTL[9]	Input (pull-up)
	USER2 Data	LOCCTL[10]	0
	CINT Polarity	LINTENB[19]	Active Low
INTI	INTI Interrupt Status	LINTSTAT[7]	0
	INTI Polarity	LINTENB[18]	Active Low
INTO	INTO Polarity	LINTENB[17]	Active Low
MCAS#/MOE#	Column Address Strobe or Memory Output Enable	(determined by DRAM Type and DRAM Enable)	MOE#
	DRAM Type	DRAMCTL[0]	EDO
	DRAM Enable	DRAMBASE[0]	Disabled
RAS[3:0]#/MCS[3:0]#	Row Address Strobe or Memory Chip Selects	(determined by DRAM Type and DRAM Enable)	MCSn#
	DRAM Type	DRAMCTL[0]	EDO
	DRAM Enable	DRAMBASE[0]	Disabled
CAS[3:0]#/MDQM[3:0]#	Column Address Strobe or Data Mask Outputs	(determined by DRAM Type and DRAM Enable)	MDQMn#
	DRAM Type	DRAMCTL[0]	EDO
	DRAM Enable	DRAMBASE[0]	Disabled
TX/TS5	Serial Transmit or RISCTrace 5 output	DEVINIT[4]	TX

Note: *The "default value after reset" is valid only if no programmed serial EEPROM is present or until the serial EEPROM has loaded new values to the Configuration registers.*

Table 18-5. PCI Bus Controller with I_2O Interface Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
AD[31:0]	Address and Data	32	I/O TS BPCI5_B and BPCI5T_B	7-11, 13-15, 19-22, 24-25, 27-28, 43-45, 47-51, 55-59, 61-63	C1, D2, G6, E4, D1, E3, E2, F5, F2, F1, G4, G3, G1, H3, H4, H5, K5, M2, L4, M3, N2, K6, P1, N3, R2, P3, L5, N4, R3, K7, M5, R4	Address and Data signals multiplexed on the same PCI pins. A Bus transaction consists of an Address phase followed by one or more Data phases. The IOP 480 supports both Read and Write bursts.
C/BE[3:0]#	Bus Command and Byte Enables	4	I/O TS BPCI5_B	16, 29, 42, 54	E1, J2, M1, M4	All multiplexed on the same PCI pins during the address phase. These signals define the Bus command. During the Data phase they are used as Byte Enables. Refer to *PCI Specification r2.2* for further details.
CLK	Clock	1	I BPCI5_A	3	B1	Provides timing for all transactions on the PCI Bus and is an input to every PCI device.
DEVSEL#	Device Select	1	I/O STS BPCI5_B	34	J5	When actively driven, indicates the driving device has decoded its address of the current access Target. As an input, indicates whether any Bus device is selected.
FRAME#	Cycle Frame	1	I/O STS BPCI5_B	31	J1	Driven by the current Master to indicate the beginning and duration of an access. When asserted, data transfers continue. When de-asserted, the transaction is in the final Data phase.
IDSEL	Initialization Device Select	1	I BPCI5T_A	17	F4	Used as a chip select during configuration Read and Write transactions.
INTA#	Interrupt	1	I/O OD BPCI5_A	65	N5	Used by the IOP 480 to request an interrupt, or as interrupt input to the IOP 480. In Adapter mode, INTA# is an output only. In Host mode, INTA# is an input only.
IRDY#	Initiator Ready	1	I/O STS BPCI5_B	32	J3	Indicates the initiating agent's ability (bus master) to complete the current transaction Data phase.
LOCK#	Lock	1	I/O STS BPCI5T_B	37	K3	Asserted by the master to indicate an atomic operation that may require multiple transactions to the Target agent to complete.

Table 18-5. PCI Bus Controller with I_2O Interface Pins (Continued)

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
PAR	Parity	1	I/O TS BPCI5_B	40	K4	Even parity across AD and C/BE signals. Parity generation is required by all PCI agents. PAR is stable and valid one clock after the Address phase. For Data phases, PAR is stable and valid one clock after either IRDY# is asserted on a Write transaction or TRDY# is asserted on a Read transaction. Once PAR is valid, it remains valid until one clock after the current Data phase completion.
PERR#	Parity Error	1	I/O STS BPCI5_B	38	J6	Reports data parity errors during all PCI transactions, except during a special cycle.
PME#	Power Management Event	1	I/O OD BPCI5T_A	203	B3	Wake up event pin. Can be an output to signal a power management event, or as an input to receive power management events.
RST#	Reset	1	I/O OD, PU BPCI5PU_A	2	D4	Used to bring PCI-specific registers, sequencers, and signals to a consistent state. Based on the HMODE pin. Can be a Reset input or a Reset output.
SERR#	Systems Error	1	I/O OD BPCI5T_B	39	L2	Reports address parity errors, data parity errors on the Special Cycle command, or any other system error where the result will be catastrophic.
STOP#	Stop	1	I/O STS BPCI5T_B	35	K1	Indicates the current target is requesting the master to stop the current transaction.
TRDY#	Target Ready	1	I/O STS BPCI5T_B	33	J4	Indicates the target agent ability (selected device) to complete the current transaction Data phase.
Total		50				

Table 18-6. PCI Arbiter Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
GNT0#/REQ#	Grant 0/Request	1	O TS BPCI5_A	5	E5	When the IOP 480 is a PCI arbiter (refer to the PCICTL register), indicates to the master that bus access is granted. Every master has its own GNT# and REQ# signals. Indicates to the PCI arbiter that Bus use is required when the IOP 480 is not a PCI arbiter.
GNT1#	Grant 1	1	O TS BPCI5T_A	206	C3	When the IOP 480 is a PCI arbiter (refer to the PCICTL register), indicates to the master that Bus access is granted. Every master has its own GNT# and REQ# signals. Not used when the IOP 480 is not a PCI arbiter.
GNT2#/TS1	Grant 2/ RISCTrace 1 Output	1	O TS BPCI5_B	204	F6	When the IOP 480 is a PCI arbiter (refer to the PCICTL register), indicates to the master that Bus access is granted. Every master has its own GNT# and REQ# signals. When enabled for RISCTrace (DEVINIT[4]=1), TS1 is one of six output pins used in IBM RISCTrace mode. (When used for RISCTrace, only two PCI Bus Masters are allowed.) Not used when the IOP 480 is not a PCI arbiter and RISCTrace is not enabled.
REQ0#/GNT#	Request 0/Grant	1	I PU BPCI5PUT_A	4	C2	When the IOP 480 is a PCI arbiter (refer to the PCICTL register), indicates a master requires Bus use. Every master has its own GNT# and REQ# signals. When the IOP 480 is not a PCI arbiter, indicates that Bus access is granted.
REQ1#	Request 1	1	I PU BPCI5PU_A	207	B2	When the IOP 480 is a PCI arbiter (refer to the PCICTL register), indicates a master requires Bus use. Every master has its own GNT# and REQ# signals. Not used when the IOP 480 is not a PCI arbiter.
REQ2#/TS2	Request 2/ RISCTrace 2 Output	1	I/O PU BPCI5PU_A	205	A2	When the IOP 480 is a PCI arbiter (refer to the PCICTL register), indicates that a master requires Bus use. Every master has its own GNT# and REQ# signals. When enabled for RISCTrace (DEVINIT[4]=1), TS2 is one of six output pins used in IBM RISCTrace mode. (When used for RISCTrace, only two PCI Bus Masters are allowed.) Not used when the IOP 480 is not a PCI arbiter and RISCTrace is not enabled.
Total		6				

Table 18-7. Serial EEPROM Interface Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
EECS	Chip Select	1	O TP 5 mA BT565_G_A	194	F7	Chip Select to serial EEPROM.
EEDATA/TS3	Serial EEPROM Data/ RISCTrace 3 Output	1	I/O TP, PU 5 mA BT565PUT_G_A	195	B5	Read/Write Data from or to the serial EEPROM. When enabled for RISCTrace (DEVINIT[4]=1), TS3 is one of six output pins used in IBM RISCTrace mode.
EESK/TS4	Serial Data Clock/ RISCTrace 4 Output	1	O TP 5 mA BT565_G_A	196	D6	Serial EEPROM Clock. When enabled for RISCTrace (DEVINIT[4]=1), TS4 is one of six output pins used in IBM RISCTrace mode.
Total		3				

Table 18-8. Hot Swap Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
ENUM#	Hot Swap	1	I/O OD BPCI5_A	202	C4	Asserted when an adapter has been inserted, or ready to be removed from, a PCI slot. Can be an output to signal a Hot Swap event, or as an input to manage Hot Swapping of other adapter cards (mode is determined by the HMODE pin).
LEDon/LEDin	LED Control Pin	1	I/O TS 24 mA BT520_G_A	198	A7	As an input, monitors the Compact PCI board latch status. As an output, acts as the Hot Swap board indicator LED.
Total		2				

Table 18-9. 16450 Compatible Serial Port Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
RX	Serial Receive	1	I PU BT565PUT_G_A	199	E6	UART Receive signal (TTL levels).
TX/TS5	Serial Transmit/ RISCTrace 5 Output	1	O TP 5 mA BT565_G_A	200	B4	UART Transmit signal (TTL levels). When enabled for RISCTrace (DEVINIT[4]=1), is one of six output pins used in IBM RISCTrace mode.
Total		2				

Table 18-10. Memory Controller Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
MA[12:0]	Memory Address	13	O TP 7 mA BT535_G_B and BT535T_G_B	178-175, 173-169, 167-164	C9, D9, A10, B10, D10, A11, E10, B11, C11, A12, D11, F9, B12	EDO DRAM: 13-bit multiplexed address allows access to 64 MB of EDO DRAM per bank. 8 16 megabits x 4 devices = 64 MB 4 8 megabits x 8 devices = 32 MB 2 4 megabits x 16 devices = 16 MB 8 4 megabits x 4 devices = 16 MB 4 2 megabits x 8 devices = 8 MB 2 1 megabits x 16 devices = 4 MB SDRAM: 12-bit multiplexed address (MA[11:0]) allows access up to 16 MB of SDRAM per bank. 8 4 megabits x 4 devices = 16 MB 4 2 megabits x 8 devices = 8 MB 2 1 megabits x 16 devices = 4 MB 14-bit megabits multiplexed address (MA[13:0]) allows access up to 64 MB of SDRAM per bank. 8 16 megabits x 4 devices = 64 MB 4 8 megabits x 8 devices = 32 MB 2 4Mx16 devices = 16 MB The address also provides the data for the Mode register during initialization. If enabled by bits in the IOP 480 Configuration register, LCS3# becomes MA17, and DP[3:0] becomes MA[16:13] to expand the multiplexed Memory Bus. These signals can also be used with Local Chip Selects (LCS[3:0]#) for accessing SRAM/EPROM and peripherals.
MCAS#/ MOE#	Column Address Strobe/ Output Enable	1	O TP 7 mA BT535_G_B	185	B7	EDO DRAM: Output Enable used to enable the Output buffers during a Read cycle. SDRAM: Column Address Strobe to Memory devices. Used with Local Chip Selects (LCS[3:0]#) for accessing SRAM/EPROM and peripherals.
MCKE	Clock Enable	1	O TP 9 mA BT535_G_C	179	B9	EDO DRAM: Not used. SDRAM: Clock enable to Memory devices (LCLKO is used to clock Memory transactions). May be used to place SDRAM devices in a Reduce Memory Power Consumption mode.

Table 18-10. Memory Controller Pins (Continued)

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
RAS[3:0]#/ MCS[3:0]#	Row Address Strobe/ Memory Chip Selects	4	O TP 7 mA BT550T_G_B and BT550_G_B	184-183, 181-180	E8, D8, C8, A9	EDO DRAM: Row Address Strobe used to strobe the row address. Also part of the CAS-before-RAS refresh sequence. Each RAS enables one of up to four banks of EDO DRAM. SDRAM: Chip Select asserted to initiate Read, Write, and Refresh cycles. Each Chip Select enables one of up to four banks of SDRAM.
CAS[3:0]#/ MDQM[3:0]#	Column Address Strobe/ Data Mask Outputs	4	O TP 7 mA BT550T_G_B and BT550_G_B	193-191, 189	C6, B6, A6, D7	EDO DRAM: Column Address Strobe. Used to strobes the column address into the EDO SDRAM. Each CASn# signal corresponds to one byte in the 32-bit-wide memory. SDRAM: Data Mask outputs to external Memory devices. Byte Input enables during Write cycles and Output enables during Read cycles. The MDQMn# signals can also be used with Local Chip Selects (LCS[3:0]#) as Byte Write enables when accessing SRAM/EPROM and peripherals.
MRAS#	Row Address Strobe	1	O TP 7 mA BT535_G_B	187	A7	EDO DRAM: Not used. SDRAM: Row Address Strobe to Memory devices.
MWE#	Write Enable	1	O TP 9 mA BT535_G_C	188	C7	Asserted for EDO DRAM, SRAM and SDRAM during Write cycles.
Total		25				

Table 18-11. IOP 480 CPU Clock and Test/Debug Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
HALT#	Halt	1	I PU IT5D1PUT_G_A	143	E14	Halt from external debugger, active low.
HMODE	Host Mode	1	I PU BT565PU_G_A	201	A3	Used to select the IOP 480 Host mode. When high, the IOP 480 drives the PCI RST# and ENUM# signals. When low, the IOP 480 is in the Adapter mode and the PCI RST# and ENUM# are inputs.
LCLK	Local Clock Input	1	I IT5T_G_A	141	F13	Processor clock input (33 MHz or 66 MHz) used to drive the IOP 480 CPU and the Local Bus. ***Note:*** *May be used for the PLL reference clock (REFCLK) in future designs. It must be located on a "T" (test) pin and near the VDDA pin.*
MTP	Manufacturing Test Pin	1	I PD IT5TEPDT_G_A	151	C14	For manufacturing testing use (IBM). Must remain open (no connect) or grounded, for normal operation. ***Note:*** *This is a special IBM Test Enable (TE) Receiver buffer.*
RESET#	Reset Input/ Output	1	I/O OD 7 mA BT535_G_B	117	L13	As an input, a logic 0 input placed on this pin for at least two LCLK causes the IOP 480 CPU to begin a system reset and asserts RESET# to a logic 0 (Output mode). As an output, RESET# is asserted for the following conditions: Processor RESET# asserted; PCI Bus RST# asserted; or software reset bit in the DEVINIT register set to 1. The RESET# output pin becomes a logic 0 for 64 (minus 1 or 2) LCLK cycles. ***Notes:****This pin is never an input in* Adapter *mode. Should only be driven by an "Open Drain" device.*
TCK	Test Clock Input	1	I PU BT550PU_G_A	161	E11	TCK is the clock source for the IOP 480 CPU test access port (TAP). The maximum clock rate into the TCK pin is LCLK rate or less than one-half of the LCLK rate.
TDI	Test Data In	1	I IT5RIT_G_A (no internal pull-up)	132	H11	Used to input serial data into the TAP. When the TAP enables this pin, it is sampled on the rising edge of TCK and the data is input to the selected TAP Shift register.
TDO	Test Data Output	1	O 5 mA BT550_G_A	153	B15	Used to transmit data from the IOP 480 CPU TAP. Data from the selected TAP Shift registers is shifted out on TDO.
TMS	Test Mode Select	1	I PU IT5D2PUT_G_A	154	C13	Sampled by TAP on the rising edge of TCK. The TAP state machine uses the TMS pin to determine the mode in which the TAP operates. ***Note:*** *Not used to select JTAG operation.*

Table 18-11. IOP 480 CPU Clock and Test/Debug Pins (Continued)

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
TRST#	Test Reset	1	I PU IT5PUT_G_A	147	F11	The TRST# signal is used by JTAG testers (not required by IBM testers).
TS6	RISCTrace 6 Output	1	O TP 5 mA BT550_G_A	155	B14	When enabled for RISCTrace (DEVINIT[4]=1), is one of six output pins used in IBM RISCTrace mode.
Total		11				

Table 18-12. Power, Ground and Unused Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
VDD	Power (3.3V)	22	VDD	1, 18, 26, 36, 41, 53, 70, 78, 88, 93, 105, 110, 118, 130, 139, 140, 145, 157, 162, 182, 190, 197	A1, A8, A15, C5, C12, E7, E13, F3, F14, F15, H1, H15, K2, L3, L14, M13, N6, N11, P10, R1, R8, R15	Core and I/O power pins.
VSS	Common Grounds	21/22	GND	6, 12, 23, 30, 46, 52, 60, 64, 75, 82, 100, 104, 116, 125, 134, 150, 156, 168, 174, 186, 208	C10, D3, D13, F8, G2, G7-G9, H6-H10, J7-J9, J12, K8, K10, N1, P4, P7	Ground pins.
VSS	Ground (PBGA package only)	6	GND	—	A5, D5, E9, L1, M11, R11	PBGA ground pins.
—	Unused PBGA Pins	9	Unused	—	B8, E12, E15, G5, H2, J11, L7, P2, P8	Available, but unused PBGA pins.
Total		43 (PQFP) 59 (PBGA)				

Table 18-13. Local Bus Interface (IOP 480 CPU Type) Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
ADS#	Address Strobe	1	I/O TS 7 mA BT535_G_B	123	K14	Asserted by the master to indicate a valid address and the start of a new bus access. Asserted for the first clock of a Bus access.
ALE	Address Latch Enable	1	I/O TS, PD 7 mA BT550PD_G_B	149	C15	Asserted by the master during the Address phase and de-asserted before the Data phase. Used to latch the Multiplexed Address/Data Bus (LAD) address portion.
BOFF#	Backoff Request Out	1	O TP 7 mA BT550_G_B	131	H12	Asserted to indicate that the IOP 480 requires the bus to perform a direct PCI-to-Local Bus access while a Direct Master access is pending on the Local Bus. Used with external logic to generate backoff to a Local Bus Master. Its operational parameters are set up through the IOP 480 Configuration registers.
BLAST#	Burst Last	1	I/O TS 9 mA BT550T_G_C	124	K15	Asserted by the master to indicate the last transfer in a bus access.
BTERM#	Bus Terminate	1	I/O TS 7 mA BT550_G_B	113	J10	If BTERM input is enabled through the Configuration registers (DMA Channel x Mode register), the IOP 480 continues to burst until the BTERM# input signal is asserted. If BTERM input is disabled, the IOP 480 bursts up to four Lwords. BTERM# is a Ready input that breaks up a Burst cycle and causes another Address cycle to occur. It is used in conjunction with the PCI IOP 480 Programmable Wait State generator. When BTERM# is an output, it is asserted, along with READY# output, to request the break up of a burst and the start of a new Address cycle (Abort only).
DACK0#	DMA Acknowledge Output Channel 0	1	O TP 7 mA BT550_G_B	133	G14	When a Channel is programmed through the Configuration registers to operate in Demand mode, its DACK output indicates a DMA transfer is being executed.
DACK1#/ DACK2#	DMA Acknowledge Outputs Channels 1 and 2	1	O TP 7 mA BT550_G_B	136	G13	When DMA Channel 1 or 2 is programmed through the Configuration registers to operate in “Demand Flyby” mode, DACK1/ DACK2 output indicates a DMA transfer is being executed. In Flyby mode, DACK indicates that the READY# output corresponds to Memory data being valid.

Table 18-13. Local Bus Interface (IOP 480 CPU Type) Pins (Continued)

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
DP[3:0]/ MA[16:13]	Data Parity/ Memory Address	4	I/O TS 7 mA BT535_G_B	148, 146, 144, 142	D14, D15, F12, G10	Parity is programmable as even or odd for the four bytes lanes on the Local Bus. Parity is checked by the Master during Read cycles and generated by the Master for Write cycles. Also can be programmed to be Memory Address signals. Upon power-up, DP mode is disabled and these MA signals are not driven (the signals must be held high by external pull-up resistors). Only after the MA mode bit has been programmed (via the serial EEPROM or processor), will these MA or DP signals be driven.
DREQ0#	DMA Request Input	1	I BT550_G_A	135	G15	When a Channel is programmed through the Configuration registers to operate in Demand mode, the DREQ0# input serves as a DMA request.
DREQ1#/ DREQ2#	DMA Request Inputs	1	I BT550T_G_A	137	G12	When DMA Channel 1 or 2 is programmed through the configuration registers to operate in Demand Flyby mode, DREQ# input serves as a DMA request for Channel 1 or Channel 2.
EOT0#/ USER3	DMA 0 End of Transfer input/User Input	1	I BT550_G_A	138	G11	When asserted, causes termination of corresponding DMA Channel transfer. Used as a General Purpose Input signal controlled from the IOP 480 LOCCTL[16] register.
EOT1#/ EOT2#/ USER4	DMA 1/2 End of Transfer Input/User Input	1	I BT550_G_A	152	F10	When asserted, causes the termination of the corresponding DMA Channel transfer. Used as a General Purpose Input signal controlled from the IOP 480 C1MODE register.
LAD[31:0]	Address and Data Bus	32	I/O 7 mA BT535_G_B and BT535T_G_B	106, 103-101, 99-94, 92-89, 87-83, 81-79, 77-76, 74-71, 69-66	M12, P14, N13, R14, P13, N12, R13, P12, L10, R12, M10, P11, K9, N10, R10, L9 M9, N9, R9, P9, L8, M8, N8, R7, N7, M7, R6, P6, M6, R5, L6, P5	Multiplexed 32-Bit Address and Data Bus. ***Note:*** *LAD31 corresponds to bit 31 (MSB) and LAD0 corresponds to bit 0 (LSB).*

Table 18-13. Local Bus Interface (IOP 480 CPU Type) Pins (Continued)

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
LBE[3:0]#	Byte Enables	4	I/O TS 7 mA BT535_G_B and BT535T_G_B	108, 109, 111, 112	N14, L11, N15, M14	Encoded Byte Enables. Encoding determined by the Configured Bus width (32-, 16-, or 8-bit). **32-Bit Bus** The four byte enables indicate which of the four bytes are active during a Data cycle: • LBE3# Byte Enable 3—LAD[31:24] • LBE2# Byte Enable 2—LAD[23:16] • LBE1# Byte Enable 1—LAD[15:8] • LBE0# Byte Enable 0—LAD[7:0] **16-Bit Bus** LBE3#, LBE1#, and LBE0# are encoded to provide BHE#, A1, and BLE#, respectively: • LBE3# Byte Enable High (BHE#)—LAD[15:8] • LBE2# Not used • LBE1# Address bit 1 (A1) • LBE0# Byte Enable Low (BLE#)—LAD[7:0] **8-Bit Bus** LBE1# and LBE0# are encoded to provide A1 and A0, respectively: • LBE3# not used • LBE2# not used • LBE1# Address bit 1 (A1) • LBE0# Address bit 0 (A0)
LCS0#/ DMPAF#	Local Bus Chip Select/ Direct Master Programmable Almost Full	1	O TP 7 mA BT550_G_B	158	D12	Local Bus Chip Select 0. Asserted by the IOP 480 when the LAD Bus address has decoded an address in the range programmed for the Chip Select. Supports 8-, 16-, or 32-bit-wide SRAM, ROM/EPROM, or I/O devices. After reset, if no programmed serial EEPROM is found, used to select the 8-bit Boot ROM device (in Big Endian mode). Direct Master Write FIFO Almost Full status output, pin selection programmable through LOCCTL[0].

Table 18-13. Local Bus Interface (IOP 480 CPU Type) Pins (Continued)

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
LCS1#/ USER0	Local Bus Chip Select/ User Input/Output	1	I/O TS 7 mA BT550_G_B	159	A14	Local Bus Chip Select 1. Asserted by the IOP 480 when the LAD Bus address has decoded an address in bit-wide range programmed for the Chip Select. Supports 8-, 16-, or 32-bit-wide SRAM, EPROM, or I/O devices. Used as a General Purpose Input/Output signal controlled from the IOP 480 Configuration registers. Pin selection programmable through LOCCTL[1].
LCS2#/ USER1	Local Bus Chip Select/ User Input/Output	1	I/O TS 7 mA BT550T_G_B	160	B13	Local Bus Chip Select 2. Asserted by the IOP 480 when the LAD Bus address has decoded an address in the range programmed for the Chip Select. Supports 8-, 16-, or 32-bit-wide SRAM, ROM/EPROM, or I/O devices. Used as a General Purpose Input/Output signal controlled from the IOP 480 Configuration registers. Pin selection programmable through LOCCTL[4].
LCS3#/MA17	Local Bus Chip Select 3/ Memory Address 17	1	O TP 7 mA BT535T_G_B	163	A13	Local Bus Chip Select 3. Asserted by the IOP 480 when the LAD Bus address has decoded an address in the range programmed for the Chip Select. Supports 8-, 16-, or 32-bit-wide SRAM, ROM/EPROM, or I/O devices. If enabled by a bit in the Configuration register, becomes Multiplexed Address Bus signal MA17 (refer to the MA[12:0] signal description). Pin selection programmable through LOCCTL[3].
LWR#	Write/Read#	1	I/O TS 7 mA BT550_G_B	107	P15	Master asserts low for reads and high for writes.
LLOCK#	Bus Lock	0/1	I/O TS 7 mA BT550_G_B_PU	—	H14	Available only in PBGA package. Indicates an atomic operation is required by the Local Bus Master. An output when the Direct Slave PCI Bus requires multiple transactions to an External Local Bus device. An input when a Local Bus Master retains bus control.

Table 18-13. Local Bus Interface (IOP 480 CPU Type) Pins (Continued)

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
READY#	Ready	1	I/O TS 7 mA BT550T_G_B	114	L12	Asserted by the Slave to indicate that the Bus Read data is valid or that the Write Data transfer is complete. Wait states are inserted until READY# is asserted. If the READY# input is disabled for a Local Address space, then the number of Wait States is determined by the Internal Wait State generator.
RD#	Read Strobe	1	O TS 7 mA BT550_G_B	120	L15	General Purpose Read strobe asserted by the IOP 480. The timing is controlled by the Bus Region Description registers. Three-stated when the IOP 480 is not the Local Bus Master. Can be used with devices such as external FIFOs.
WAIT#	Wait	1	I/O TS 7 mA BT550_G_B	115	M15	During Direct Master accesses, WAIT# can be used to signal the IOP 480 that the Local Bus Master cannot accept/provide the data, and requires Wait States inserted. When IOP 480 is the Local Bus Master, WAIT# is an output that provides the status of the Internal Wait State generators.
Total		59 (PQFP) 60 (PBGA)				

Table 18-14. Local Bus Arbiter Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
LHOLDACK0/ LDREQ	Local Bus Hold Acknowledge 0/ Local Bus Hold Request	1	O TP 7 mA BT550T_G_B	126	J13	When the IOP 480 is a Local Bus Arbiter (refer to the LARBR register), indicates to the master that bus access is granted (acknowledged). Every master has its own LHOLDACK and LHOLDREQ signals. Asserted when the IOP 480 requests Local Bus use when the IOP 480 is not a Local Bus arbiter.
LHOLDACK1/ BREQ	Local Bus Hold Acknowledge 1/ Local Bus Request (Preempt)	1	O TP 9 mA BT550T_G_C	128	J15	Indicates to the master that access to the bus is granted (acknowledged). Every master has its own LHOLDACK and LHOLDREQ signals. Not used when the IOP 480 is not a Local Bus arbiter (refer to the LARBR register). ***Note:*** *BREQ output is asserted to request an external Local Master to release the Local Bus when there is a Refresh cycle or the Local Latency Timer is expired.*
LHOLDREQ0/ LHOLDACK	Local Bus Hold Request 0/ Local Bus Hold Acknowledge	1	I PD BT550PD_G_A	127	J14	When the IOP 480 is a Local Bus arbiter (refer to the LARBR register), indicates that a master requires Bus use. Every master has its own LHOLDACK and LHOLDREQ signals. Control has been granted to the IOP 480 when it is not a Local Bus arbiter, nor is it asserted by the Local Bus arbiter.
LHOLDREQ1	Local Bus Hold Request 1	1	I PD BT550PD_G_A	129	H13	Indicates that a master requires Bus use. Every master has its own LHOLDACK and LHOLDREQ signals. Not used when the IOP 480 is not a Local Bus arbiter (refer to the LARBR register).
Total		4				

Table 18-15. Local Bus Interrupt Pins

Symbol	Signal Name	Total	Pin Type	PQFP Pin #	PBGA Pin #	Function
CINT/ USER2	Critical Interrupt	1	I/O TS 5 mA BT550T_G_A	119	K11	Used as a General Purpose Input or Output signal or IOP 480 CPU Critical Interrupt source. The CINT polarity is selectable by LINTENB[19]. Pin selection programmable through LOCCTL[8].
INTI	Interrupt Input	1	I BT550T_G_A	121	K12	Source for the IOP 480 CPU Interrupt. The polarity of INTI is selectable by LINTENB[18].
INTO	Interrupt Output	1	O TP 5 mA BT550_G_A	122	K13	IOP 480 interrupt output. The polarity of INTO is selectable by LINTENB[17]. Asserted each time the power state in the PMCSR register changes.
Total		3				

Note: *EDO/SDRAM uses LAD[31:0] for the 32-bit Data Bus.*

19 ELECTRICAL SPECIFICATIONS

19.1 I/O TIMING SPECIFICATIONS

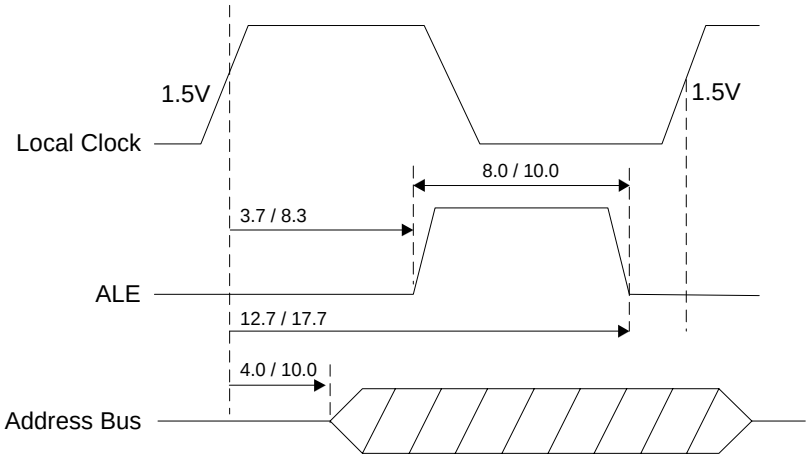


Figure 19-1. IOP 480 ALE Output Delay from the Local Clock (Min/Max, in Nanoseconds)

Table 19-1. AC Electrical Characteristics (Worst Case Process, T_a=85 °C, Vcc=3.0V)

Synchronous Signal	Output Delay		Setup Time	Hold Time
	Min	Max		
PCI GNT# Signals	3.0 ns	12.0 ns	10.0 ns	0.0 ns
PCI REQ# Signals	3.0 ns	12.0 ns	12.0 ns	0.0 ns
All Other PCI Signals	3.0 ns	11.0 ns	7.0 ns	0.0 ns
LADx	3.0 ns	10.0 ns	4.5 ns	1.0 ns
LCSx#_USERx	3.0 ns	10.0 ns	5.0 ns	1.0 ns
READY#	3.0 ns	10.0 ns	5.5 ns	1.0 ns
LWR#	3.0 ns	10.0 ns	10.0 ns	1.0 ns
CAS_MDQMx (IOP 480 as Master)	3.0 ns	8.5 ns	N/A	N/A
CAS_MDQMx (External Local Master)	3.0 ns	12.5 ns	N/A	N/A
All Other Local Pins (Not Serial EEPROM)	3.0 ns	10.0 ns	10.0 ns	1.0 ns
Input Clocks	**Min**		**Max**	
PCI Bus Frequency	0		33 MHz	
Local Bus Frequency	0		66 MHz	

Notes:
Local output delays evaluated with 25 pF loading.
PCI output delays evaluated with 50 pF loading.
Timings may be derated at 0.8 ns per 10 pF.
The Maximum Local Bus Frequency for the IOP 480-AA60BI and IOP 480-AA60PI is 60 MHz.
N/A indicates "Not applicable."

It is recommended that approximately 8 to 16 of either 0.1 µF or 0.01 µF decoupling capacitors be distributed evenly near the IOP 480 power and ground pins.

Table 19-2. IOP 480 Local Bus Driver Loading Derating

Buffer Type	Transition	Worst Case (ps/pF)	Typical (ps/pF)	Best Case (ps/pF)
BT520	t_{PLH}	35	23	17
	t_{PHL}	43	28	20
BT520	t_{PLH}	47	33	26
	t_{PHL}	65	44	35
BT550	t_{PLH}	57	41	34
	t_{PHL}	82	58	48
BT565	t_{PLH}	72	54	45
	t_{PHL}	106	78	66

Note: *Values provided are worst case of performance levels A, B, and C. The differences are very small.*

Table 19-3. IOP 480 PCI Buffer Loading Derating

Buffer Type	Transition	Worst Case (ps/pF)	Typical (ps/pF)	Best Case (ps/pF)
BPCI5	t_{PLH}	40	28	21
	t_{PHL}	44	30	22

Table 19-4. Absolute Maximum Ratings

Specification	Maximum Rating
Storage Temperature	-55 to +125 ºC
Ambient Temperature with Power Applied	-40 to +85 ºC
Supply Voltage to Ground	-0.5 to +4.6V
Input Voltage (VIN)	VSS -0.5V to 11.0V
Output Voltage (VOUT)	VSS -0.5V to VDD +0.5
Maximum Power Dissipation (208-Pin PQFP)	1.0W
Maximum Power Dissipation (225-Pin PBGA)	1.0W

Table 19-5. Operating Ranges

Ambient Temperature	Supply Voltage (VDD)	Input Voltage (VIN)	
		Min	Max
-40 to +85 ºC	3.0 to 3.6V	VSS	VDD

Table 19-6. Capacitance (Sample Tested Only)

Parameter	Test Conditions	Pin Type	Value		Units
			Typical	Maximum	
CIN	VIN = 0V	Input	4	8	pF
COUT	VOUT = 0V	Output	6	12	pF

Table 19-7. Package Thermal Resistance

Package Type	Air Flow			
	0m/s	1m/s	2m/s	3m/s
208-Pin PQFP	65 (C/W)	45	35	30
225-Pin PBGA	72 (C/W)	46	37	32

Table 19-8. Electrical Characteristics over Operating Range

Parameter	Description	Test Conditions		Min	Max	Units
VOH	Output High Voltage for 65-Ohm Driver, 50-Ohm Driver, 35-Ohm Driver, 20-Ohm Driver	VDD = Min VIN = VIH or VIL	IOH =-7.0,-10.0, -13.0, -23.0 mA	2.4	—	V
VOL	Output Low Voltage for 65-Ohm Driver, 50-Ohm Driver, 35-Ohm Driver, 20-Ohm Driver		IOL = 5.0, 7.0, 9.0, 16.0 mA	—	0.4	V
VIH	Input High Level[1]	—	—	2.0	5.5	V
VIL	Input Low Level[1]	—	—	-0.6	0.8	V
VOH3	PCI 3.3V Output High Voltage	VDD = Min VIN = VIH or VIL	IOH = -500uA	0.9 VDD	—	V
VOL3	PCI 3.3V Output Low Voltage		IOL = 1500uA	—	0.1 VDD	V
VIH3	PCI 3.3V Input High Voltage	—	—	0.5 VDD	VDD + 0.5	V
VIL3	PCI 3.3V Input Low Voltage	—	—	-0.5	0.3 VDD	V
IIL	Input Leakage	VSS<=VIN<=VDD, VDD = Max		-10	+10	uA
IIL_Rup	Input Leakage with 10K-Ohm Pull-Up Resistors	VSS<=VIN<=VDD, VDD = Max		-370	+10	uA
IIL_Rdown	Input Leakage with 10K-Ohm Pull-Down Resistors	VSS<=VIN<=VDD, VDD = Max		-10	+370	uA
ILPC	DC Current Per Pin during Precharge[2]	VDD = Max, VP=1.0V ±10%		—	4.0	mA
IOZ	Three-State Output Leakage Current	VDD = Max VSS<=VIN<=VDD		-10	+10	uA
ICC	Power Supply Current	VDD = 3.6V, PCLK = LCLK=33Mhz		—	200	mA
ICCL ICCH ICCZ	Quiescent Power Supply Current	VDD = Max VIN = GND or VDD		—	50	uA

1. VIH and VIL limit for 5V Tolerant Receivers.
2. ILPC is the DC current flowing from VDD to Ground during precharge, as both PMOS and NMOS devices remain on during precharge. It is not the leakage current flowing into or out of the pin under precharge.

20 PACKAGE

20.1 208-PIN PQFP PACKAGE

208-Pin PQFP, Body Size 30.80 x 30.80 mm.

20.1.1 208-Pin PQFP Package Mechanical Specifications

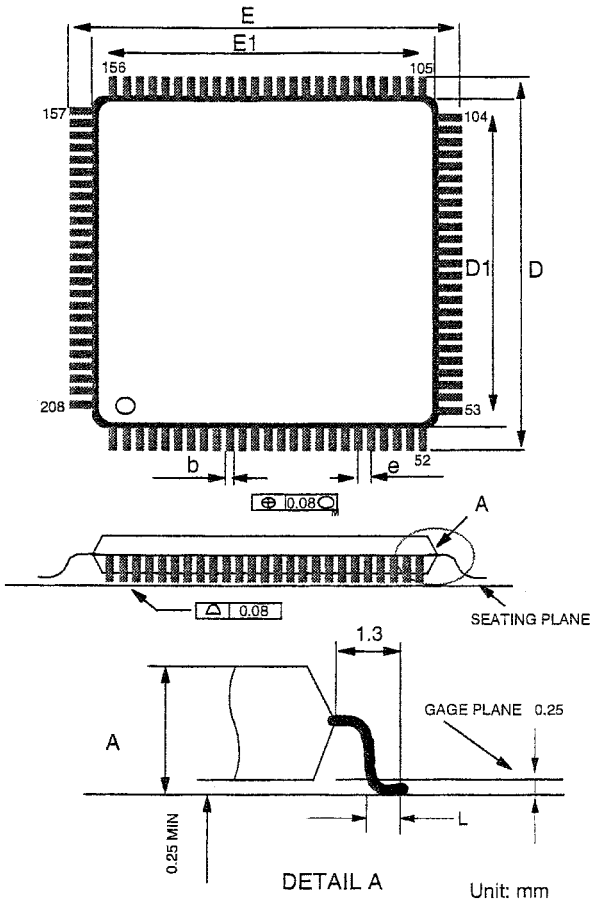


Figure 20-1. 208-Pin PQFP Package Mechanical Specifications

Note: *Drawing is not to scale. See your PLX representative for released documentation.*

Table 20-1. 208-Pin PQFP Mechanical Specifications (Legend for Figure 20-1)

Symbol	Outside Area (mm)	Dimensions	Symbol	Outside Area (mm)	Dimensions
A	Max	4.20	D	Min	30.40
b	Min	0.17		Max	30.80
	Max	0.27	E	Min	30.40
D_1	Min	27.90		Max	30.80
	Max	28.10	e	Nominal	0.50
E_1	Min	27.90	Weight (g)	—	6.00
	Max	28.10	L	Min	0.50
—				Max	0.75

Note: *Metric conversion to inches: x 0.03937. When designing patterns, use only metric dimensions.*

20.1.2 208-Pin PQFP Pinout

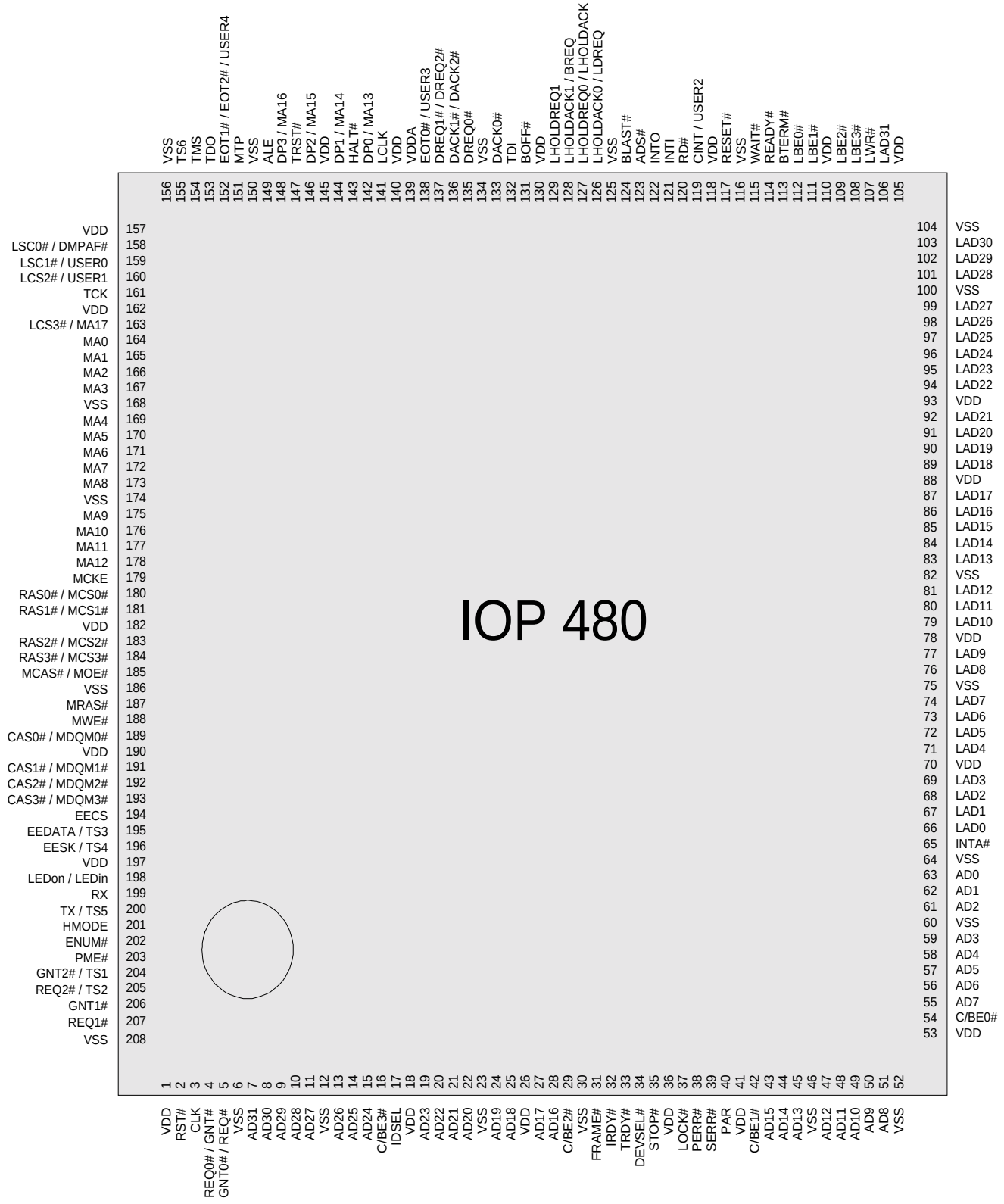


Figure 20-2. 208-Pin PQFP Pinout

20.1.3 208-Pin PQFP Package Materials and Properties

Table 20-2. Package Materials and Properties

Materials	Properties
Conductor	Au wire
Pin frame thermal conductivity (W/m-°C)	155-351
Pin frame thermal expansion Coeff (ppm/°C)	17
Inner lead pitch (µm)	190-220
Maximum wire length (mm)	4.5
Wire diameter (µm)	25-33
Pin frame material	Copper
Pin alloy	85/15 SnPb Typical
Pin pitch (mm)	0.5
Decoupling capacitors	None
Signal Interconnect	Wire bond
Die attachment material	Ag paste
Package encapsulation	Thermosetting epoxy resin (mold compound)
Mold Compound TGE (ppm/°C)	10-18

20.1.4 208-Pin PQFP Printed Circuit Board (PCB) Assembly Compatibility

All CMOS 5 ASIC packaging products are compatible with existing surface-mount assembly tools and procedures. ASIC packages were qualified to use a range of solvents, fluxes, printed circuit board materials, and cross-sections. Typical process conditions are listed in the following table.

Table 20-3. 208-Pin PQFP PCB Assembly Compatibility

Process	Conditions
PCB Assembly	Infrared or Vapor Phase Reflow
PCB Assembly Temp	220°C (VPS) or 235°C (IR)
PCB Clean Process	Aqueous
PCB Rework	Remove / Replace
Max Placement Force	TBD
Bakeout	125°C / 24 hours

20.2 225-PIN PBGA PACKAGE

PBGA 225-pin, body size 27 x 27 x 2.65 mm.

20.2.1 225-PIN PBGA Package Mechanical Specifications

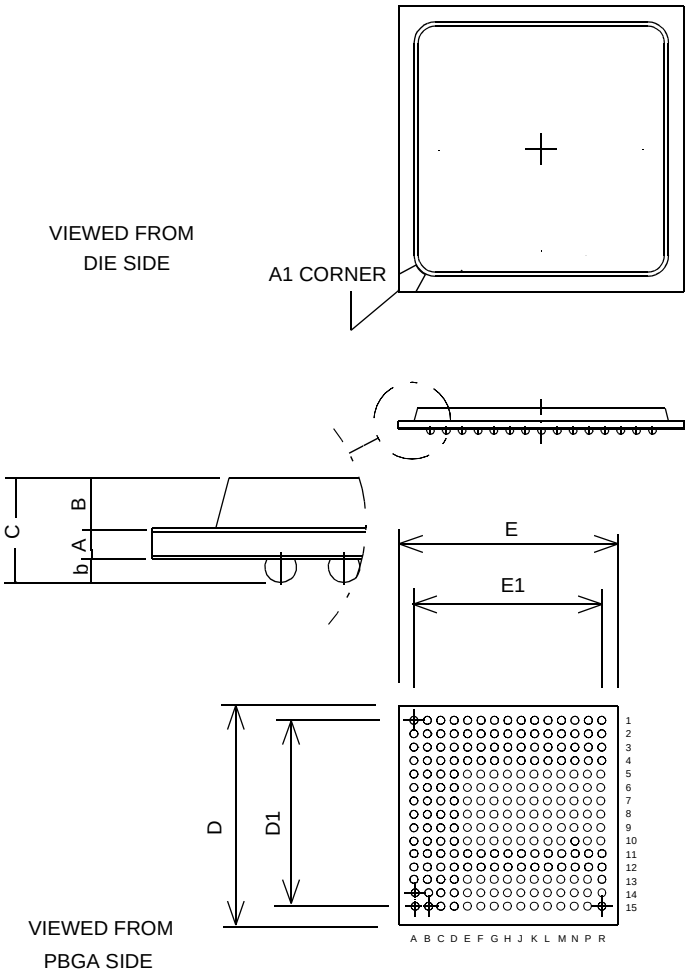


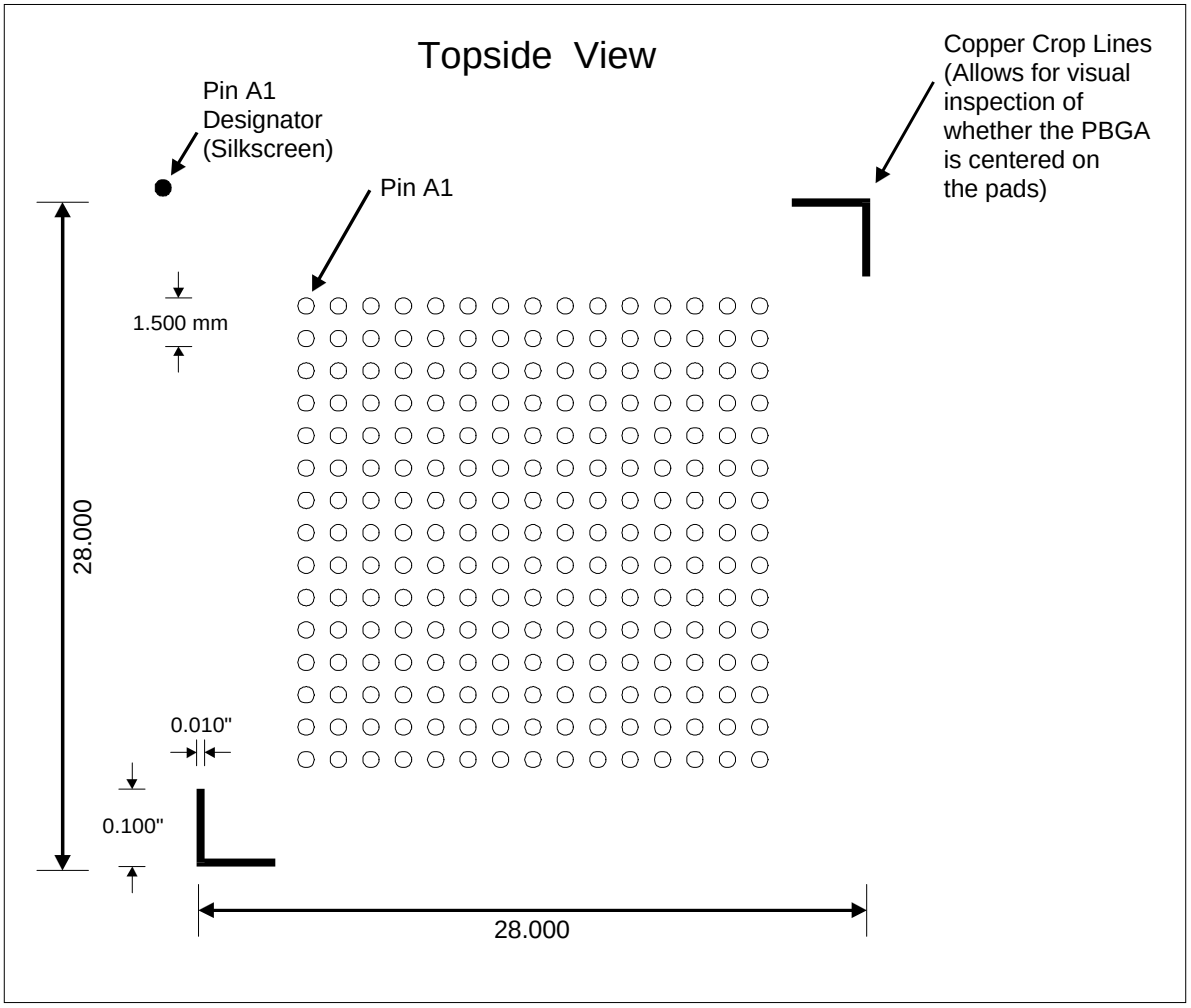
Figure 20-3. 225-Pin PBGA Package Mechanical Specifications

Table 20-4. 225-Pin PBGA Package Mechanical Specifications (Legend for Figure 20-3)

Symbol	Tolerance (+/-)	Dimensions	Symbol	Tolerance (+/-)	Dimensions
A	0.109	0.67	D1 and E1	0.051	21.00
B	0.100	1.20	e	Nominal	1.50
b	0.100	0.60	I/O Matrix	—	15 x 15
C, max	—	2.65	Populated Rows (f)	—	Full
D and E	0.203	27.00	—		

Note: *Metric conversion to inches: x 0.03937.*
When designing patterns, use only metric dimensions.

20.2.2 225-Pin PBGA Suggested Land Pattern for PCB Layout



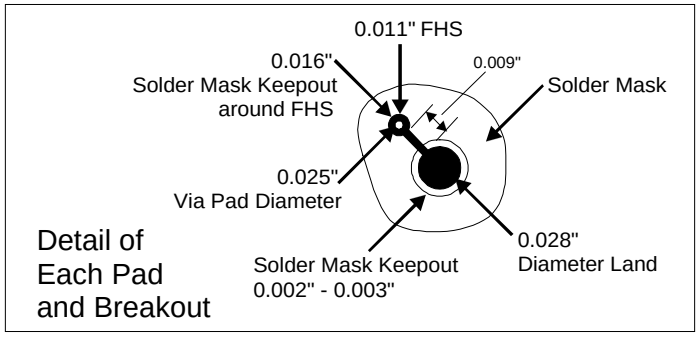


Figure 20-4. 225-Pin PBGA Suggested Land Pattern for PCB Layout

20.2.3 225-Pin PBGA Package Layout

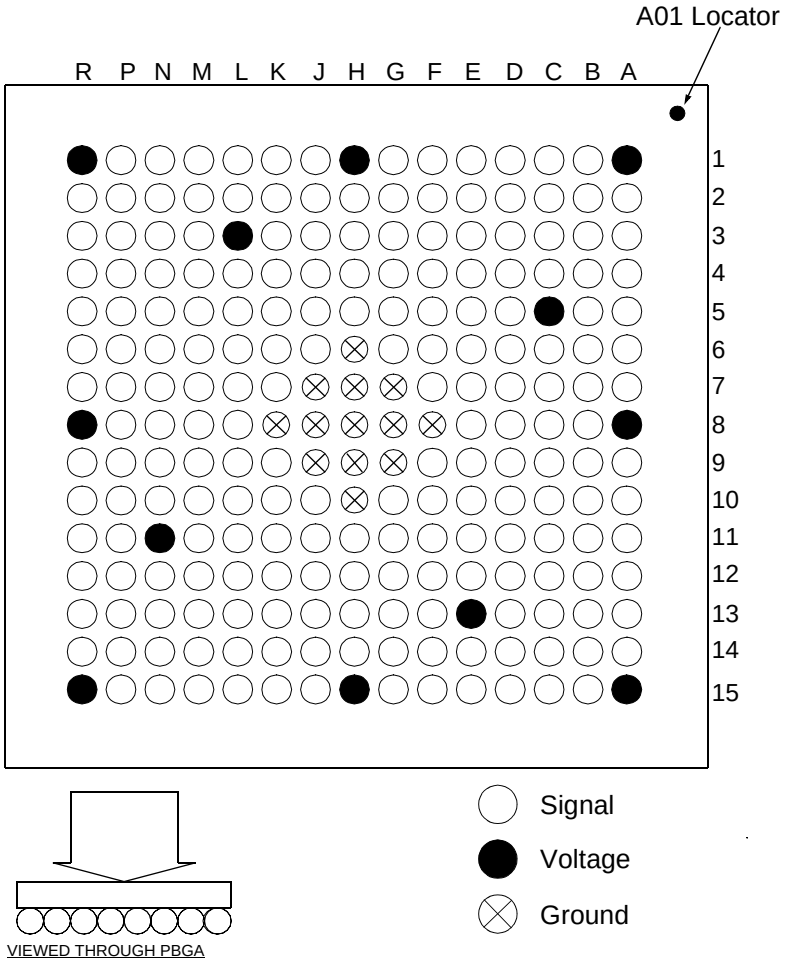


Figure 20-5. 225-Pin PBGA Underside

20.2.4 225-Pin PBGA Pinout

Table 20-5. 225-Pin PBGA Pinout

Pin #	Symbol	Pin #	Symbol	Pin #	Symbol	Pln #	Symbol
A1	VDD	C1	AD31	E1	C/BE3#	G1	AD19
A2	REQ2# / TS2	C2	REQ0# / GNT#	E2	AD25	G2	VSS
A3	HMODE	C3	GNT1#	E3	AD26	G3	AD20
A4	LEDon / LEDin	C4	ENUM#	E4	AD28	G4	AD21
A5	VSS *	C5	VDD	E5	GNT0# / REQ#	G5	NC
A6	CAS1# / MDQM1#	C6	CAS3# / MDQM3#	E6	RX	G6	AD29
A7	MRAS#	C7	MWE#	E7	VDD	G7	VSS
A8	VDD	C8	RAS1# / MCS1#	E8	RAS3# / MCS3#	G8	VSS
A9	RAS0# / MCS0#	C9	MA12	E9	VSS *	G9	VSS
A10	MA10	C10	VSS	E10	MA6	G10	DP0 / MA13
A11	MA7	C11	MA4	E11	TCK	G11	EOT0# / USER3
A12	MA3	C12	VDD	E12	NC	G12	DREQ1# / DREQ2#
A13	LCS3# / MA17	C13	TMS	E13	VDD	G13	DACK1# / DACK2#
A14	LCS1# / USER0	C14	MTP	E14	HALT#	G14	DACK0#
A15	VDD	C15	ALE	E15	NC	G15	DREQ0#
B1	CLK	D1	AD27	F1	AD22	H1	VDD
B2	REQ1#	D2	AD30	F2	AD23	H2	NC
B3	PME#	D3	VSS	F3	VDD	H3	AD18
B4	TX /TS5	D4	RST#	F4	IDSEL	H4	AD17
B5	EEDATA / TS3	D5	VSS *	F5	AD24	H5	AD16
B6	CAS2# / MDQM2#	D6	EESK / TS4	F6	GNT2# / TS1	H6	VSS
B7	MCAS# / MOE#	D7	CAS0# / MDQM0#	F7	EECS	H7	VSS
B8	NC	D8	RAS2# / MCS2#	F8	VSS	H8	VSS
B9	MCKE	D9	MA11	F9	MA1	H9	VSS
B10	MA9	D10	MA8	F10	EOT1# / EOT2# / USER4	H10	VSS
B11	MA5	D11	MA2	F11	TRST#	H11	TDI
B12	MA0	D12	LCS0# / DMPAF#	F12	DP1 / MA14	H12	BOFF#
B13	LCS2# / USER1	D13	VSS	F13	LCLK	H13	LHOLDREQ1
B14	TS6	D14	DP3 / MA16	F14	VDD	H14	LLOCK# *
B15	TDO	D15	DP2 / MA15	F15	VDDA	H15	VDD

Table 20-5. 225-Pin PBGA Pinout (Continued)

Pin #	Symbol	Pin #	Symbol	Pin #	Symbol	Pln #	Symbol
J1	FRAME#	L1	VSS	N1	VSS	R1	VDD
J2	C/BE2#	L2	SERR#	N2	AD11	R2	AD7
J3	IRDY#	L3	VDD	N3	AD8	R3	AD3
J4	TRDY#	L4	AD13	N4	AD4	R4	AD0
J5	DEVSEL#	L5	AD5	N5	INTA#	R5	LAD2
J6	PERR#	L6	LAD1	N6	VDD	R6	LAD5
J7	VSS	L7	NC	N7	LAD7	R7	LAD8
J8	VSS	L8	LAD11	N8	LAD9	R8	VDD
J9	VSS	L9	LAD16	N9	LAD14	R9	LAD13
J10	BTERM#	L10	LAD23	N10	LAD18	R10	LAD17
J11	NC	L11	LBE2#	N11	VDD	R11	VSS *
J12	VSS	L12	READY#	N12	LAD26	R12	LAD22
J13	LHOLDACK0 / LDREQ	L13	RESET#	N13	LAD29	R13	LAD25
J14	LHOLDREQ0 / LHOLDACK	L14	VDD	N14	LBE3#	R14	LAD28
J15	LHOLDACK1 / BREQ	L15	RD#	N15	LBE1#	R15	VDD
K1	STOP#	M1	C/BE1#	P1	AD9	—	
K2	VDD	M2	AD14	P2	NC		
K3	LOCK#	M3	AD12	P3	AD6		
K4	PAR	M4	C/BE0#	P4	VSS		
K5	AD15	M5	AD1	P5	LAD0		
K6	AD10	M6	LAD3	P6	LAD4		
K7	AD2	M7	LAD6	P7	VSS		
K8	VSS	M8	LAD10	P8	NC		
K9	LAD19	M9	LAD15	P9	LAD12		
K10	VSS	M10	LAD21	P10	VDD		
K11	CINT / USER2	M11	VSS *	P11	LAD20		
K12	INTI	M12	LAD31	P12	LAD24		
K13	INTO	M13	VDD	P13	LAD27		
K14	ADS#	M14	LBE0#	P14	LAD30		
K15	BLAST#	M15	WAIT#	P15	LWR#		

Note: ** PBGA package only.*

20.2.5 225-Pin PBGA Package Materials and Properties

Table 20-6. PBGA Package Materials/Properties

Materials	Properties
Conductor	Copper
Dielectric	BT epoxy
Dielectric constant	4.1-4.3 at 1 MHz
Thermal conductivity (W/m-°C)	0.40
Thermal expansion coeff (ppm/°C)	40-50 Z-directions 10-15 XY-directions
Nom line width (mm)	0.102
Nom line spacing (mm)	0.102
Line thickness (mm)	0.033
Dielectric thickness / layer (mm)	0.52
Pin (ball) alloy	63/37 SnPb
Pin (ball) pitch (mm) (225 pins)	1.5
Pin (ball) diameter (mm)	0.75
Discrete decoupling capacitor	None
Die interconnect	Wire bond
Package Encapsulation	Epoxy overmold

20.2.6 225-Pin PBGA Printed Circuit Board (PCB) Assembly Compatibility

All CMOS 5 ASICs packaging products are compatible with existing surface mount assembly tools and procedures. Our packages were qualified to use a range of fluxes, printed circuit board materials, and cross-sections. Typical process conditions are listed in the following table.

Table 20-7. PBGA Typical Process Conditions

Process	Conditions
PCB Assembly	Infrared Reflow
Max PCB Assembly Temp	220 °C
PCB Clean Process	Aqueous
PCB Rework	Remove / Replace

Single-sided card assembly is qualified. Contact your PLX representative regarding dual-sided card assembly.

20.2.7 225-Pin PBGA Die Attach

- Wire bond

20.2.8 225-Pin PBGA Encapsulation

- Molded resin

21 TIMING DIAGRAMS REFERENCE LIST

Sections in which IOP 480 timing diagrams can be found are listed in the following table, with a reference page to the beginning of each section. For a complete catalog of IOP 480 timing diagrams, refer to the List of Timing Diagrams on page xxxiii.

Note: *Each section describes a specific operation.*

Table 21-1. IOP 480 Timing Diagram Sections

Section	Page
IOP 480 CPU Bootup Cycle	6-3
VPD	15-3
Big Endian	2-14
PCI Signals	3-2
Configuration Cycle PCI Configuration Reads or Writes PCI Memory Reads or Writes Local Master Configuration Reads or Writes	 4-12 4-14 5-10
Direct Slave	4-16
Direct Slave Burst	4-27
Direct Master PCI Configuration Reads or Writes Direct Master Operation	 5-8 5-10
DMA	7-14
I_2O	13-7
Local Bus Controller Wait States	2-7
Local Bus Controller Direct Slave or DMA Burst	2-9
Local Bus Controller Arbiter Round-Robin Arbitration	8-2
Memory Controller SRAM Write Access	12-6
Memory Controller SRAM Read Access	12-10
Memory Controller SDRAM Initialization, Refresh, Start, and End	12-20
Memory Controller SDRAM Write Access	12-23
Memory Controller SDRAM Read Access	12-33
Memory Controller EDO DRAM Refresh	12-43
Memory Controller EDO Write Access	12-44
Memory Controller EDO Read Access	12-49

22 SERIAL PORT OPERATION

This section describes the IOP 480 asynchronous serial port unit (SPU). Included is information about the serial port hardware elements, operations and operating modes, and details of registers and buffers.

The SPU can be accessed only from the IOP 480 CPU, although it uses Local Memory space.

22.1 OVERVIEW

The SPU runs at speeds up to one-sixteenth of the external (LCLK) clock rate and provides features typically found on advanced serial communications controllers, including internal loopback mode, automatic echo mode, and automatic handshaking capability on receive operations.

The SPU comprises three main elements—receiver, transmitter, and baud rate generator (Figure 22-1).

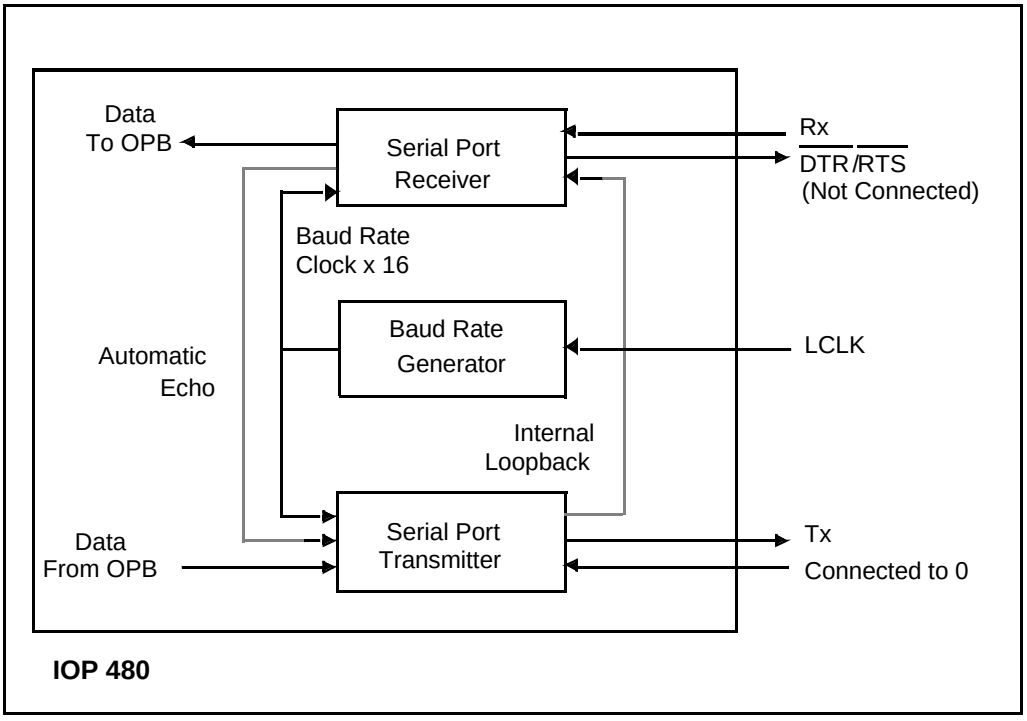


Figure 22-1. Serial Port Functional Block

22.2 SPU OPERATING MODE SELECTION

The SPU can be configured to operate in Normal, Loopback, or Automatic Echo mode by setting the Loopback Mode (LM) bits in the Serial Port Control (SPCTL) register, as shown in Table 22-1.

Table 22-1. SPU Operating Mode Selection

Mode	SPCTL Register Bit Settings
Normal	LM = 00
Internal Loopback	LM = 01
Automatic Echo	LM = 10

22.2.1 Normal Mode

In Normal mode, the SPU performs as a standard UART. Bytes of data are received and transmitted by way of the Rx and Tx lines, respectively.

22.2.2 Internal Loopback Mode

In Internal Loopback mode, data being transmitted is internally routed to the receiver.

Note: *In this operation mode, both the transmitter and receiver must be enabled.*

22.2.3 Automatic Echo Mode

In Automatic Echo mode, as each bit of data is received on the Rx input, it is retransmitted on the Tx output with a delay of one baud rate clock cycle. Thus, the transmitter is acting in a “Pass-Through” mode, implying that any errors in the data stream are also transmitted as they are seen. All handshaking signals operate as they do in Normal mode.

Note: *In this operation mode, the receiver must be enabled and the transmitter must be disabled.*

22.3 SPU REGISTERS

Serial Port registers are memory-mapped and are therefore accessed via load/store instructions at the address of the register. They can only be accessed by the internal IOP 480 CPU. Because the SPU is a byte device, its registers are eight bits and are usually accessed using load byte and store byte instructions. Load/Store Word and Lword instructions can also access the serial port registers. When these instructions are used, the Bus Interface Unit (BIU) breaks register accesses into discrete byte operations.

Table 22-2 lists the Memory-Mapped addresses for the user-accessible registers in the serial port.

Note: *Although the SPRB and SPTB are logically separate registers, they are mapped to the same physical address.*

Figure 22-2 represents the serial port registers as they relate to the individual components within the SPU.

The Serial Port Control Register (SPCTL) controls the overall operation of the serial port. It provides the data frame format to the transmitter and receiver (number of Data/Stop bits and parity generation/detection), controls the state of the $\overline{\text{DTR}}$ and $\overline{\text{RTS}}$ signals (should always be inactive for the IOP 480), and specifies the SPU mode of operation (Normal, Internal Loopback, Auto Echo).

The Line Status register (SPLS) and Handshake Status register (SPHS) gather status information for the serial port. The SPLS contains status information for the receiver (RxReady, Parity, Framing and Overrun Errors, and Line Break Detection) and the transmitter (TBR and TSRE). The SPHS contains the status of the input handshake signals ($\overline{\text{CTS}}$/$\overline{\text{DSR}}$ Inactive).

The Baud Rate Generator contains the two baud rate divisor registers, BRDH and BRDL. Although both baud rate registers are 8-bit registers, only the least significant four bits of the BRDH are implemented. (If this is true, then the slowest baud rate for the external 66 MHz clock is greater than 1000). The contents of these registers are concatenated to form a 12-bit divisor which is used to derive the frequency at which to transmit and receive data through the SPU.

Table 22-2. Serial Port Register Addresses, Names, and Access Modes

Memory-Mapped I/O Address	Name	Mnemonic	Access Modes
0x4000 0000	Serial Port Line Status Register (Clear)	SPLS	Read/Write
0x4000 0004	Serial Port Line Status Register (Set)	—	—
0x4000 0008	Serial Port Handshake Status Register	SPHS	Read/Write
0x4000 000C	***Reserved***	—	—
0x4000 0010	Baud Rate Divisor High Register	BRDH	Read/Write
0x4000 0014	Baud Rate Divisor Low Register	BRDL	Read/Write
0x4000 0018	Serial Port Control Register	SPCTL	Read/Write
0x4000 001C	Serial Port Receiver Command Register	SPRC	Read/Write
0x4000 0020	Serial Port Transmitter Command Register	SPTC	Read/Write
0x4000 0024	Serial Port Receive/Transmit Buffer	SPRB/SPTB	SPRB is read only SPTB is write only

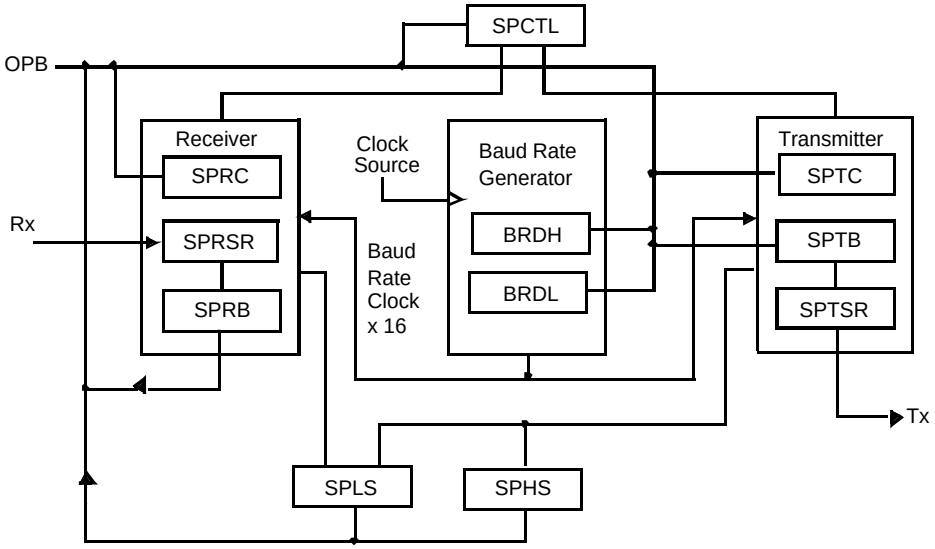


Figure 22-2. SPU Registers and Buffers

The Serial Port Transmitter (SPT) consists of an 8-bit Transmit buffer (SPTB), a parallel-to-serial shift register (SPTSR), and a command register (SPTC). The SPTC contains control information specific to the transmitter, such as the transmitter enable, interrupt enables, line break generation enable, pattern generation mode enable, and auto-handshaking enable (Stop/Pause on $\overline{CTS}$ input). Data to be transmitted is loaded into the SPTB until the data is transferred into the SPTSR when transmission begins. The transmitter reports SPTSR and SPTB full/empty status, reflected as TSRE and TBR, respectively in the SPLS register. The transmitter also provides the status of the handshaking inputs for the SPHS register.

The Serial Port Receiver (SPR) consists of an 8-bit Receive buffer (SPRB), a serial-to-parallel shift register (SPRSR), and a command register (SPRC). The SPRC contains control information specific to the receiver, such as the receiver enable, interrupt enables, and auto-handshaking enable (hardware/software control of the $\overline{RTS}$ output). When a data byte has been received by the SPRSR, it is transferred to the SPRB, provided that no errors were detected, and then SPRB full status (reflected in RxReady) is reported to the SPLS register. Data characters that contain errors are discarded, and the error condition (framing, parity, overrun or line breaks) is posted in the SPLS register.

For a complete bit description of these registers, refer to Section 22.5, “SPU Register Descriptions.”

22.4 SPU OPERATIONS

22.4.1 SPU Baud Rate Generator

The frequency used to transmit and receive data through the SPU is derived from the Baud Rate Generator Clock input and the contents of the Baud Rate Divisor registers. The Baud Rate Generator Clock input comes from the external system clock (LCLK).

The Baud Rate Generator divides the rate of the Baud Rate Generator Clock by the value obtained by the concatenated value of the two Baud Rate Divisor registers, BRDH and BRDL, resulting in a clock frequency 16 times faster than the desired baud rate. As shown in Figure 22-1, the output from the baud rate generator drives the transmitter and receiver baud rate inputs. Those inputs are then divided by 16 internally to the receiver and the transmitter to generate the required Baud Rate Clock frequency.

The baud rate for the serial port can be calculated with the following equation.

$$\text{Baud Rate} = \frac{1}{16} \times \frac{\text{LCLK Frequency (in Hz)}}{(\text{BRDH} \,||\, \text{BRDL})}$$

Equation 22-1. Calculating the Serial Port Baud Rate

Alternatively, the contents of BRDH and BRDL can be calculated from the baud rate with the following equation.

$$(\text{BRDH, BRDL}) = \left(\frac{1}{16} \times \frac{\text{LCLK}}{\text{Baud Rate}} \right)$$

Equation 22-2. Calculating the Contents of BRDH and BRDL

Table 22-3 shows selected contents of the Baud Rate Divisor registers and the resulting baud rates when a 66 MHz clock input is applied to the LCLK input. (Refer to Equation 22-2.)

Table 22-3. Baud Rate Divisor Selection

BRDH Contents	BRDL Contents	Baud Rate
0x35	0x6C	1200
0x0D	0xB5	2400
0x06	0x5A	4800
0x03	0xAC	9600
0x01	0xD5	19200
0x00	0x6A	38400

22.4.2 SPU Transmitter

As mentioned previously, the main components of the SPT are an 8-bit Transmit buffer (SPTB), a parallel-to-serial shift register (SPTSR), and a command register (SPTC). To enable the transmitter, the ET bit of the SPTC must be set to 1. If ET is set to 0, the transmitter is disabled, no data is transmitted, and no interrupts ($\overline{DSR}$ inactive, $\overline{CTS}$ inactive, Transmit Buffer Ready, or Transmitter Shift Register Empty) are requested. If ET is reset to 0 during transmission, the transmission immediately terminates and the Tx output is driven to 1.

Transmit Buffer Ready (TBR) and Transmit Shift Register Empty (TSRE) are the only two bits in the SPLS register that reflect transmitter status (represented by bits TBR and TSR, respectively). The following table describes what the possible combinations of these status bits represent.

Table 22-4. TBR/TSRE Status Representation

TBR	TSRE	Description
1	1	Transmitter is idle; no data in Transmit buffer. ***Note:*** *This is the transmitter status out of reset.*
0	1	Transmitter is idle; data was either just loaded into Transmit buffer, or is being held there due to handshaking line loss.
1	0	Transmitter is transmitting a character; no new data loaded into Transmit buffer.
0	0	Transmitter is transmitting a character; next character waiting in Transmit buffer.

Data to be transmitted is first loaded into the SPTB; TBR updates from a logic 1 to a logic 0. If the transmitter is enabled and the SPTSR is empty (TSRE = 1), then the data is immediately transferred to the SPTSR, causing TSRE to update from a logic 1 to a logic 0 and TBR to return to a value of logic 1, indicating that the SPTB is ready to be loaded with another piece of data. When a piece of data is transferred to the SPTSR, the Start, Stop, and Parity bits are added, and the transmission of the character begins immediately.

Note: *For 7-data bit frame formats, the least significant seven bits of the SPTB are transmitted, least significant bit first.*

While the transmitter is shifting out the current piece of data, the SPTB may be loaded with the next piece of data. When transmission of the current piece of data completes, the next piece of data is transferred to the SPTSR and the process repeats.

Note: *TSRE stays at a value of logic 0 until either a handshaking error is flagged or no other characters are loaded into the SPTB.*

22.4.2.1 Pattern Generation Mode

Pattern Generation mode (PGM) is enabled by the PGM bit in the SPTC, SPTC[PGM] = 1. When PGM is enabled, the SPT operates in basically the same way as when PGM is disabled, except that the generation of Start and Stop bits during character transmissions are suppressed. Thus, the SPU essentially becomes a pattern generator, where only the seven or eight Data bits and Parity bit (if enabled) will be transmitted. This allows the SPT to be used for pulse width modulation, with duty-cycle variation controlled by frame size, baud rate, and data pattern.

22.4.2.2 Transmitter Line Break Generation

Setting the Transmit Break (TB) bit of the SPTC to a logic 1 forces a continuous stream of zeros on the Tx output, assuming that the transmitter is enabled. The SPU transmitter will continue to transmit break characters until the TB bit is reset to a logic 0.

Setting the TB bit to 1 should only be done when the SPT is idle. Forcing the transmission of break characters when the SPT is active has undefined results.

22.4.2.3 Transmitter Interrupts

The SPT interrupts can be generated from one of the following three conditions. Each of these conditions has separate interrupt enables in the SPTC.

- TxReady (TBR) is active in the SPLS register
- TSRE is active in the SPLS register
- Handshaking error (DIS or CIS) is flagged in the SPHS register

An interrupt is generated whenever the SPTB is ready to be loaded with new data. The TIE bit of the SPTC is the enable for generating interrupts based on TSRE. When TSRE and TIE are both set to 1, the SPU will raise the transmitter interrupt request to the asynchronous interrupt controller unit to indicate that character transmission has completed.

When a transmitter interrupt request to the asynchronous interrupt controller occurs, the Serial Port Transmitter Interrupt Status (STIS) bit of the External Interrupt Status register (EXISR) is set.

Note: *The exception is processed only if the Serial Port Transmitter Interrupt Enable (STIE) bit of the External Interrupt Enable Register (EXIER) and the External Interrupt Enable (EE) bit of the Machine State register (MSR) are also set.*

22.4.3 SPU Receiver

The main components of the SPU receiver are an eight-bit Receive buffer (SPRB), a serial-to-parallel shift register (SPRSR), and a command register (SPRC). To enable the receiver, the ER bit of the

SPRC must be set to one. When the SPR is enabled, data received from the Rx pin is input to the SPRSR. If a character is received without errors, the character is transferred from the SPRSR to the SPRB, and the RxReady (RBR) bit in the SPLS is set. If the data frame is configured for seven bits, the most significant bit of the byte loaded from the SPRB is set to 0 and the received data occupies the rest of the byte. The RBR bit of the SPLS register is reset to 0 when the data byte in the SPRB is read by the processor.

If errors are detected during the reception of a character, then the error condition (Parity Error, Framing Error, Overrun Error or Line Break Detect) is posted in the SPLS register and the data byte discarded. Parity Errors are detected when the received parity of a data frame does not match the expected parity of the data frame. This causes the Parity Error (PE) bit of the SPLS register to be set. Framing errors imply that one or both of the first Stop bits of the received data frame were a zero. When this error is detected, the Framing Error (FE) bit of the SPLS register is set.

A line break is detected when a character is received and all of the bits of the frame, including Data, Parity, and Stop bits are zeros. This condition will cause the Line Break (LB) bit of the SPLS to be set to 1.

Notes: *Framing errors are also flagged when a line break is detected.*

Line breaks are detected only on character boundaries.

If a line break begins within a character, a framing error is detected. If the line break condition persists for the time equivalent to receiving another character, the line break error is detected. After a line break is detected, the SPR begins searching for the end of the line break, defined as a period of logic equivalent to the time required to receive one data bit of a frame. When the end of the line break is detected, receiver operation continues normally.

There are two cases where the SPRSR receives a character without errors but that character is not transferred to the SPRB and is instead held in the SPRSR. The first is when the SPRB is full, and the second is if there are any receiver errors (PE, FE, OE, LB) posted in the SPLS register. The character being held in the SPRSR is not moved into the SPRB until application software either removes the character currently in the SPRB by performing a read operation, or clears the error condition(s) in the SPLS register.

Errors are cleared from the SPLS register by writing a logic 1 to the appropriate bit. An Overrun Error occurs when the SPRSR is holding a valid character (it cannot move into the SPRB due to the conditions mentioned) and another character starts to be received. In this event, the character in the SPRSR is overwritten with the new character and the Overrun Error (OE) bit in the SPLS register is set.

22.4.3.1 Receiver Interrupts

The SPR interrupts are generated based on two conditions: RxReady (RBR) being active, or any of the receiver errors (PE, FE, OE, LB) being active in the SPLS register. Both of these conditions have separate interrupt enables in the SPRC register.

An interrupt request is sent to the asynchronous interrupt control unit whenever the SPRB has received a new character. Similarly, the Error Interrupt Enable (EIE) bit of the SPRC register enables an interrupt request whenever a framing, parity, overrun error, or line break is detected.

When a receiver interrupt request to the asynchronous interrupt controller occurs, the Serial Port Receiver Interrupt Status (SRIS) bit of the External Interrupt Status Register (EXISR) is set. However, the exception is processed only if the Serial Port Receiver Interrupt Enable (SRIE) bit of the External Interrupt Enable Register (EXIER) and the External Interrupt Enable (EE) bit of the Machine State Register (MSR) are also set.

22.5 SPU REGISTER DESCRIPTIONS

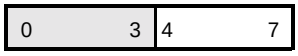
The following sections provide a complete bit description of the nine addressable SPU registers.

22.5.1 Baud Rate Divisor Registers

The BRDH and BRDL registers store a divisor for reducing the rate of the input clock source to a convenient baud rate. The contents of the BRDH are concatenated with the contents of the BRDL to form a 12-bit divisor. The baud rate calculation using the 12-bit divisor is described in Section 22.4.1, “SPU Baud Rate Generator.”

Register 22-1 illustrates the BRDH bits. Register 22-2 illustrates the BRDL bits.

Register 22-1. Baud Rate Divisor High Register (BRDH)



0:3		***Reserved***	
4:7		Divisor High Bits	The four most significant bits of the baud rate divisor; concatenated with the contents of the BRDL.

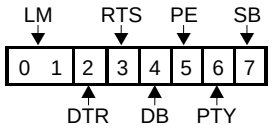
Register 22-2. Baud Rate Divisor Low Register (BRDL)

0:7		Divisor Low Bits	The eight least significant bits of the baud rate divisor; concatenated with the contents of the BRDH.

22.5.2 Serial Port Control Register (SPCTL)

The SPCTL configures the serial port for Normal, Internal Loopback, or Automatic Echo mode. This register also controls the activity of the $\overline{DTR}$/$\overline{RTS}$ signal and the frame format of the serial transfers. Register 22-3 illustrates the SPCTL bits.

Register 22-3. Serial Port Control Register (SPCTL)

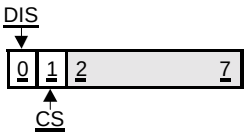


0:1	LM	Loopback Modes 00 - Normal mode 01 - Internal Loopback mode 10 - Automatic Echo mode 11 - ***Reserved***	
2	DTR	Data Terminal Ready 0 - DTR signal is inactive 1 - DTR signal is active	
3	RTS	Request To Send 0 - RTS signal is inactive 1 - RTS is active	
4	DB	Data Bits 0 - 7 Data bits 1 - 8 Data bits	When DB = 0, a data frame contains the least significant seven bits (bits 1:7) in the SPTB or the SPRB.
5	PE	Parity Enable 0 - No parity 1 - Parity enabled	When PE = 0, parity detection and generation are disabled for the serial port receiver and transmitter.
6	PTY	Parity 0 - Even parity 1 - Odd parity	When PTY = 0, even parity is used in parity detection and generation. When PTY = 1, odd parity is used.
7	SB	Stop Bits 0 - One stop bit 1 - Two stop bits	When SB = 0, one stop bit is transmitted at the end of each data frame. When SB = 1, two stop bits are transmitted. In either case, the receiver only checks the first stop bit to detect the end of a received data frame.

22.5.3 Serial Port Handshake Status Register (SPHS)

The SPHS reports the status of the $\overline{DSR}$ or $\overline{CTS}$ signal while the SPT is enabled. Register 22-4 illustrates the SPHS bits.

Register 22-4. Serial Port Handshake Register (SPHS)

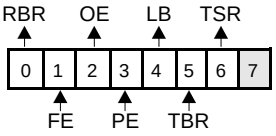


0	DIS	$\overline{DSR}$ Input Inactive Error: 0 - $\overline{DSR}$ input is active 1 - $\overline{DSR}$ input has gone inactive	To reset the DIS bit, the application software must store a 1 in this bit location at the address of the SPHS.
1	CS	$\overline{CTS}$ Input Inactive Error: 0 - $\overline{CTS}$ input is active 1 - $\overline{CTS}$ input has gone inactive	To reset the CS bit, the application software must store a 1 in this bit location at the address of the SPHS.
2:7		***Reserved***	

22.5.4 Serial Port Line Status Register (SPLS)

The SPLS reflects the status of the SPR and SPT, and it reports errors detected in a received character. The bits in the SPLS are reset to 0 by writing a 1 to the bit positions using a store instruction. Writing a 0 to a bit position does not affect the bit value. Register 22-5 illustrates the SPLS bits.

Register 22-5. Serial Port Line Status Register (SPLS)



0	RBR	Receive Buffer Ready 0 - Receive buffer is not full 1 - Receive buffer is full	Reset by hardware when received data is read from the SPRB into a GPR using a load instruction or during chip reset; can be reset by software.
1	FE	Framing Error 0 - No framing error detected 1 - Framing error detected	Must be reset by software.
2	OE	Overrun Error 0 - No overrun error detected 1 - Overrun error detected	Must be reset by software.
3	PE	Parity Error 0 - No parity error detected 1 - Parity error detected	Must be reset by software.
4	LB	Line Break 0 - No line break detected 1 - Line break detected	Must be reset by software.
5	TBR	Transmit Buffer Ready 0 - Transmit buffer is full (not ready) 1 - Transmit buffer is empty and ready	TBR is set to 1 whenever the SPTSR is loaded with a character from the SPTB. TBR is reset to 0 when a new character is stored in the SPTB.
6	TSR	Transmitter Shift Register Ready 0 - Transmitter Shift Register is full 1 - Transmitter Shift Register is empty	TSR is set to 1 whenever the SPTSR is empty. TSR is reset to 0 when a new character is transferred from the SPTB into the SPTSR and remains reset as characters are transmitted.
7		***Reserved***	

22.5.5 Serial Port Receive Buffer (SPRB)

Register 22-6 illustrates the SPRB bits.

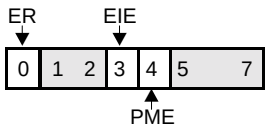
Register 22-6. Serial Port Receive Buffer (SPRB)

0:7		Received Data

22.5.6 Serial Port Receiver Command Register (SPRC)

The SPRC controls the SPR. Register 22-7 illustrates the SPRC bits.

Register 22-7. Serial Port Receiver Command Register (SPRC)



0	ER	Enable Receiver 0 - Disable receiver 1 - Enable receiver	For the SPR to operate, ER must be set to 1. If ER is reset to 0, the SPR is disabled, no data is shifted into the SPRSR, and no SPR interrupts are active.
1	IE	RBR Interrupt Enable 0 - Disable RBR interrupt 1 - Enable RBR interrupt	
2		***Reserved***	
3	EIE	Error Interrupt Enable 0 - Receiver error interrupt disabled 1 - Receiver error interrupts enabled	
4	PME	Pause Mode Enable 0 - $\overline{RTS}$ is controlled by software 1 - $\overline{RTS}$ is controlled by hardware	
5:7		***Reserved***	

22.5.7 Serial Port Transmit Buffer (SPTB)

The SPTB holds characters to be transmitted using the SPTSR. If the SPT is enabled and the SPTSR is empty, data loaded into the SPTB is loaded into the SPTSR and transmitted.

When data is loaded from the SPTB into the SPTSR, then SPLS[TBR] is set to 1 and SPLS[TSR] is reset to 0.

Register 22-8 illustrates the SPTB bits.

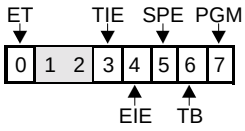
Register 22-8. Serial Port Transmit Buffer (SPTB)

0:7		Transmit Data	Data to be transmitted by the SPTSR

22.6 SERIAL PORT TRANSMIT COMMAND REGISTER (SPTC)

Register 22-9 illustrates the SPTC bits.

Register 22-9. Serial Port Transmitter Command Register (SPTC)



0	ET	Enable Transmitter: 0 - Disable transmitter 1 - Enable transmitter	Chip Reset or System Reset clears to 0.
1	IE	TBR Interrupt Enable 0 - Disable TBR interrupt 1 - Enable TBR interrupt	
2		***Reserved***	
3	TIE	Transmitter Empty Interrupt Enable 0 - Transmitter shift register empty interrupt disabled 1 - Transmitter shift register empty interrupt enabled	
4	EIE	Transmitter Error Interrupt Enable: 0 - Transmitter shift register error interrupt disabled 1 - Transmitter shift register error interrupt enabled	

Register 22-9. Serial Port Transmitter Command Register (SPTC) (Continued)

5	SPE	Stop/Pause on $\overline{CTS}$ Inactive 0 - Pause mode when $\overline{CTS}$ is inactive 1 - Stop mode when $\overline{CTS}$ is inactive	
6	TB	Transmit Break 0 - Disable break character generation 1 - Enable break character generation	
7	PGM	Pattern Generation Mode 0 - Disable pattern generation 1 - Enable pattern generation	Chip Reset or System Reset clears to 0.

22.7 INITIALIZATION AND CONFIGURATION

Prior to enabling SPU operation, SPU configuration registers have to be written with appropriate initial settings. SPU status registers have to be cleared to a clean initial idle state. Following is a sample sequence for writing or clearing the SPU registers:

- Baud Rate Divisor High register
- Baud Rate Divisor Low register
- Line Status register
- Handshake Status register
- Control register
- Receiver Command register
- Transmitter Command register

The line status (SPLS) register is functionally two locations that either clear or set bits in the SPLS. The “0” location is written with zeros to clear (reset) selected bits in the SPLS, and the “1” location is written with ones to set selected bits in the SPLS. All register offsets are shown in Table 22-2.

During initialization the Transmit Buffer Ready (TBR) bit in the Line Status register should be set to 1, or the invalid contents of the Transmit buffer will be shifted into the transmit shift register and sent out as soon as transmitter operation is enabled.

The handshake status (SPHS) register is also functionally two locations that either clear or set bits in the SPHS. The “2” location is written with zeros to clear (reset) selected bits in the SPLS, and the “3” location is written with ones to set selected bits in the SPLS. Only these two status registers are organized in this manner.

22.7.1 Initializing SPU Registers

Exact SPU configuration settings vary, depending on specific operating system and application requirements affecting SPU speed and operating mode. For that reason, the following pseudo code samples present a generalized sequence for configuring the SPU status and control registers and for initializing normal operation. **The following code is shown for example purposes only; however, the first seven commands must be implemented exactly as written, with the exact same values, and in the exact same order (**refer to the assembly code example provided in Appendix C, “Real Code Example”).

```
/* */
/* Initialize and configure the Serial Port
 */
/**/
stb(BRDH_addr, 0); /* clear Serial Port
 baud rate divisor high */
stb(BRDL_addr, 0); /* clearSerial Port baud
 rate divisor low */
stb(SPLS_addr, 0x78); /* clear error bits
 in Serial Port line status */
stb(SPHS_addr, 0xff); /* clear error bits
 in Serial Port handshake status*/
stb(SPCTL_addr, 0); /* clear Serial Port
 control register */
stb(SPRC_addr, 0); /* clear Serial Port
 receiver command register */
stb(SPTC_addr, 0); /* clear Serial Port
 transmitter command register*/
stb(BRDH_addr, baud_rate_high); /* program
 desired baud rate divisor, high */
stb(BRDL_addr, baud_rate_low); /* program
 desired baud rate divisor, low */
stb(SPCTL_addr, serial_port_config);/*
 configure serial port */
/* transmitter and receiver may be */
/* enabled any time after an interrupt
 service */
/* routine is available */
```

Table 22-5 lists the parameters that must be initialized in the Control register.

Table 22-5. Initialized Control Register Parameters

Parameter	Description
Mode	SPU set for normal operation.
Data length	Either seven or eight bits.
Stop bits	Either one or two bits.
Parity	Either disabled or enabled. If parity is enabled, whether odd or even parity.

22.7.2 Enabling Normal SPU Operation

SPU data transfers may be handled either in Polled mode or with normal interrupt processing. In Polled mode, the Receive Buffer Ready (RBR) bit and the pair of Transmit Buffer Ready (TBR) Transmit Shift Register Ready (TSR) bits may be read to determine the status of the Receive and Transmit buffers.
In Interrupt mode, the Interrupt Control bits in the SPRC and the SPTB must be set to enable or disable interrupt generation from the Receive or Transmit buffers.

In Polled mode, the RBR bit in the Line Status register must be tested periodically to determine when the Receive buffer is full. When RBR = 1, the Receive buffer is full and should be read. Prior to reading the byte from the buffer, test the Framing Error bit and the Overrun Error bit to determine whether errors were detected during the transfer.

Similarly, the TBR bit in the Line Status register must be tested periodically to determine when the Transmit buffer is empty. When TBR = 1, the Transmit buffer is ready to be written with another byte to be transmitted.

In Interrupt mode, the receiver or the transmitter sets an interrupt to indicate that an interrupt service routine (ISR) needs to read or write the receive/Transmit buffer. The Receiver Interrupt and Receiver Interrupt Error bits are set in the SPRC to enable receiver interrupt processing. The Transmitter Interrupt and Transmitter Error bits are set in the SPTC to enable transmitter interrupt processing.

In either Polled or Interrupt mode, the Receiver Enable bit SPRC[ER] or the Transmitter Enable bit SPTC[TE] is set to enable either unit to begin processing.
In Interrupt mode, the machine state register bit MSR[EE] must also be set to enable interrupts from the SPU so that normal SPU interrupt service routine processing can begin.

```
/* */
/* Enable Serial Port Operation and
 Interrupt Processing*/
/**/
stb(SPCTL_addr, 0x0C); /* enable normal
 Serial Port operating mode */
stb(SPRC_addr, 0xA0); /* enable Serial Port
 receiver and interrupt */
stb(SPTC_addr, 0x9A); /* enable Serial Port
 transmitter and interrupt */
lwz(GPRn, 0x00029030);/* load MSR
 configuration into GPRn */
mtmsr(GPRn);/* enable MSR exception
 handling*/
```

The SPU reads and writes are processed as loads and stores between GPRs in the IOP 480 CPU and the SPU Receive/Transmit buffer. When an RBR interrupt is generated, the associated ISR is called to test Error Status bits and load the byte from the Receive buffer to a GPR. Conversely, when a TBR interrupt is generated, an ISR is called to store a byte from a GPR to the Transmit buffer. Interrupt processing continues as long as a unit and its interrupts are enabled.

By contrast, the rate at which polling is scheduled depends on the selected SPU operating speed. The SPU polling routine can be configured to execute often enough to assure that the Receive buffer is not overrun or that the Transmit buffer does not go empty as long as data is available to process.

Error handlers are also required to handle anomalies in SPU operations, especially considering that serial link speeds vary, affecting intercharacter timing, and links can time out altogether. Error handlers can be written to handle many such anomalies, as well as processing interrupts from framing errors, Receive buffer overruns, and other SPU errors.

23 IOP 480 CPU OVERVIEW

The IOP 480 incorporates the PowerPC, 32-bit reduced instruction set computer (RISC) embedded controller core, and in this document is referred to as the IOP 480 CPU, which implements PowerPC Architecture with extensions for embedded applications.

This section describes:

- IOP 480 CPU features
- Layered PowerPC Architecture
- IOP 480 CPU implementation of IBM PowerPC Embedded Environment, an extension of the PowerPC Architecture for embedded applications
- IOP 480 CPU organization, including a block diagram and descriptions of the functional units
- IOP 480 CPU registers
- IOP 480 CPU addressing modes

23.1 FEATURES

The IOP 480 CPU 32-bit RISC-embedded controller core provides high performance and low power consumption. The IOP 480 CPU RISC executes at sustained speeds approaching one cycle per instruction.

The IOP 480 CPU features follow.

23.1.1 PowerPC RISC Fixed-Point CPU

- Thirty-two, 32-bit general purpose registers (GPRs)
- Branch prediction
- Single-cycle execution for most instructions
- Hardware multiply/divide for faster integer arithmetic
- Enhanced string and multiple-word handling
- Progammable Interval Timer (PIT), Fixed Interval Timer (FIT), and watchdog timer
- Support for trace and trace-back

23.1.2 Storage Control

- Separate, configurable instruction and data cache units
- Supports 4 KB and 2 KB instruction and data cache arrays
- Flush queue for fill-first operations during cache misses
- Operand forwarding during cache line fills

23.1.3 Memory Management

- Translation of the 4-GB logical address space into physical addresses
- Independent enabling of instruction and data translation/protection
- Page level access control using the translation mechanism
- Software control of page replacement strategy
- Additional control over protection using zones
- WIKGE (write-through, cacheability, compressed, guarded, Endian) storage attributes
- Core interfaces that support a wide range of function and performance:
- Separate 32-bit instruction and data interfaces to the processor local bus (PLB)
- Clock and power management
- JTAG debug interface

23.2 POWERPC ARCHITECTURE

The PowerPC Architecture comprises three levels of standards:

- PowerPC User Instruction Set Architecture, including the base user-level instruction set, user-level registers, programming model, data types, and addressing modes. This is referred to as Book I of the PowerPC Architecture.
- PowerPC Virtual Environment Architecture, describing the memory model, cache model, cache-control instructions, address aliasing, and related issues. While accessible from the user level, these features are intended to be accessed from within library routines provided by the system software. This is referred to as Book II of the PowerPC Architecture.
- PowerPC Operating Environment Architecture, including the memory management model, supervisor-level registers, and the exception model. These features are not accessible from the user level. This is referred to as Book III of the PowerPC Architecture.

Book I and Book II define instructions and facilities available to the application programmer. Book III defines features, such as system-level instructions, that are not directly accessible by user applications.

The PowerPC Architecture guarantees application code compatibility across all PowerPC implementations to help maximize the cross-platform portability of applications developed for PowerPC processors. This is accomplished through compliance with the first level of architectural standard, the PowerPC User Instruction Set Architecture, which is common for all PowerPC implementations.

23.3 IOP 480 CPU AS POWERPC IMPLEMENTATION

The IOP 480 CPU implements the PowerPC User Instruction Set Architecture, user-level registers, programming model, data types, and addressing modes for 32-bit fixed-point operations. The IOP 480 CPU fully complies with specifications for 32-bit implementations of the PowerPC User Instruction Set Architecture. The 64-bit operations are not supported, nor are the floating point operations. Such operations are trapped and can be emulated in software.

Most of the architected features of the IOP 480 CPU are compatible with the specifications for the PowerPC Virtual Environment and Operating Environment Architectures, as specified for processors such as the 6xx family of PowerPC processors. The IOP 480 CPU also provides a number of optimizations and extensions to the lower layers of the PowerPC Architecture. The full architecture of the IOP 480 CPU is defined by the PowerPC Embedded Environment and the PowerPC User Instruction Set Architecture.

The primary extensions of the PowerPC Architecture defined in the Embedded Environment are:

- A simplified memory management mechanism with enhancements for embedded applications
- An enhanced, dual-level interrupt structure
- The addition of several instructions to support these modified and extended resources

Finally, some of the specific implementation features of the IOP 480 CPU are beyond the scope of PowerPC Architecture. These features are included to enhance performance, integrate functionality, and reduce system complexity in embedded control applications.

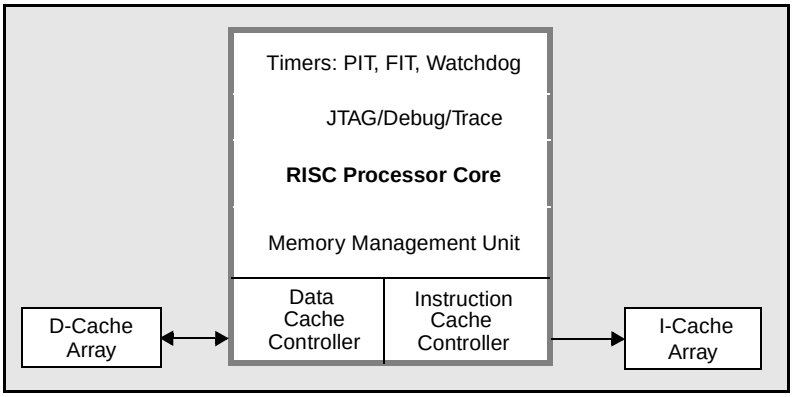


Figure 23-1. IOP 480 CPU Block Diagram

23.4 IOP 480 CPU ORGANIZATION

The IOP 480 CPU consists of a three-stage pipelined processor core, memory management unit (MMU), separate instruction and data cache units, JTAG, debug, trace logic, and three timers. The PowerPC User Instruction Set Architecture and special purpose registers (SPRs) provide a high degree of user control over configuration and operation of the functional units, both interface and core.

Figure 23-1 illustrates the logical organization of the IOP 480 CPU.

23.4.1 RISC Processor Core

The RISC processor core comprises a three-stage instruction pipeline with fetch, decode, and execute stages.

The fetcher provides an instruction stream to the execution unit (EXU). The fetcher speculatively requests up to two instructions from the instruction cache unit (ICU), using two prefetch buffers (PFBs) to queue incoming instructions when the EXU is busy. If the EXU is not busy, instructions from the ICU are forwarded directly to the decode stage. To reduce external bus contention, fetching is suspended when the PFBs are full or requests to fill available positions are pending.

To save area, the pre-fetch buffers do not have dedicated address registers. Because branches are only examined in decode, addresses associated with the pre-fetch buffers can be calculated from the internal decode address register as pre-fetched instructions move into the decode stage. When a branch is predicted taken, the next sequential instruction (NSI), if available, is saved in PFB 0. Because a branch predicted taken, but determined to be not taken, does not need to refetch the NSI, instruction fetch latency is reduced.

The fetcher uses an instruction bit to predict the direction of a conditional branch. If an unresolved conditional branch is encountered, the fetcher speculatively fetches along the predicted address path until the conditional branch is resolved. If the branch was incorrectly predicted, the fetcher aborts all instruction requests made down the mispredicted path. Aborts from the fetcher can propagate through the ICU to terminate requests to the BIU.

The IOP 480 CPU EXU is optimized for minimal area and power dissipation, yet completes most instructions in one cycle. The EXU supports aligned and unaligned storage accesses. Hardware multiply and divide is implemented using existing data flow; the APU interface can be used to attach hardware units to accelerate these operations.

The EXU provides a 32 x 32 register file with two read ports and one write port. Although store instructions require three operands, and the load with update instructions require two write ports for single-cycle performance, significant area was saved by the reducing the register ports and the associated dataflow, resulting in 2-cycle loads and stores.

When the EXU is presented with a load or store instruction, the EXU decomposes the instruction into pseudo operations. All load and store operations, including strings and multiples, share the same structure. This simplifies the control logic. The control logic to support unaligned accesses was a natural

extension of the byte steering logic required to handle string operations combined with the similar structure created by the pseudo operations. Such support for unaligned storage accesses can eliminate gaps, traditionally found in data structures where data must be aligned, to reduce system memory size.

An adjunct register improves the performance of string and unaligned storage accesses, and enables the data cache unit (DCU) to handle 1-, 2-, 3-, or 4-byte requests. The 1-, 2-, and 3-byte requests allow the EXU to align to an Lword boundary in one transfer. Once on an Lword boundary, Lword requests are made until the byte count is exhausted, or until the transfer is completed by a 1-, 2-, or 3-byte transfer. The adjunct register collects the unaligned data for load/store operations to allow Lword accesses to be made to the DCU.

The EXU uses one carry-skip adder for all arithmetic operations, including multiply/divide, and effective address calculation. The adjunct register stores the multiplier and dividend for multiply and divide operations.

23.4.2 Instruction and Data Cache Controllers

The IOP 480 CPU core provides an instruction cache unit (ICU) and a data cache unit (DCU) that allow concurrent accesses and minimize pipeline stalls. Both cache units are two-way set associative, use a 16-byte line size, and provide array built-in self test (ABIST) for manufacturing. The instruction set provides a rich assortment of cache control instructions, including instructions to read tag information and data arrays. See Section 25, “IOP 480 CPU Cache Operations,” for detailed information about the ICU and DCU.

The cache units, optimized for minimal size and power consumption, maintain high performance.

23.4.2.1 Instruction Cache Unit (ICU)

The ICU provides one instruction per cycle to the EXU over a 32-bit bus. A line buffer (built into the output of the array for manufacturing test) enables the ICU to be accessed only once for every four instructions, to reduce power consumption by the array.

The ICU can forward any or all of the four Lwords of a line fill to the EXU to minimize pipeline stalls caused by cache misses. The ICU aborts speculative fetches abandoned by the EXU, eliminating unnecessary line fills and enabling the ICU to handle the next EXU fetch. Aborting abandoned requests also eliminates unnecessary external bus activity to increase external bus utilization.

Cache line locking can completely eliminate ICU misses in critical code. Cache line locking can be performed line by line, and is controlled by the Cache Debug Control Register (CDBCR) and PowerPC 4xx instructions.

23.4.2.2 Data Cache Unit (DCU)

The DCU transfers one, two, three, or four bytes per cycle, depending on the number of byte enables presented by the CPU. The DCU contains a single-element command and store data queue to reduce pipeline stalls; this queue enables the DCU to independently process load/store and cache control instructions. Dynamic PLB request prioritization reduces pipeline stalls even further. When the DCU is busy with a low-priority request while a subsequent storage operation requested by the CPU is stalled, the DCU automatically increases the priority of the current request to the PLB.

The DCU uses a two-line flush queue to minimize pipeline stalls caused by cache misses. Line flushes are postponed until after a line fill is completed. Registers comprise the first position of the flush queue; the line buffer built into the output of the array for manufacturing test serves as the second position of the flush queue. Pipeline stalls are further reduced by forwarding the requested Lword to the CPU during the line fill. Single-queued flushes are non-blocking. When a flush operation is pending, the DCU can continue to access the array to determine subsequent load or store hits. Under these conditions, load hits can occur concurrently with store hits to write-back memory without stalling the pipeline. Requests abandoned by the CPU can also be aborted by the cache controller.

Cache line locking can completely eliminate DCU line misses in critical data. Cache line locking can be performed line by line, and is controlled by the CDBCR and IOP 480 CPU instructions.

The DCU provides two additional features that allow the programmer to tailor its performance for a given application. The DCU can function in write-back or write-through mode, as controlled by the Data Cache Write-thru Register (DCWR) or the translation look-aside buffer (TLB); performance of the cache controller can be tuned for a balance of performance and memory coherency. Write-on-allocate, controlled by the CDBCR[WOA] field, can inhibit line fills caused by a store miss to further reduce potential pipeline stalls and unwanted external bus traffic.

23.4.3 Timers

The IOP 480 CPU contains a time base and three timers:

- Programmable Interval Timer (PIT)
- Fixed Interval Timer (FIT)
- Watchdog timer

The time base is a 64-bit counter incremented either by an internal signal equal to the CPU clock rate or by a separate external timer clock signal. No interrupts are generated when the time base rolls over.

The PIT is a 32-bit register that is decremented at the same rate as the time base is incremented. The user loads the PIT register with a value to create the desired delay. When the register is decremented to zeros, the timer stops decrementing, a bit is set in the Timer Status Register (TSR), and a PIT interrupt is generated. Optionally, the PIT can be programmed to reload automatically the last value written to the PIT register, after which the PIT begins decrementing again.The Timer Control Register (TCR) contains the interrupt enable for the PIT interrupt.

The FIT generates periodic interrupts based on selected bits in the time base. Users can select one of four intervals for the timer period by setting the appropriate bits in the TCR. When the selected bit in the time base changes from 0 to 1, a bit is set in the TSR and a FIT interrupt is generated. The FIT interrupt enable is contained in the TCR.

The watchdog timer generates a periodic interrupt based on selected bits in the time base. Users can select one of four time periods for the interval and the type of reset generated if the watchdog timer expires twice without an intervening clear from software.

23.4.4 Memory Management Unit (MMU)

The IOP 480 CPU has a 4-GB address space, which is presented as a flat address space.

The IOP 480 CPU MMU provides address translation, protection functions, and storage attribute control for embedded applications. The MMU supports demand paged virtual memory and other management schemes that require precise control of logical-to-physical address mapping and flexible memory protection. Working with appropriate system level software, the MMU provides the following functions:

- Translation of the 4-GB logical address space into physical addresses
- Independent enabling of instruction and data translation/protection
- Page level access control using the translation mechanism
- Software control of page replacement strategy
- Additional control over protection using zones
- Storage attributes for cache policy and speculative memory access control

The MMU can be disabled under software control. If the MMU is not used, the IOP 480 CPU provides other storage control mechanisms.

The translation lookaside buffer (TLB) is the hardware resource that controls translation and protection. It consists of 64 entries, each specifying a page to be translated. The TLB is fully associative; a given page entry can be placed anywhere in the TLB. The translation function of the MMU occurs pre-cache. Cache tags and indexing use physical addresses.

Software manages the establishment and replacement of TLB entries. This gives system software significant flexibility in implementing a custom page replacement strategy. For example, to reduce TLB thrashing or translation delays, software can reserve several TLB entries in the TLB for globally accessible static mappings. The instruction set provides several instructions used to manage TLB entries. These instructions are privileged and require the software to be executing in supervisor state. Additional TLB instructions are provided to move TLB entry fields to and from GPRs.

The MMU divides logical storage into pages. Eight page sizes (1 KB, 4 KB, 16 KB, 64 KB, 256 KB, 1 MB, 4 MB, 16 MB) are simultaneously supported, such that, at any given time, the TLB can contain entries for any combination of page sizes. In order for a logical to physical translation to exist, a valid entry for the page containing the logical address must be in the TLB. Addresses for which no TLB entry exists cause TLB-Miss exceptions.

To improve performance, four instruction-side TLB entries are kept in a shadow array. The shadow array helps to avoid TLB contention with load/store operations. Hardware manages the replacement and invalidation of shadow-TLB entries; no system software action is required. The shadow array can be thought of as a level 1 instruction-side TLB, with the main TLB serving as a level 2 instruction-side and a level 1 data-side TLB.

When address translation is enabled, the translation mechanism provides a basic level of protection. Physical addresses not mapped by a page entry are inaccessible when translation is enabled. Read access is implied by the existence of the valid entry in the TLB. The EX and WR bits in the TLB entry further define levels of access for the page, by permitting execute and write access, respectively.

The Zone Protection Register (ZPR) enables the system software to override the TLB access controls. For example, the ZPR provides a way to deny read access to application programs. The ZPR can be used to classify storage by type; access by type can be changed without manipulating individual TLB entries.

When translation is disabled, the IOP 480 CPU uses several registers to control the storage attribute settings.

23.4.5 Debug

The IOP 480 CPU debug facilities include debug modes for the various types of debugging used during hardware and software development. Also included are debug events that allow developers to control the debug process. Debug modes and debug events are controlled using debug registers in the chip. The debug registers are accessed either through software running on the processor, or through the JTAG port. The JTAG port can also be used for board test.

The debug modes, events, controls, and interfaces provide a powerful combination of debug facilities for a complete set of hardware and software development tools such as RISCWatch and OS Open from IBM.

23.4.5.1 Development Tool Support

The IOP 480 CPU provides powerful debug support for a wide range of hardware and software development tools.

The OS Open Real-time Operating System Debugger is an example of an operating system-aware debugger, implemented using software traps.

RISCWatch an example of a development tool that uses the external debug mode, debug events, and the JTAG port to support hardware and software development and debugging.

Logic analyzers from Hewlett-Packard and Tektronix provide the IOP 480 CPU disassembler with support as well as support for the RISCTrace feature of RISCWatch.

23.4.5.2 Debug Modes

The IOP 480 CPU supports two debug modes, internal and external; each mode supports a different type of debug tool used in embedded systems development. Internal debug mode supports ROM monitors, and external debug mode supports emulators. Both modes can be enabled simultaneously. The debug modes are controlled by the Debug Control Register (DBCR).

Internal debug mode supports accessing architected processor resources, setting hardware and software breakpoints, and monitoring processor status. In internal debug mode, debug events can generate debug exceptions, which can interrupt normal program flow so that monitor software can collect processor status and alter processor resources.

Internal debug mode relies on exception-handling software, running in the processor, and an external communications path, to debug software problems. This mode is used while the processor is executing instructions and enables debugging of problems in application or operating system code.

Access to debugger software executing in the processor, while in internal debug mode, is through a communications port on the processor board, such as a serial port.

External debug mode, accessed through a JTAG port, supports stopping and starting the processor, accessing architected processor resources, setting hardware and software breakpoints, and monitoring processor status. In external debug mode, debug events can architecturally “freeze” the processor. While the processor is frozen, normal instruction execution stops, and the architected processor resources can be accessed and altered.

External debug mode relies only on internal processor resources to debug system hardware and software problems. This mode can also be used for software development on systems without a control program, or to debug control program code.

23.4.5.3 PLB (Processor Local Bus)

The PLB-compliant interface provides separate 32-bit address and data buses for the instruction and data sides.

23.4.5.4 JTAG

The IOP 480 CPU JTAG port is enhanced to support the attachment of a debug tool such as the RISCWatch product from IBM Microelectronics. Through the JTAG test access port, a debug workstation can single-step the processor and interrogate internal processor state to facilitate software debugging. The enhancements comply with the IEEE 1149.1 specification for vendor-specific extensions, and are therefore compatible with standard JTAG hardware for boundary-scan system testing.

23.4.5.5 Interrupts

The IOP 480 CPU provides an interface to an interrupt controller that is logically outside the IOP 480 CPU processor core. This controller combines the asynchronous interrupt inputs and presents them to the core as a single interrupt signal. The sources of asynchronous interrupts are external signals, the JTAG/debug unit, and any implemented peripherals.

23.4.6 Data Types

The IOP 480 CPU operands are bytes, words, or Lwords. Multiple words or strings of bytes can be transferred using the load/store multiple and load/store string instructions. Data is represented in *twos* complement notation or in unsigned fixed-point format.

The address of a multi-byte operand is always the lowest memory address occupied by that operand.

23.4.7 Register Set Summary

The registers can be grouped into basic categories based on function and access mode—general purpose registers (GPRs), special purpose registers (SPRs), the machine state register (MSR), the condition register (CR), and, in standard products, device control registers (DCRs).

Section 29, “IOP 480 CPU Register Summary,” provides a register diagram and a register field description table for each register.

23.4.7.1 General Purpose Registers

The IOP 480 CPU contains thirty-two 32-bit GPRs. The contents of the GPRs can be transferred from memory using load instructions and stored to memory using store instructions. GPRs, which are specified as operands in many of the IOP 480 CPU instructions, can also hold instruction results and contents of other registers.

23.4.7.2 Special Purpose Registers

Special Purpose Registers (SPRs), which are part of the PowerPC Architecture, are accessed using the **mtspr** and **mfspr** instructions. SPRs control the use of the debug facilities, timers, interrupts, storage control attributes, and other architected processor resources.

The only SPRs that are not privileged for read and write access are the Count Register (CTR), Link Register (LR), and Fixed Point Exception Register (XER). User-mode programs have read-only access to the Time Base High User-mode (TBHU) and Time Base Low User-mode (TBLU) time base registers.

23.4.7.3 Machine State Register

The IOP 480 CPU contains a 32-bit Machine State Register (MSR). The contents of a GPR can be written to the MSR using the **mtmsr** instruction, and the MSR contents can be read into a GPR using the **mfmsr** instruction. The MSR contains fields that control the operation of the IOP 480 CPU.

23.4.7.4 Condition Register

The IOP 480 CPU contains a 32-bit Condition Register (CR). These bits are grouped into eight 4-bit fields, CR[CR0]–CR[CR7]. Instructions are provided to perform logical operations on CR fields and bits within fields and to test CR bits within fields. The CR fields, which are set by compare instructions, can be used to control branches. CR[CR0] can be set implicitly by arithmetic instructions.

23.4.7.5 Device Control Registers

Device Control Registers (DCRs), which are architecturally outside of the processor core, are accessed using the mtdcr and mfdcr instructions. DCRs are used to control, configure, and hold status for various functional units that are not part of the processor core. Although the IOP 480 CPU does not contain DCRs, the mtdcr and mfdcr instructions are provided.

The mtdcr and mfdcr instructions are privileged, for all DCRs; therefore, all accesses to DCRs are privileged. See Section 24.8, “Privileged Mode Operation,” on page 24-27.

All DCR numbers are ***reserved***, and should not be read nor written, unless they are part of an IBM Core+ASIC implementation.

23.4.8 Addressing Modes

The IOP 480 CPU supports the following addressing modes to allow efficient retrieval and storage of data in memory:

- Base plus displacement addressing
- Indexed addressing
- Base plus displacement addressing and indexed addressing, with update

In the base plus displacement addressing mode, an effective address (EA) is formed by adding a displacement to a base address contained in a GPR (or to an implied base of 0). The displacement is an immediate field in an instruction.

In the indexed addressing mode, the EA is formed by adding an index contained in a GPR to a base address contained in a GPR (or to an implied base of 0).

The base plus displacement and the indexed addressing modes also have a “with update” mode. In “with update” mode, the effective address calculated for the current operation is saved in the base GPR, and can be used as the base in the next operation. The “with update” mode relieves the processor from repeatedly loading a GPR with an address for each piece of data, regardless of the proximity of the data in memory.

24 IOP 480 CPU PROGRAMMING MODEL

The programming model of the IOP 480 CPU embedded controller describes how the following features and operations appear to programmers:

- Memory organization and addressing
- Registers
- Data types and alignment
- Byte ordering
- Instruction processing
- Branching control
- Speculative accesses
- Privileged mode operation
- Synchronization
- Instruction set

24.1 MEMORY ORGANIZATION AND ADDRESSING

The PowerPC Architecture defines a 32-bit, 4 GB flat address space for instructions and data.

The user's manuals for standard products containing the IOP 480 CPU describe their memory organizations and physical address maps.

24.1.1 Storage Attributes

PowerPC Architecture defines storage attributes that control data and instruction accesses. Storage attributes are provided to control cache write-through policy (the W storage attribute), cacheability (the I storage attribute), memory coherency in multiprocessor environments (the M storage attribute), and guarding against speculative memory accesses (the G storage attribute). The IBM PowerPC Embedded Environment defines additional storage attributes for storage compression (the K storage attribute) and byte ordering (the E storage attribute).

The IOP 480 CPU provides control mechanisms for the WIGKE attributes. Because the IOP 480 CPU does not provide hardware support for multiprocessor environments, the M storage attribute, when present, has no effect.

When the IOP 480 CPU operates in virtual mode (address translation is enabled), each storage attribute is controlled by the WIGEK fields in the TLB entry for each memory page. (An M field is present but ignored.) The size of memory pages, and hence the size of storage attribute control regions, can be set to 1 KB, 4 KB, 16 KB, 64 KB, 256 KB, 1 MB, 4 MB, or 16 MB. Multiple sizes can be in effect simultaneously on different pages.

When the IOP 480 CPU operates in real mode (address translation is disabled), the storage attribute control registers control the storage attributes. These registers are:

- Data Cache Write-thru Register (DCWR)
- Data Cache Cacheability Register (DCCR)
- Instruction Cache Cacheability Register (ICCR)
- Storage Guarded Register (SGR)
- Storage Compression Register (SKR)
- Storage Little-Endian Register (SLER)

Section 29, "IOP 480 CPU Register Summary," contains bit descriptions for these registers.

When the IOP 480 CPU operates in virtual mode (address translation is enabled), the storage attribute control registers are ignored.

Each storage attribute control register contains 32 bits; each bit controls one of thirty-two 128 MB storage attribute control regions. Bit 0 of each register controls the lowest-order region, with ascending bits controlling ascending regions in memory. Each region is selected by address bits A[0:4]. The storage attributes in each storage attribute region are set independently.

24.2 REGISTERS

Some of the more commonly-used registers are described in this section. Other registers are covered in their respective topic sections (*for example*, the cache registers are described in Section 25, "IOP 480 CPU Cache Operations"). All registers are summarized in Section 29, "IOP 480 CPU Register Summary."

All registers in the IOP 480 CPU are 32-bit registers. The registers are grouped into categories, based on access mode: general purpose registers (GPRs), special purpose registers (SPRs), the time base, the Machine State Register (MSR), the Condition Register (CR), and, in standard products, device control registers (DCRs).

For all registers with fields marked as *reserved*, the reserved fields should be written as 0 and read as *undefined*. That is, when writing to a register with a reserved field, write a 0 to the reserved field. When reading from a register with a reserved field, ignore that field. A good coding practice is to perform the initial write to a register with reserved fields as described, and to perform all subsequent writes to the register using a read-modify-write strategy: read the register, use logical instructions to alter defined fields, leaving reserved fields unmodified, and write the register.

24.2.1 General Purpose Registers

The IOP 480 CPU contains 32 general purpose registers (GPRs); each contains 32 bits. Data from memory can be loaded into GPRs using load instructions; the contents of GPRs can be stored in memory using store instructions. Most integer instructions reference GPRs. See Register 24-1 for GPR numbering.

24.2.2 Special Purpose Registers

Special Purpose Registers (SPRs), which are part of the PowerPC Architecture and the IBM PowerPC Embedded Environment, are accessed using the **mtspr** and **mfspr** instructions.

SPRs control the use of the debug facilities, timers, interrupts, storage control attributes, and other architected processor resources. Table 24-1 shows the mnemonic, name, and number for each SPR. Table 24-1 also lists the IOP 480 CPU SPRs by function and points to the pages where the SPRs are described more fully.

Register 24-1. General Purpose Register (R0-R31)

0:31		GPR data	

Table 24-1. IOP 480 CPU SPRs

Function	Register				Access	Page
Branch Control	CTR				User	24-4
	LR				User	24-4
Debug	CDBCR				Privileged	25-7
	DAC1				Privileged	26-7
	DBCR				Privileged	26-3
	DBSR				Privileged	26-5
	IAC1				Privileged	26-8
	ICDBDR				Privileged	25-9
Fixed-Point Exception	XER				User	24-5
General Purpose	SPRG0	SPRG1	SPRG2	SPRG3	Privileged	24-6
Interrupts and Exceptions	DEAR				Privileged	11-19
	ESR				Privileged	11-17
	EVPR				Privileged	11-17
	SRR0	SRR1			Privileged	11-15
	SRR2	SRR3			Privileged	11-15
Processor Version	PVR				Privileged, read-only	24-6
Storage Attributes	DCCR				Privileged	27-13
	DCWR				Privileged	
	ICCR				Privileged	
	SGR				Privileged	
	SKR				Privileged	
	SLER				Privileged	
Timer Facilities	TBHI	TBLO			Privileged	11-30
	TBHU	TBLU			User read-only	11-30
	PIT				Privileged	11-26
	TCR				Privileged	11-36
	TSR				Privileged	11-35

Except for the Link Register (LR), the Count Register (CTR), the Fixed-point Exception Register (XER), and the Time Base High User-mode (TBHU) and theTime Base Low User-mode (TBLU), all SPRs are privileged. See Section 24.8.3, “Privileged SPRs,” on page 24-28. The Processor Version Register (PVR) is read-only.

24.2.2.1 Count Register (CTR)

The CTR is written from a GPR using the **mtspr** instruction. The CTR contents can be used as a loop count that is decremented and tested by some branch instructions. This usage does not incur any performance penalty; the branches execute in the normal branch instruction execution time. Alternatively, the CTR contents can specify a target address for the **bcctr** instruction, enabling indirectly-addressed branching to any address.

The CTR is available to user programs.

Register 24-2 lists the CTR bits.

24.2.2.2 Link Register (LR)

The LR is written from a GPR using the **mtspr** instruction or branch instructions that have the LK bit set to 1. Such branch instructions load the LR with the address of the instruction following the branch instruction (4 + address of the branch instruction). Thus, the LR contents can be a return address for a subroutine which was entered using the branch.

The LR contents can be used as a target address for the **bclr** instruction. This allows indirectly-addressed branching to any address.

When the LR contents represent an instruction address, $LR_{30:31}$ are assumed to be zero, because all instructions must be Lword-aligned. However, when LR is written using **mtspr** and then read using **mfspr**, all 32 bits are returned.

The LR is available to user programs.

Register 24-3 lists the LR bits.

Register 24-2. Count Register (CTR)

0:31		Count	Used as count for branch conditional with decrement instructions, or as address for branch-to-counter instructions.

Register 24-3. Link Register (LR)

0:31		Link Registers contents	If (LR) represents an instruction address, $LR_{30:31}$ should be zero.

24.2.2.3 Fixed Point Exception Register (XER)

The XER records overflow and carry conditions from arithmetic operations.

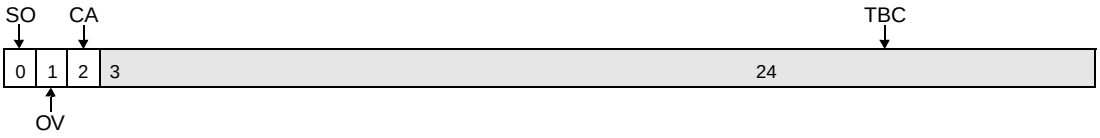
The Summary Overflow (SO) field does not necessarily indicate that an overflow occurred on the most recent arithmetic operation, but that one occurred previously sometime since the last clearing of the XER. XER[SO] can be set to zero only by using the **mtspr** instruction or the **mcrxr** instruction.

The TBC field can be written, using **mtspr**, with a byte count for load/store string instructions. The XER is available to user programs.

Several special cases are associated with the use of the XER bits, as documented in the sections that follow.

Register 24-4 illustrates the XER bits.

Register 24-4. Fixed Point Exception Register (XER)



0	SO	Summary Overflow 0 No overflow has occurred. 1 Overflow has occurred.	Can be *set* by **mtspr** or using arithmetic instructions with the “OE” option (see Table 24-2); can be *reset* by **mtspr** or by **mcrxr**.
1	OV	Overflow 0 No overflow has occurred. 0 Overflow has occurred.	Can be *set* by **mtspr** or arithmetic instructions with the “OE” option (see Table 24-2); can be *reset* by **mtspr**, by **mcrxr**, or by arithmetic instructions with the “OE” option.
2	CA	Carry 0 Carry has not occurred. 1 Carry has occurred.	Can be *set* by **mtspr** or arithmetic instructions that update the CA field (see Table 24-2); can be *reset* by **mtspr**, by **mcrxr**, or by arithmetic instructions that update the CA field.
3:24		*Reserved*	

Table 24-2. XER-Updating Arithmetic Instructions

Update XER[CA]		Update XER[OV] Set XER[SO]	
addc addc. addco addco. adde adde. addeo addeo. addic addic. addme addme. addmeo addmeo. addze addze. addzeo addzeo.	subfc subfc. subfco subfco. subfe subfe. subfeo subfeo. subfic subfme subfme. subfmeo subfmeo. subfze subfze. subfzeo subfzeo.	addo addo. addco addco. addeo addeo. addmeo addmeo. addzeo addzeo. divwo divwo. divwuo divwuo.	mullwo mullwo. nego nego. subfo subfo. subfco subfco. subfeo subfeo. subfmeo subfmeo. subfzeo subfzeo.

24.2.2.3.1 XER[SO]

Summary overflow; set to 1 when an instruction causes XER[OV] to be set to 1, except for **mtspr**(XER), which sets XER[SO,OV] to the value of bit positions 0 and 1 in the source register, respectively. Once set, XER[SO] is not reset until an **mtspr**(XER) is executed with data that explicitly puts a 0 in the SO bit, or until an **mcrxr** instruction is executed.

24.2.2.3.2 XER[OV]

Overflow; set to indicate whether or not an instruction that updates XER[OV] produces a result that "overflows" the 32-bit target register. XER[OV] = 1 indicates overflow. For arithmetic operations, this occurs when an operation has a carry-in to the most-significant bit of the instruction result that does not equal the carry-out of the most-significant bit (that is, the exclusive-or of the carry-in and the carry-out is 1).

The following instructions set XER[OV] differently.The specific behavior is indicated in the instruction descriptions.

- Move instructions
  mcrxr, **mtspr**(XER)
- Multiply and divide instructions
  mullwo, **mullwo.**, **divwo**, **divwo.**, **divwuo, divwuo.**

24.2.2.3.3 XER[CA]

Carry; set to indicate whether or not an instruction that updates XER[CA] produces a result that has a carry-out of the most-significant bit. XER[CA] = 1 indicates a carry.

The following instructions set XER[CA] differently. The specific behavior is indicated in the instruction descriptions.

- Move instructions
  mcrxr, **mtspr**(XER)
- Shift-algebraic operations
  sraw, **srawi**

24.2.2.3.4 XER[TBC]

Transfer Byte Count.

This field provides a byte count for the **lswx** and **stswx** instructions.

This field is updated by **mtspr**(XER).

24.2.2.4 Special Purpose Register General (SPRG0-SPRG3)

These four registers are provided as temporary storage locations. *For example*, a supervisor routine might save the contents of a GPR to an SPRG, and later restore the GPR from it. This is faster than the standard save/restore to a memory location. These registers are written to using the **mtspr** instruction and read from using the **mfspr** instruction.

Access to the SPRGs is privileged. See Section 24.8.3, "Privileged SPRs," on page 24-28 for more information.

See Register 24-5 for numbering.

24.2.2.5 Processor Version Register (PVR)

PVR is a read-only register that identifies the processor by Version and Revision numbers. Software can use features that depend upon an exact identification of the target processor. Such software can examine the PVR to select appropriate features dynamically.

The 16-bit Version number (comprised of the FAM and MEM fields) is assigned by the PowerPC Architecture process.

The 16-bit Revision number (comprised of the CORE and CHIP fields) is assigned by the chip implementer.

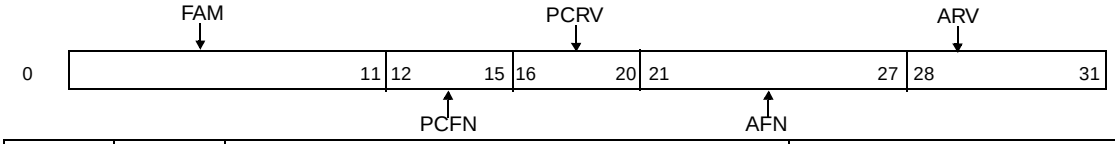
Access to the PVR is privileged. See Section 24.8.3, "Privileged SPRs," on page 24-28 for more information.

Register 24-6 illustrates the PVR bits.

Register 24-5. Special Purpose Register General (SPRG0-SPRG3)

0:31		General data	Privileged user-specified; no hardware usage.

Register 24-6. Processor Version Register (PVR)



0:11	FAM	Processor Family. Identifies a PowerPC family, such as 4xx or 6xx.	0x002 for the 4xx family.
12:15	PCFN	Processor Core Function. Identifies a specific processor core implementation.	2 for PPC401B2.
16:20	PCRV	Processor Core Revision. Identifies a revision of the processor core defined by the PFN field.	
21:27	AFN	ASIC Function. An assigned identifier for an ASIC containing a PowerPC 400 Series processor core.	
28:31	ARV	ASIC Revision. An assigned identifier for a revision of the ASIC defined by the AFN field.	

24.2.3 Condition Register (CR)

The Condition Register (CR) contains eight 4-bit fields (CR0–CR7), as shown in Register 24-7. The CR reflects the results of some operations (as indicated in the instruction descriptions in Section 28, “IOP 480 CPU Instruction Set”). The CR supports condition testing and conditional branching.

Fields of the CR can be set in any of the following ways:

- Specified fields can be set by writing to the CR from a GPR (**mtcrf** instruction).
- A specified field can be set by writing to the field from another CR field (**mcrf** instruction) or from the XER (**mcrxr** instruction).
- CR[CR0] can be set as the implicit result of various fixed-point instructions.
- The bits in a specified field can be set as the result of a Compare instruction.

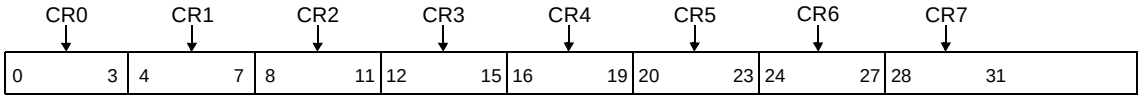
Additional instructions perform logical operations on one or more bits in a CR field (the CR-logical instructions); other instructions (the branch conditional instructions) test the bits in a CR field.

If a CR field is set by a compare instruction, the bits in the selected field are set as described in Section 24.2.3.1, “CR Fields after Compare Instructions,” on page 24-8. Further, CR[CR0] field is altered implicitly by numerous instructions and the interpretation of CR[CR0] is discussed further in Section 24.2.3.1, “CR Fields after Compare Instructions,” on page 24-8.

The CR is non-privileged. See Section 24.8.3, “Privileged SPRs,” on page 24-28 for more information.

Register 24-7 illustrates the CR bits.

Register 24-7. Condition Register (CR)



0:3	CR0	Condition Register Field 0	$CR[CRn]_{0:3}$ indicate less than, greater than, equal to, and summary overflow, respectively.
4:7	CR1	Condition Register Field 1	See the description of CR[CR0].
8:11	CR2	Condition Register Field 2	See the description of CR[CR0].
12:15	CR3	Condition Register Field 3	See the description of CR[CR0].
16:19	CR4	Condition Register Field 4	See the description of CR[CR0].
20:23	CR5	Condition Register Field 5	See the description of CR[CR0].
24:27	CR6	Condition Register Field 6	See the description of CR[CR0].
28:31	CR7	Condition Register Field 7	See the description of CR[CR0].

24.2.3.1 CR Fields after Compare Instructions

Compare instructions compare the values of two 32-bit numbers. The two types of compare instructions, *arithmetic* and *logical*, are distinguished by the interpretation given to the 32-bit numbers. For *arithmetic* compares, the numbers are considered to be signed, where 31 bits are significant; the most-significant bit is a sign bit. For *logical* compares, the numbers are considered to be unsigned (all 32 bits are significant; there is no sign bit). As an example, consider the comparison of 0 with 0xFFFF FFFF. In an *arithmetic* compare, 0 is larger; in a *logical* compare, 0xFFFF FFFF is larger.

A compare instruction can direct its results to any CR field. The BF field (bits 6:8) of the instruction specifies the CR field. The first data operand of a compare instruction specifies a GPR. The second data operand specifies another GPR, or immediate data derived from the IM field (bits 16:31) of the immediate instruction form. The contents of the GPR specified by the first data operand are compared with the contents of the GPR specified by the second data operand (or with the immediate data). See descriptions of the compare instructions for precise details.

After a compare, the specified CR field is interpreted as follows:

LT (bit 0) — The first operand is less than the second operand.

GT (bit 1) — The first operand is greater than the second operand.

EQ (bit 2) — The first operand is equal to the second operand.

SO (bit 3) — Summary overflow; a copy of XER[SO].

24.2.3.2 CR0 Field

After the execution of compare instructions with BF=0, the CR[CR0] is interpreted as described in Section 24.2.3.1, “CR Fields after Compare Instructions,” above. The “dot” forms of arithmetic and logical instructions also alter CR[CR0]. After most fixed-point instructions that update CR[CR0], the bits of CR0 are interpreted as follows:

LT (bit 0) — Less than 0; set if the most-significant bit of the 32-bit result is 1.

GT (bit 1) — Greater than 0; set if the 32-bit result is non-zero and the most-significant bit of the result is 0.

EQ (bit 2) — Equal to zero; set if the 32-bit result is 0.

SO (bit 3) — Summary overflow; a copy of XER[SO] at instruction completion.

The $CR[CR0]_{LT, GT, EQ}$ subfields are set as the result of an algebraic comparison of the instruction result to 0, regardless of the type of instruction that sets CR[CR0]. If the instruction result is 0, the EQ subfield is set to 1. If the result is not 0, whether the LT subfield or the GT subfield is set depends on the value of the most-significant bit of the instruction result.

When updating CR[CR0], the most significant bit of an instruction result is considered a sign bit, even for instructions that produce results that are not usually thought of as signed. *For example*, logical instructions such as **and.**, **or.**, and **nor.** update $CR[CR0]_{LT, GT, EQ}$ using such an arithmetic comparison to 0, although the result of such a logical operation is often not actually an arithmetic result.

Note: *If an arithmetic overflow occurs, the "sign" of an instruction result indicated by $CR[CR0]_{LT, GT, EQ}$ might not represent the "true" (infinitely precise) algebraic result of the instruction that set CR0. For example, if an* **add.** *instruction adds two large positive numbers and the magnitude of the result cannot be represented as a twos-complement number in a 32-bit register, an overflow occurs and $CR[CR0]_{LT, SO}$ are set, although the infinitely precise result of the add is positive.*

Adding the largest 32-bit *twos*-complement negative number, 0x8000 0000, to itself results in an arithmetic overflow and 0x0000 0000 is recorded in the target register. $CR[CR0]_{EQ, SO}$ is set, indicating a result of 0, but the infinitely precise result is negative.

The $CR[CR0]_{SO}$ subfield is a copy of XER[SO]. Instructions that do not alter the XER[SO] bit cannot cause an overflow, but even for these instructions $CR[CR0]_{SO}$ is a copy of XER[SO].

Some instructions set CR[CR0] differently or do not specifically set any of the subfields. These instructions include:

- Compare instructions
  cmp, **cmpi**, **cmpl**, **cmpli**
- CR logical instructions
  crand, **crandc**, **creqv**, **crnand**, **crnor**, **cror**, **crorc**, **crxor**, **mcrf**
- Move CR instructions
  mtcrf, **mcrxr**
  stwcx

The instruction descriptions provide detailed information about how the listed instructions alter CR[CR0].

24.2.4 Time Base

The IOP 480 CPU implements a 64-bit time base. The time base, which increments once during each period of the time base clock, provides a time reference. The time base is accessed using the 32-bit registers TBLO and TBHI. Software access to the time base is through the **mfspr** and **mtspr** instructions.

Access to the time base registers TBHI and TBLO is privileged.

User-mode read-only access to the Time Base is provided by reading from different SPR numbers. Specifically, read-only access to TBHI is accomplished by reading TBHU, and read-only access to TBLO is accomplished by reading TBLU. Both TBHU and TBLU are read using **mfspr** instructions. An **mtspr** to these registers is boundedly undefined.

The time base differs from the time base described in the PowerPC Architecture. See Section 11.7.18.1, "Time Base," on page 11-30 for detailed differences between the IOP 480 CPU time base and the time base described in PowerPC Architecture.

24.2.5 Machine State Register (MSR)

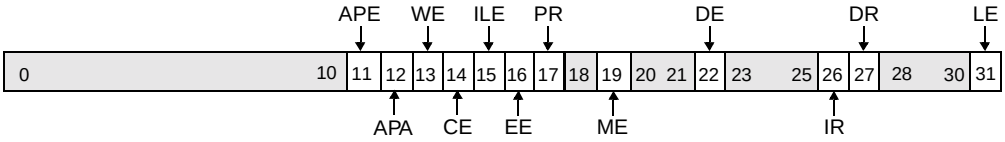
The Machine State Register (MSR) controls important chip functions, such as the enabling or disabling of interrupts and debugging exceptions.

The MSR can be written from a GPR using the **mtmsr** instruction. The contents of the MSR can be written into a GPR using the **mfmsr** instruction. The MSR[EE] (External Interrupt Enable) bit may be set/cleared atomically using the **wrtee** or **wrteei** instructions.

The MSR contents are automatically saved, altered, and restored by the interrupt-handling mechanism. See Section 11.7.3, "General Exception Handling Registers," on page 11-13.

Register 24-8 illustrates the MSR bits.

Register 24-8. Machine State Register (MSR)



0:10		*Reserved*	
11	APE	Auxiliary Processor Exception Enable 0 Auxiliary processor exception disabled. 1 Auxiliary processor exception enabled.	
12	APA	Auxiliary Processor Available 0 Auxiliary processor not available. 1 Auxiliary processor available.	
13	WE	Wait State Enable 0 The processor is not in the wait state. 1 The processor enters the wait state until an exception is taken, or the IOP 480 CPU is reset, or an external debug tool clears WE.	
14	CE	Critical Interrupt Enable 0 Critical interrupts are disabled. 1 Critical interrupts are enabled.	CE controls the critical interrupt input and watchdog timer first timeout interrupts.
15	ILE	Interrupt Little Endian 0 Interrupt handlers execute in Big Endian mode. 1 Interrupt handlers execute in PowerPC Little Endian mode.	MSR(ILE) is copied to MSR(LE) when an interrupt is taken.
16	EE	External Interrupt Enable 0 Asynchronous exceptions are disabled. 1 Asynchronous exceptions are enabled.	EE controls the non-critical external interrupt input, Programmable Interval Timer, and Fixed Interval Timer interrupts.
17	PR	Problem State 0 Supervisor State (all instructions allowed) 1 Problem State (some instructions not allowed)	
18		*Reserved*	
19	ME	Machine Check Enable 0 Machine check exceptions are disabled 1 Machine check exceptions are enabled.	
20:21		*Reserved*	
22	DE	Debug Exception Enable 0 Debug exceptions are disabled. 1 Debug exceptions are enabled.	
23:25		*Reserved*	
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.	If TIE_cpuMmuEn is 0, reading or writing this bit has no effect.
27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.	If TIE_cpuMmuEn is 0, reading or writing this bit has no effect.
28:30		*Reserved*	
31	LE	Little Endian 0 Processor executes in Big Endian mode. 1 Processor executes in PowerPC Little Endian mode.	

24.2.6 Device Control Registers

Device Control Registers (DCRs), on-chip registers that exist architecturally outside the processor core, are not part of the IBM PowerPC Embedded Environment. The Embedded Environment simply defines the existence of a DCR address space and the instructions that access the DCRs, but does not define any DCRs. The instructions that access the DCRs are **mtdcr** (move to device control register) and **mfdcr** (move from device control register).

24.3 DATA TYPES AND ALIGNMENT

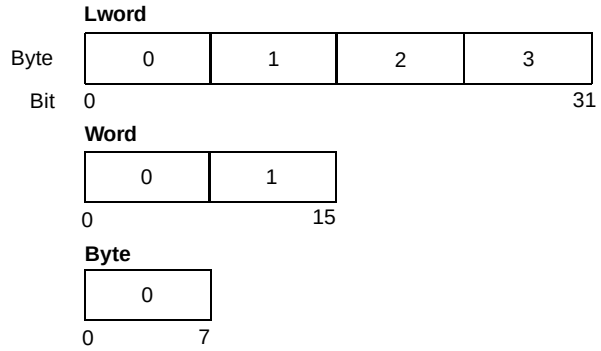


Figure 24-1. IOP 480 CPU Data Types

The IOP 480 CPU data types consist of bytes (8 bits), words (16 bits), Lwords (32 bits), and strings (one or more bytes containing character data). Figure 24-1 shows the byte, word, and Lword data types and their bit and byte definitions.

Data is represented in *twos* complement notation or in an unsigned integer format; data representation is independent of alignment issues.

The address of an a data object is always the lowest address of any byte comprising the object.

All instructions are Lwords, and are Lword-aligned (the byte address is divisible by 4).

24.3.1 Alignment for Storage Reference and Cache Control Instructions

The storage reference instructions (loads and stores; see Table 24-11 on page 24-32) move data to and from storage. The data cache control instructions (see Table 24-17 on page 24-34) control the contents and operation of the data cache unit (DCU). Both types of instructions form an effective address (EA). The method of calculating the EA for the storage reference and cache control instructions is detailed in the description of those instructions. See Section 28, “IOP 480 CPU Instruction Set,” for more information.

Cache control instructions ignore the four least significant bits in the EA; no alignment restrictions exist in the DCU because of EAs. However, storage control attributes for a storage region can cause alignment exceptions. Specifically, when data translation is disabled and a **dcbz** instruction references a region that is non-cacheable or for which write-through caching is enabled, an alignment exception is *taken.* Such exceptions result from the storage control attributes, not from EA alignment.

Alignment requirements for the EAs of the storage reference instructions and the **dcread** cache control instruction depends on the instruction and the Endian mode of the IOP 480 CPU (see Section 24.4, “Byte Ordering,” on page 24-13 for information about Endian operation). Table 24-3 on page 24-12, summarizes the instructions that cause alignment exceptions.

The data targets of instructions are of types that depend upon the instruction. The load/store instructions have the following “natural” alignments:

- Load/store Lword instructions have Lword targets, Lword-aligned
- Load/ store word instructions have word targets, word-aligned
- Load/store byte instructions have byte targets, byte-aligned (that is, any alignment)

Note: *The IOP 480 CPU implementation handles misalignments within and across Lword boundaries.*

Misalignments are addresses that are not naturally aligned on data type boundaries. An address not divisible by four is misaligned with respect to Lword instructions. An address not divisible by two is misaligned with respect to word instructions.

24.3.2 Alignment and Endian Operation

The IOP 480 CPU is operating as a *Big Endian processor* (MSR[LE] = 0), and EA misalignments do not cause alignment exceptions except as summarized in Table 24-3.

24.3.3 Instructions Causing Alignment Exceptions Summary

Table 24-3 summarizes the instructions that cause alignment exceptions and the conditions under which the alignment exceptions occur.

Table 24-3. Alignment Exception Summary

IOP 480 CPU MSR	Instructions Causing Alignment Exceptions	Conditions
MSR[LE] = 0	dcbz	EA in non-cacheable or write-through storage
	dcread, lwarx, stwcx.	EA not Lword-aligned
MSR[LE] = 1	dcbz	EA in non-cacheable or write-through storage
	lha, lhau, lhaux, lhax, lhbrx, lhz, lhzu, lhzux, lhzx, sth, sthbrx, sthu, sthux, sthx	EA not word-aligned
	dcread, lwarx, lwbrx, lwz, lwzu, lwzux, lwzx, stw, stwbrx, stwcx., stwu, stwux, stwx	EA not Lword-aligned
	lmw, lswi, lswx, stmw, stswi, stswx, stswcx.	Always

24.4 BYTE ORDERING

If scalars (individual data items and instructions) were indivisible, there would be no such concept as “byte ordering.” It is meaningless to consider the order of bits or groups of bits within the smallest addressable unit of storage; nothing can be observed about such order. Only when scalars, which the programmer and processor regard as indivisible quantities, can comprise more than one addressable unit of storage does the question of order arise.

For a machine in which the smallest addressable unit of storage is the 64-bit double Lword, there is no question of the ordering of bytes within double Lwords. All transfers of individual scalars between registers and storage are of double Lwords, and the address of the byte containing the high-order eight bits of a scalar is no different from the address of a byte containing any other part of the scalar.

For PowerPC Architecture, as for most computer architectures currently implemented, the smallest addressable unit of storage is the 8-bit byte. Many scalars are words, Lwords, or double Lwords, which consist of groups of bytes. When an Lword-length scalar is moved from a register to storage, the scalar occupies four consecutive byte addresses. It thus becomes meaningful to discuss the order of the byte addresses with respect to the value of the scalar: which byte contains the highest-order eight bits of the scalar, which byte contains the next-highest-order eight bits, and so forth.

Given a scalar that contains multiple bytes, the choice of byte ordering is essentially arbitrary. There are 4! = 24 ways to specify the ordering of four bytes within an Lword, but only two of these orderings are sensible:

- The ordering that assigns the lowest address to the highest-order (“leftmost”) eight bits of the scalar, the next sequential address to the next-highest-order eight bits, and so forth.

  This ordering is called *Big Endian* because the “big end” of the scalar, considered as a binary number, comes first in storage. IBM RISC System/6000, IBM System/390, and Motorola 680x0 are examples of computers using this byte ordering.

- The ordering that assigns the lowest address to the lowest-order (“rightmost”) eight bits of the scalar, the next sequential address to the next-lowest-order eight bits, and so forth.

  This ordering is called *Little Endian* because the “little end” of the scalar, considered as a binary number, comes first in storage. DEC VAX and Intel x86 are examples of computers using this byte ordering.

24.4.1 Structure Mapping Examples

The following C language structure s contains an assortment of scalars and a character string. The comments show the value assumed to be in each structure element; these values show how the bytes comprising each structure element are mapped into storage.

```
struct {
   int a;      /* 0x1112_1314 Lword */
   long long b;/* 0x2122_2324_2526_2728 double Lword
   */
   char *c;  /* 0x3132_3334 Lword */
   char d[7];/* 'A','B','C','D','E','F','G' array of bytes */
   short e;  /* 0x5152 word */
   int f;      /* 0x6162_6364 Lword */
} s;
```

C structure mapping rules permit the use of padding (skipped bytes) to align scalars on desirable boundaries. The structure mapping examples show each scalar aligned at its natural boundary. This alignment introduces padding of four bytes between a and b, one byte between d and e, and two bytes between e and f. The same amount of padding is present in both Big Endian and Little Endian mappings.

24.4.1.1 Big-Endian Mapping

The Big Endian mapping of structure s follows. The data is highlighted in the structure mappings. Addresses, in hexadecimal, are below the data stored at the address. The contents of each byte, as defined in structure s, is shown as a (hexadecimal) number or character (for the string elements).

11	**12**	**13**	**14**				
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
31	**32**	**33**	**34**	**'A'**	**'B'**	**'C'**	**'D'**
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
'E'	**'F'**	**'G'**		**51**	**52**		
0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
61	**62**	**63**	**64**				
0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27

24.4.2 PowerPC Byte Ordering

By default, the PowerPC Architecture is Big Endian. This book describes the processor as if it operated only in a Big Endian fashion. IOP 480 bus control mechanisms support Little Endian operation. Subsequent sections explain these mechanisms in more detail.

24.4.3 PowerPC Endian Mode

PowerPC Endian mode is useful for system environments in which some processes and their associated data structures are written as Little Endian, and other processes are written as Big Endian. The PowerPC Endian mode mechanism handles such bi-Endian systems and manages communications and data sharing between processes running in the system. However, because of how PowerPC Endian mode operates, it does not provide for direct processor connections to Little Endian hardware, nor for operating the IOP 480 CPU in a hardware system environment that is connected in a Little Endian manner. Instead, for such environments, one should use the Endian (E) storage attribute described in Section 24.4.4, “Endian Storage Attribute,” on page 24-18.

When the IOP 480 CPU operates with the PowerPC Endian mode set to Little Endian, instructions and data in memory *appear*, from the programmer's point of view, to be arranged in Little Endian format. However, instructions and data in memory are arranged in a unique order that is neither Big Endian nor Little Endian. In addition, the processor manipulates the low-order address bits used for all instruction fetches and data references such that, when combined with the unique ordering of the bytes in memory, the instructions and data appear to the executing program to be arranged in true Little Endian order. Section 24.4.3.1, “Byte Ordering in PowerPC Little Endian Mode” below describes this unique byte arrangement and the address manipulation in detail, while Section 24.4.3.2, “Control of PowerPC Endian Mode” explains how the PowerPC Endian mode is controlled.

24.4.3.1 Byte Ordering in PowerPC Little Endian Mode

When the processor operates in PowerPC Little Endian mode, bytes, *in memory*, are rearranged from the order in which they would appear in a true Little Endian environment. Specifically, for each aligned double Lword (eight bytes) of memory, the eight bytes are reversed across the double Lword. *For example*, for the aligned double Lword at addresses A0–A7, the byte at A0 in Little Endian format is instead placed at A7 for PowerPC Little Endian mode. Likewise, the byte from A1 is moved to A6, A2 to A5, A3 to A4, A4 to A3, A5 to A2, A6 to A1, and A7 to A0. This is repeated for the next double Lword at addresses A8–A15, and so forth.

Structure s (defined in Section 24.4.1, “Structure Mapping Examples,” on page 24-13) appears *in memory* as follows after being rearranged as described.

				11	**12**	**13**	**14**
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
'D'	**'C'**	**'B'**	**'A'**	**31**	**32**	**33**	**34**
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
	51	**52**			**'G'**	**'F'**	**'E'**
0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
				61	**62**	**63**	**64**
0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27

This arrangement of bytes is neither Big Endian nor Little Endian, but is rather the result of taking the bytes from the Little Endian mapping and “swapping” them byte-for-byte across each double Lword. For this unique arrangement of bytes to appear to the executing program as equivalent to a true Little Endian arrangement, the address of each storage reference (whether for instruction fetches or for data accesses from load and store instructions) must be modified.

Specifically, the address of each storage access is modified by exclusive-ORing the low-order three bits of the address (addresses for instruction fetches are modified as for Lword accesses, because all PowerPC instructions are Lwords).

Access Type	Address Modification
Byte	XOR with 0b111
Word	XOR with 0b110
Lword	XOR with 0b100

To see how this address modification, combined with the unique ordering of bytes in memory, results in the appearance to the executing program of a true Little Endian byte arrangement, consider the following example, using the value of the Lword a from structure s. If a were stored, in Little Endian format, to address 00, it would appear as follows:

14	**13**	**12**	**11**
0x00	0x01	0x02	0x03

This memory could be accessed using Lword, word, or byte accesses in a true Little Endian system with the following results:

Lword load from address 00	0x1112_1314
Word load from address 00	0x1314
Word load from address 02	0x1112
Byte load from address 00	0x14
Byte load from address 01	0x13
Byte load from address 02	0x12
Byte load from address 03	0x11

For programmers to view memory in PowerPC Little Endian mode as equivalent to memory in a true Little Endian system, the values observed for each kind of access must match those shown.

This example shows how a is arranged in memory when stored, in PowerPC Little Endian mode, to address 0:

11	**12**	**13**	**14**
0x04	0x05	0x06	0x07

This Lword could then be accessed using Lword, word, or byte accesses in PowerPC Little Endian mode with the following results:

Lword load from (processor) address 00	converts to (memory) address 04	0x1112_1314
Word load from (processor) address 00	converts to (memory) address 06	0x1314
Word load from (processor) address 02	converts to (memory) address 04	0x1112
Byte load from (processor) address 00	converts to (memory) address 07	0x14
Byte load from (processor) address 01	converts to (memory) address 06	0x13
Byte load from (processor) address 02	converts to (memory) address 05	0x12
Byte load from (processor) address 03	converts to (memory) address 04	0x11

This example shows that a program, running on the IOP 480 CPU operating in PowerPC Little Endian mode, views Lword a in memory as if it were instead arranged in true Little Endian format. Similar results are obtained for the other members of structure s.

It should be recognized that because *instructions* in PowerPC Architecture are defined as aligned Lwords, their addressing is also affected by Endian mode. Specifically, each pair of Lwords in an aligned double Lword of memory are reversed with respect to each other when operating in PowerPC Little Endian mode. So, although a Big Endian and Little Endian program may have a sequence of instructions at addresses 00, 04, 08, 12, and so forth, and the executing program requested these instructions in order, the address modification causes the Little Endian program executing in PowerPC Little Endian mode to receive the instructions *from memory* in the following order of addresses: 04, 00, 12, 08, and so forth.

Care must be taken when loading Little Endian programs into memory to ensure that the instructions are arranged in the proper order. See Section 24.4.3.5, “Switching Endian Modes,” on page 24-17, for more detailed information.

24.4.3.2 Control of PowerPC Endian Mode

The selection of the PowerPC Endian mode is controlled by two bits in the MSR: the Little Endian mode bit (MSR[LE]) and the Interrupt Little Endian bit (MSR[ILE]).

MSR[LE] describes the current Endian mode. If MSR[LE] = 1, the processor is executing in PowerPC Little Endian mode. Otherwise, the processor executes in Big Endian mode.

When the IOP 480 CPU takes an interrupt, the MSR contents are saved in either Save/Restore Register 1 (SRR1) or Save/Restore Register 3 (SRR3), depending on the interrupt type. The content of MSR[ILE] replaces the content of MSR[LE]. The IOP 480 CPU can switch Endian modes in this fashion when entering an interrupt handler. The original value of MSR[LE] is restored from SRR1 or SRR3 upon leaving the interrupt handler (using an **rfi** or **rfci** instruction as appropriate) and returning to the previously executing program. Hence, the IOP 480 CPU can also switch Endian modes when leaving an interrupt handler. This mode-switching capability enables an operating system written in one Endian mode to support application programs written in the other mode.

The IOP 480 CPU resets to Big Endian mode, MSR[LE] = 0 and MSR[ILE] = 0.

24.4.3.3 Addressing in PowerPC Little Endian Mode

The address modification performed in PowerPC Little Endian mode affects only those addresses that are presented to the storage subsystem (including the caches). Specifically, it does not affect the original calculation of addresses, nor the value of addresses saved in registers as part of the semantics of instruction execution.

For example, the following address values are calculated independently of Endian mode, and are stored in the appropriate registers without modification:

- The address placed into the LR by a branch with link update instruction, which is equal to the Program Counter (PC) + 4
- The offset in a relative branch instruction, which reflects the difference between the addresses of the branch and target instructions as they appear to the executing program (*not necessarily* as they appear in the actual memory arrangement)
- The address placed into RA by a load/store with update instruction, which is the value computed as described in the instruction description
- The address saved in system registers, such as SRR0, SRR2, and the DEAR, as computed by the executing program and as defined for these registers

These examples do not include all addresses that are not affected by the Little Endian address modification.

The cache management instructions (**dcbi**, **icbi**, and others) are unaffected by Endian mode, because the addresses used by these instructions refer to an entire cache block (16 bytes) and the low-order four bits of the address are not used.

24.4.3.4 Little Endian Mode Alignment Requirements

The “trick” of Exclusive OR-ing the low-order three bits of the address of an individual scalar does not work unless the scalar is aligned in memory to the size of the scalar. To illustrate, consider the following example of an Lword w (containing 0x1112_1314) stored in memory at address 05, and arranged in Little Endian format:

					14	**13**	**12**
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
11							
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F

In PowerPC Little Endian mode, Lword w would be arranged in memory as follows (remember that the bytes in each aligned double Lword are reversed in the format used by PowerPC Little Endian mode):

12	**13**	**14**					
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
							11
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F

The unaligned Lword w spans two double Lwords. The two parts of the unaligned Lword are not contiguous in memory. Applying the address modification to a load Lword at address 0x05 results in address 0x01; the load Lword from address 0x05 causes the four bytes at addresses 0x01, 0x02, 0x03, and 0x04 to be accessed—clearly an incorrect result. Because of the complexity of dealing with this unusual arrangement of unaligned scalars when operating in PowerPC Little Endian mode, the IOP 480 CPU generates alignment exceptions when attempting to execute any of the following instruction types, if the IOP 480 CPU is in PowerPC Little Endian mode:

- Unaligned word or Lword load/store instruction
- String or multiple instruction (**lmw**, **lswi**, **lswx**, **stmw**, **stswi**, **stswx**)

Note: *Although there are other conditions that can result in alignment exceptions, the alignment exceptions caused by those conditions occur regardless of Endian mode. See Section 11.7.9, “Alignment Exception,” on page 11-24, for more information.*

24.4.3.5 Switching Endian Modes

Because bytes in memory are arranged differently when operating in the different Endian modes, care must be taken, when switching modes, to convert programs and data structures to the new mode. The operating system must understand the differences in the two memory formats, and must reorder the bytes in memory, as appropriate, before dispatching a new process that accesses these memory structures in the new Endian mode.

For example, if a process executing in Big Endian mode creates a data structure, and a new process executing in Little Endian mode accesses this data structure, the operating system must reverse the eight bytes within each aligned double Lword in the data structure before passing control to the new process.

24.4.3.6 Direct Memory Access in PowerPC Little Endian Mode

Another aspect of the unique arrangement of bytes used by PowerPC Little Endian mode that must be considered is that of access to memory by devices other than the IOP 480 CPU. Because these other devices, such as a direct memory access (DMA) device or another non-PowerPC processor, are not likely to handle the special address modifications associated with PowerPC Little Endian mode, they must be aware of the special arrangement of bytes used by IOP 480 CPU when they operate in Little Endian mode.

For example, if an I/O device loads a Little Endian program and data structure from a disk and places it into memory so that the IOP 480 CPU can execute the program in Little Endian mode, the I/O device must reverse the eight bytes in each aligned double Lword after reading the data from the disk and before writing it to memory. Alternatively, the operating system running on the IOP 480 CPU must understand that the program image loaded from the disk was placed into memory in true Little Endian format, in which case the operating system must rearrange the bytes before executing the program.

24.4.4 Endian Storage Attribute

The Endian storage attribute (E bit), defined in the IBM PowerPC Embedded Environment, also supports using the IOP 480 CPU in a Little Endian system. For every storage reference (instruction fetch or load/store access), an E bit is associated with the address region of the storage reference. The E bit specifies whether that region is organized as Big Endian (E = 0) or Little Endian (E = 1).

Unlike the organization of memory when using the PowerPC Little Endian mode, bytes in storage regions that are programmed as Little Endian using the E bit are arranged in true Little Endian format. Furthermore, no address modification is performed when accessing storage regions programmed as E = 1. Instead, when accessing storage regions with E = 1, the IOP 480 CPU reorders the bytes as they are transferred between the processor and memory. Unlike PowerPC Little Endian mode, the E storage attribute supports direct connections to Little Endian hardware and to memory containing Little Endian programs and data structures that may be shared with other Little Endian devices.

The on-the-fly reversal of bytes accessed in Little Endian storage regions is handled in one of two ways, depending on whether the storage access is an instruction fetch or a data (load/store) access. The following sections describe byte reversal for the two kinds of storage accesses.

24.4.4.1 Fetching Instructions from Little Endian Storage Regions

PowerPC Architecture defines instructions as aligned Lwords (four bytes) in memory. As such, instructions in a Big Endian program image are arranged with the most significant byte (MSB) of the instruction Lword at the lowest numbered address.

Consider the Big Endian mapping of instruction p at address 00, where, *for example*, p = add r7, r7, r4:

MSB			LSB
0x00	0x01	0x02	0x03

On the other hand, in a Little Endian program the same instruction is arranged with the least significant byte (LSB) of the instruction Lword at the lowest numbered address:

LSB			MSB
0x00	0x01	0x02	0x03

When an instruction is fetched from memory, the instruction must be placed in the pipeline in the proper order. Otherwise, the instruction decoder cannot recognize it. Because the PowerPC Architecture, by default, is Big Endian, the MSB of an instruction Lword is assumed to be at the lowest address. Therefore, when instructions are fetched from Little Endian storage regions, the four bytes of an instruction Lword must be reversed before the instruction is decoded. In the IOP 480 CPU, the byte reversal occurs between memory and the Instruction Cache Unit (ICU). The ICU always contains instructions in Big Endian format, regardless of whether the storage region containing the instruction was programmed as Big Endian or Little Endian. Thus, the bytes are already in the proper order when an instruction is transferred from the ICU to the decode stage of the pipeline.

If a storage region is reprogrammed from one Endian format to the other, the contents of the storage region must be reloaded with program and data structures in the appropriate Endian format. If the contents of instruction memory change, the ICU must be made coherent with the updates. The ICU must be invalidated and the updated memory contents must be fetched in the new Endian format so that the proper byte reversal (or for Big Endian, no byte reversal) occurs before the new instructions are placed in the ICU.

24.4.4.2 Accessing Data in Little Endian Storage Regions

Unlike instruction fetches from Little Endian storage regions, data accesses from Little Endian storage regions are *not* byte-reversed between memory and the DCU. Data byte ordering, in memory, depends on the data type (byte, word, or Lword) of a specific data item. It is only when moving a data item *of a specific type* from or to a GPR that it becomes known whether byte reversal is required due to the Endian format of the data item. Therefore, byte reversal during load/

store accesses is performed between the DCU and the GPR file, depending on whether the load/store was for a byte, word, or Lword.

Referring to the Big Endian and Little Endian mappings of structure s (as shown in Section 24.4.1, “Structure Mapping Examples,” on page 24-13), the differences between the byte locations of any data item in the structure depends upon the size of the particular data item. *For example* (again referring to the Big Endian and Little Endian mappings of structure s):

- The Lword a has its four bytes reversed within the Lword spanning addresses 00–03
- The word e has its two bytes reversed within the word spanning addresses 1C–1D

The array of bytes d, where each data item is a byte, is not reversed when the Big Endian and Little Endian mappings are compared. *For example*, the character “A” is located at address 14 in both the Big Endian and Little Endian mappings.

The size of the data item being loaded or stored must be known before the processor can decide whether, and if so, how to reorder the bytes when moving them between a GPR and storage.

When accessing data in a Little Endian storage region:

- For byte loads/stores, no reordering of bytes occurs
- For word loads/stores, bytes are reversed within the word
- For Lword loads/stores, bytes are reversed within the Lword

This mechanism applies, regardless of the alignment of data. *For example*, when loading a data Lword from a Little Endian storage region, all four bytes of the Lword are retrieved from memory (or the DCU). Then, the bytes are placed in the GPR so that the byte from the lowest address is placed in the LSB of the GPR.

In Little Endian storage regions, the alignment of data is treated as it is in Big Endian storage regions. Unlike PowerPC Little Endian mode, no special alignment exceptions occur when accessing data in Little Endian storage regions. Note that the alignment exceptions that apply to Big Endian region accesses also apply to Little Endian storage region accesses. See Section 11.7.9, “Alignment Exception,” on page 11-24, for detailed descriptions of conditions causing alignment exceptions.

24.4.4.3 Endian Storage Attribute Control

Control of the Endian (E) storage attribute, for a given access, depends upon whether the IOP 480 CPU is operating with the associated MSR relocation bit on or off (MSR[IR] for instruction fetches and MSR[DR] for data accesses).

In virtual mode (address translation is enabled: MSR[IR] = 1 for instruction fetches, or MSR[DR] = 1 for data accesses), the E storage attribute for an access is supplied as the E bit from the TLB entry for the page containing the addressed memory. If the E bit is 1, the page is Little Endian. Otherwise the page is Big Endian. See Section 27, “IOP 480 CPU Memory Management,” for more information about the TLB and the storage attribute control registers.

In real mode (MSR[IR] = 0 or MSR[DR] = 0), the E storage attribute, for a given access, is controlled by the Storage Little-Endian Register (SLER), which is a storage attribute control register similar to those controlling the other storage attributes.

The SLER is a 32-bit register that provides the E storage attribute for each 128-MB storage attribute control region in the 4-GB address space. The high-order five bits of the storage address select, from the SLER, the E storage attribute associated with the address region. Setting a bit to 1 in the SLER specifies that the associated storage region is Little Endian.

24.4.4.4 PowerPC Byte-Reverse Instructions

PowerPC Architecture defines byte-reverse load/store instructions, which can perform a function similar to the action taken automatically by the IOP 480 CPU when it accesses data in Little Endian storage regions using the normal load/store instructions. However, the byte-reverse load/store instructions are not as generally useful as the Endian storage attribute mechanism.

For Big Endian storage regions, the normal (non-byte-reverse) load/store instructions operate as defined in the instruction descriptions, moving the more significant bytes of the register to and from the lower-numbered memory addresses. The load/store with

byte-reverse instructions move the more significant bytes of the register to and from the higher numbered memory addresses.

The opposite is true for Little Endian storage regions, where the normal load/store instructions give the same results that load/store with byte-reverse instructions do in Big Endian storage regions. Load/store with byte-reverse instructions give the same results that normal load/store instructions do in Big Endian storage regions.

As Figure 24-2 through Figure 24-5 illustrate, a normal store to a Big Endian storage region is the same as a byte-reverse store to a Little Endian storage region, while a normal store to a Little Endian storage region is the same as a byte-reverse store to a Big Endian storage region.

Figure 24-4 illustrates the contents of a GPR and memory (starting at address 00) after a normal load/store in a Big Endian storage region.

Note that the results are identical to the results of a load/store with byte-reverse in a Little Endian storage region, as illustrated in Figure 24-3.

Figure 24-4 illustrates the contents of a GPR and memory (starting at address 00) after a load/store with byte-reverse in a Big Endian storage region.

Note that the results are identical to the results of a normal load/store in a Little Endian storage region, as illustrated in Figure 24-5.

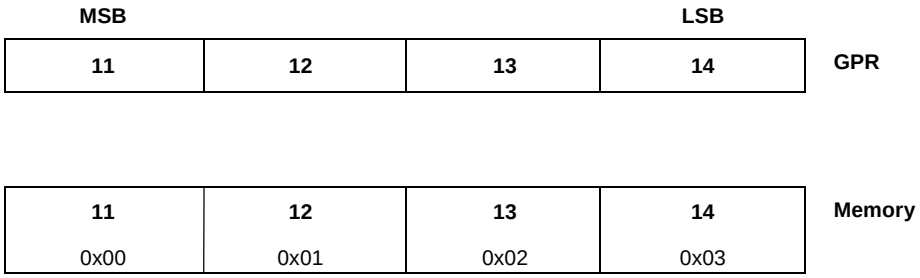


Figure 24-2. Normal Lword Load or Store (Big Endian Storage Region)

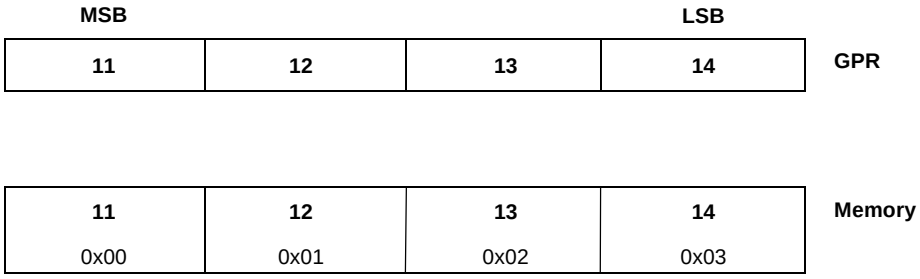


Figure 24-3. Byte-Reverse Lword Load or Store (Little Endian Storage Region)

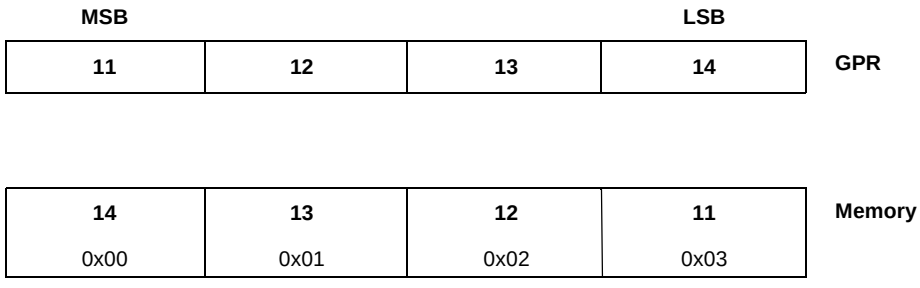


Figure 24-4. Byte-Reverse Lword Load or Store (Big Endian Storage Region)

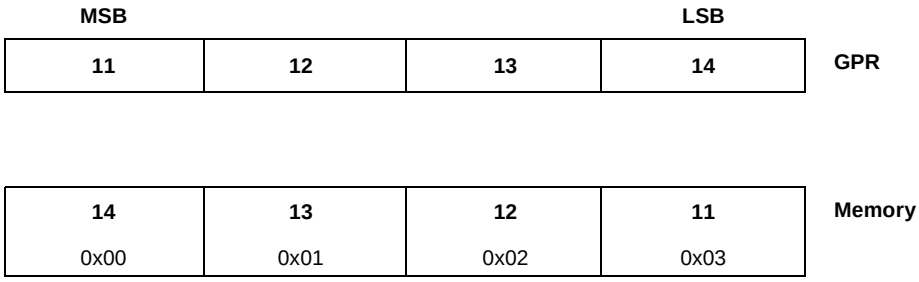


Figure 24-5. Normal Lword Load or Store (Little Endian Storage Region)

The E storage attribute augments the byte-reverse load/store instructions in two important ways:

- The load/store with byte-reverse instructions do not solve the problem of fetching instructions from a program image in true Little Endian format.

  Only the Endian storage attribute mechanism supports the fetching of true Little Endian program images.
- Typical compilers cannot make general use of the byte-reverse load/store instructions, so these instructions are ordinarily used only in special, hand-coded device drivers.

  Compilers can, however, take full advantage of the Endian storage attribute mechanism, enabling application programmers working in a high-level language, such as C, to compile programs and data structures into Little Endian format.

24.5 INSTRUCTION PROCESSING

The instruction queue, illustrated in Figure 24-6, contains three queue locations—prefetch buffer 1 (PFB1), PFB0, and decode (DCD). This queue implements a pipeline with the following functional stages: fetch, decode, and execute. Instructions are fetched from the instruction cache unit (ICU) and dispatched to the execution unit (EXU).

Instructions are fetched, at the request of the EXU, from the ICU. Cacheable instructions are forwarded directly to the instruction queue and stored in the cache. Non-cacheable instructions are also forwarded directly to the instruction queue, but are not stored in the cache. Fetched instructions drop to the empty queue location closest to the EXU. If the queue is empty, an entering instruction drops directly to DCD. PFB0 and PFB1 simply buffer instructions when the pipeline stalls.

Instructions are decoded entirely in DCD. Branches are predicted and determined during decoding. After decoding (and determination, for branch instructions), the instruction is dispatched to the execution unit (EXU), where it is executed.

24.6 BRANCHING CONTROL

The IOP 480 CPU, which provides a variety of conditional and unconditional branching instructions, uses the branch prediction techniques described in Section 24.6.5, “Branch Prediction,” on page 24-23.

24.6.1 AA Field on Unconditional Branches

The unconditional branches (**b**, **ba**, **bl**, **bla**) carry the displacement to the branch target address as a 26-bit value (the 24-bit LI field right-extended with two zeroes). This displacement is regarded as a signed 26-bit number covering an address range of ±32 MB.

For the relative (AA = 0) forms (**b**, **bl**), the target address is the Current Instruction Address (CIA, the address of the branch instruction) plus the signed displacement.

For the absolute (AA = 1) forms (**ba**, **bla**), the target address is zero plus the signed displacement. If the sign bit (LI[0]) is zero, the displacement is the target address. If the sign bit is one, the address is “below zero” and wraps to high memory. *For example*, if the displacement is 0x3FF FFFC (the 26-bit representation of negative four), the target address is 0xFFFF FFFC (zero minus four bytes, or four bytes from the top of memory).

24.6.2 AA Field on Conditional Branches

The conditional branches (**bc**, **bca**, **bcl**, **bcla**) carry the displacement to the branch target address as a 16-bit value (the 14-bit BD field right-extended with two zeroes). This displacement is regarded as a signed 16-bit number, covering an address range of ±32 KB.

For the relative (AA = 0) forms (**bc**, **bcl**), the target address is the current instruction address (CIA, the address of the branch instruction) plus the signed displacement.

For the absolute (AA = 1) forms (**bca**, **bcla**), the target address is zero plus the signed displacement. If the sign bit (BD[0]) is zero, the displacement is the target address. If the sign bit is one, the address is "below zero" and wraps to high memory. *For example*, if the displacement is 0xFFFC (the 16-bit representation of negative four), the target address is 0xFFFF FFFC (zero minus four bytes, or four bytes from the top of memory).

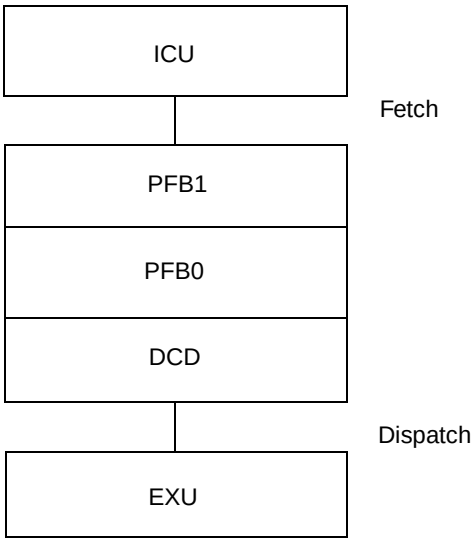


Figure 24-6. IOP 480 CPU Instruction Queue

24.6.3 BI Field on Conditional Branches

Conditional branch instructions can test one bit of the Condition Register (CR). The value of the BI field specifies the bit to be tested (bit 0–31). The content of the BI field is meaningless unless BO[0] = 0.

24.6.4 BO Field on Conditional Branches

The BO field specifies the condition under which a branch is taken, and how the branch affects the CTR.

Conditional branch instructions can test one bit in the CR. This option is selected when BO[0] = 0; if BO[0] = 1, the CR does not participate in the branch condition test. If this option is selected, the condition is satisfied (branch can occur) if CR[BI] = BO[1].

Conditional branch instructions can decrement the Count Register (CTR) by one, and after the decrement, test the CTR value. This option is selected when BO[2] = 0. If this option is selected, BO[3] specifies the condition that must be satisfied to allow a branch to be taken. If BO[3] = 0, CTR ≠ 0 is required for a branch to occur. If BO[3] = 1, CTR = 0 is required for a branch to occur.

If BO[2] = 1, the contents of CTR remain unchanged, and the CTR does not participate in the branch condition test.

Table 24-4 summarizes the usage of the bits of the BO field. BO[4] is further discussed in Section 24.6.5, “Branch Prediction,” on page 24-23.

Table 24-5 lists specific BO field contents, and the resulting actions. In Table 24-5, z represents a mandatory value of zero, and y is a branch prediction option discussed in Section 24.6.5, “Branch Prediction,” on page 24-23, .

Table 24-4. Bits of the BO Field

BO Bit	Description
BO[0]	CR Test Control 0 Test CR bit specified by BI field for value specified by BO[1] 1 Do not test CR
BO[1]	CR Test Value 0 If BO[0] = 0, test for CR[BI] = 0. 1 If BO[0] = 0, test for CR[BI] = 1.
BO[2]	CTR Test Control 0 Decrement CTR by one and test whether CTR satisfies the condition specified by BO[3]. 1 Do not change CTR, do not test CTR.
BO[3]	CTR Test Value 0 If BO[2] = 0, test for CTR ≠ 0. 1 If BO[2] = 0, test for CTR = 0.
BO[4]	Branch Prediction Reversal 0 Apply standard branch prediction. 1 Reverse the standard branch prediction.

Table 24-5. Conditional Branch BO Field

BO Value	Description
0000y	Decrement the CTR, then branch if the decremented CTR ≠ 0 and CR[BI]=0.
0001y	Decrement the CTR, then branch if the decremented CTR = 0 and CR[BI] = 0.
001zy	Branch if CR[BI] = 0.
0100y	Decrement the CTR, then branch if the decremented CTR ≠ 0 and CR[BI] = 1.
0101y	Decrement the CTR, then branch if the decremented CTR=0 and CR[BI] = 1.
011zy	Branch if CR[BI] = 1.
1z00y	Decrement the CTR, then branch if the decremented CTR ≠ 0.
1z01y	Decrement the CTR, then branch if the decremented CTR = 0.
1z1zz	Branch always.

24.6.5 Branch Prediction

Conditional branches present a problem to the fetcher. A branch might be taken; if not taken, the branch simply falls through to the next sequential instruction. The IOP 480 CPU attempts to predict whether a branch is taken before all information necessary to determine the branch direction is available. This decision is called a *branch prediction*. The fetcher can then prefetch instructions down the predicted path. If the prediction is correct, time is saved because the branched-to instruction is available in the instruction queue. Otherwise, time is lost while the correct instruction is fetched into the instruction queue. To be effective, branch prediction must be correct most of the time.

The IOP 480 CPU uses PowerPC branch prediction to minimize incorrect predictions, and enables software to reverse the standard branch prediction, which is defined as follows:

> Predict that the branch is to be taken if ((BO[0] ∧ BO[2]) ∨ s)=1
>
> where s is bit 16 of the instruction (the sign bit of the displacement for all **bc** forms, and zero for all **bclr** and **bcctr** forms).

(BO[0] ∧ BO[2]) = 1 only when the conditional branch tests nothing (the “branch always” condition). Obviously, the branch should be predicted taken for this case.

If the branch tests anything, (BO[0] ∧ BO[2]) = 0, and s entirely controls the prediction. The standard prediction for this case derives from considering the relative form of **bc**, often used at the end of loops to control the number of times that a loop is executed. The branch is taken each time the loop is executed except the last, so it is best if the branch is predicted *taken.* The branch target is the beginning of the loop, so the branch displacement is negative and s = 1. Because this situation is so common, a branch is taken if s = 1.

If branch displacements are positive, s = 0, and the branch is predicted not taken. If the branch instruction is any form of **bclr** or **bcctr** except the “branch always” forms, then s = 0, and the branch is predicted *not taken*.

There is a peculiar consequence of this prediction algorithm for the absolute forms of **bc** (**bca** and **bcla**).

As described in Section 24.6.2, “AA Field on Conditional Branches,” on page 24-22, if s = 1, the branch target is in high memory. If s = 0, the branch target is in low memory. Because these are absolute-addressing forms, there is no reason to treat high and low memory differently. Nevertheless, for the high memory case the standard prediction is taken, and for the low memory case the standard prediction is *not taken*.

BO[4] is the *prediction reversal bit*. If BO[4] = 0, the standard prediction is applied. If BO[4] = 1, the reverse of the standard prediction is applied. For the cases in Table 24-5 where BO[4] = y, software can reverse the standard prediction. This should only be done when the standard prediction is likely to be wrong. Note that for the “branch always” condition, reversal of the standard prediction is not allowed.

PowerPC Architecture requires assemblers to provide a way to conveniently control branch prediction. For any conditional branch mnemonic, a suffix may be added to the mnemonic to control prediction, as follows:

- \+ Predict branch to be taken
- – Predict branch to be not taken

For example, **bcctr+** causes BO[4] to be selected appropriately to force the branch to be predicted *taken.*

24.7 SPECULATIVE ACCESSES

The PowerPC Architecture permits implementations to perform speculative accesses to memory, either for instruction fetching, or for data loads. A speculative access is defined as any access which is not required by a sequential execution model.

For example, prefetching instructions beyond an undetermined conditional branch is a speculative fetch; if the branch is not in the predicted direction, the program, as executed, never needs the instructions from the predicted path. Similarly, in a superscalar processor that performs out-of-order execution, a program can speculatively fetch a load instruction that is past an undetermined branch.

Sometimes speculative accesses are inappropriate, however. *For example*, attempting to fetch instructions from addresses that cannot contain instructions can cause problems.To protect against errant accesses to “sensitive” memory or I/O devices, the PowerPC Architecture provides the G (guarded) storage attribute, which can be used to specify memory pages from which speculative accesses are prohibited. (Actually, speculative accesses to guarded storage are allowed in certain limited circumstances; if an instruction in a cache block is executed, the remainder of the cache block can be speculatively accessed.

24.7.1 Speculative Accesses in IOP 480 CPU

The IOP 480 CPU does not perform out-of-order execution, nor does it perform speculative loads.

The IOP 480 CPU provides two methods to enable or disable speculative instruction fetching. If address translation is enabled (MSR[IR] = 1), the G (guarded) field in each translation lookaside buffer (TLB) entry controls speculative accesses. Each TLB entry controls speculative access for a page of virtual memory, which can range in size from 1 KB–16 MB.

If address translation is disabled (MSR[IR] = 0), the Storage Guarded Register (SGR) controls speculative accesses for regions of memory. When a page is guarded (speculative fetching is disallowed), prefetching is disabled for that page. A fetch request must be completely resolved (no longer speculative) before it is issued. There is a considerable performance penalty for fetching from guarded storage, so guarding should be used only when required.

Note: *Following any reset, the IOP 480 CPU operates with all of the storage guarded.*

When address translation is enabled, an attempt to access guarded storage results in an instruction storage exception. Because the MMU provides high granularity (pages can be as small as 1 KB), fetching instructions from guarded storage should be unnecessary.

24.7.1.1 Prefetch Distance Down an Unresolved Branch Path

The fetcher speculatively accesses up to five instructions down a predicted branch path, whether taken or sequential. The unresolved branch is in the DCD stage of the instruction queue (see Section 24.5, “Instruction Processing,” on page 24-21 for a description of the instruction queue). If PFB0 and

PFB1 are full, no further speculative accesses occur. If PFB0 or PFB1 is empty, the fetcher requests the next speculative instruction from the ICU; that instruction is placed in PFB0 or PFB1. If the fetched instruction is at the end of a cache line, and if PFB1 is empty, the fetcher requests the next cache line. The instruction at the beginning of the cache line is placed in PFB1. In this case, five instructions are speculatively accessed. The fetcher can speculatively access no more than four instructions (a cache line) from the cache with a single request, assuming the speculative address is cacheable.

If the address is non-cacheable [as controlled by the Instruction Cache Cacheability Register (ICCR)], no more than two instructions are speculatively accessed.

24.7.1.2 Prefetch of Branches to Count Register and Branches to Link Register

When the fetcher predicts that a **bctr** or **blr** instruction is taken, it does not attempt to access the target address in the Count Register (CTR) or Link Register (LR) if an executing instruction updates the CTR or LR ahead of the branch in DCD in the instruction queue. (See Section 24.5, “Instruction Processing,” on page 24-21 for description of the instruction queue). The fetcher recognizes that the CTR or LR contains data from an earlier use of the CTR or LR. Such data is probably not valid.

In such cases, the fetcher does not fetch the instruction at the target address until the instruction updating the CTR or LR completes and EXU is empty; only then are the “correct” CTR or LR contents known. This prevents the fetcher from speculatively accessing a completely “random” address. When the CTR or LR contents are known to be correct, the fetcher accesses no more than five instructions down the sequential or taken path of an unresolved branch, or at the address contained in the CTR or LR.

24.7.2 Preventing Inappropriate Speculative Accesses

A memory-mapped I/O device that has a status register that is automatically reset when read provides a simple example of storage that should not be speculatively accessed. Consider a serial port that reads the receive buffer on the port and then resets the RxRdy bit in the status register. If the processor speculatively loads from this register, and an intervening branch or interrupt takes the program flow away from the code containing the load instruction and then returns, the wrong result is obtained when the status register is read again.

Similarly, if the program code is in memory “next to” the I/O device (*for example*, code goes from 0x0000 0000 to 0x0000 0FFF, and the I/O device is at 0x0000 1000), prefetching past the end of the code can “hit” the I/O device.

Guarded storage can prevent prefetching past the “end” of memory. The fetcher attempts to fetch past the last valid address, likely getting machine checks on the fetches to invalid addresses. While the machine checks do not result in an exception until the processor attempts to execute an instruction at an invalid address, some systems may suffer from the attempt to access such an invalid address. *For example*, an external memory controller might log the error.

System designers can avoid problems from speculative fetching in other ways, without using the guarded storage attributes. The remainder of this section describes ways to guard against speculative instruction fetches to sensitive addresses in unguarded memory regions.

24.7.2.1 Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction

Suppose a **bctr** or **blr** instruction follows an interrupt-causing or interrupt-returning instruction (**sc**, **rfi**, or **rfci**). The fetcher does not prevent speculatively fetching past one of these instructions. In other words, the fetcher does not treat the interrupt-causing and interrupt-returning instructions specially when deciding whether to predict down a branch path. Instructions after an **rfi**, *for example*, are considered to be on the determined branch path.

To understand the implications of this situation, consider the code sequence:

```
handler:    aaa
            bbb
            rfi
subroutine: bctr
```

When executing the interrupt handler, the fetcher does not recognize the **rfi** as a break in the program flow, and speculatively fetches the target of the **bctr**, which is really the first instruction of a subroutine that has not been called. Therefore, the CTR might contain an invalid pointer.

To protect against such a prefetch, the software should insert an unconditional branch hang (**b $**) just after the **rfi**. This prevents the hardware from prefetching the wrong “target” of the **bctr**.

Consider also the above code sequence, with the **rfi** instruction replaced by an **sc** instruction. The purpose of the system call is to initialize the CTR with the appropriate value for the **bctr** to branch to, upon return from the system call. The **sc** handler returns to the instruction following the **sc**, which can’t be a branch hang. Instead, software could put a **mtctr** just before the **sc** to load a nonsensitive address into the CTR. This address is used as the prediction address before the **sc** executes. An alternative would be to put a **mfctr** or **mtctr** between the **sc** and the **bctr**; the **mtctr** prevents the fetcher from speculatively accessing the address contained in the CTR before initialization.

24.7.2.2 Fetching Past tw or twi Instructions

The interrupt-causing instructions, **tw** and **twi**, do not require the special handling described in Section 24.7.2.1, “Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction,” on page 24-26. These instructions are typically used by debuggers, which implement software breakpoints by substituting a trap instruction for the instruction originally at the breakpoint address. In a code sequence **mtlr** followed by **blr** (or **mtctr** followed by **bctr**), replacement of **mtlr**/**mtctr** by **tw** or **twi** leaves the LR/CTR uninitialized. It would be inappropriate to fetch from the **blr**/**bctr** target address. This situation is common, and the fetcher is designed to prevent the problem.

24.7.2.3 Fetching Past an Unconditional Branch

When an unconditional branch is in DCD in the instruction queue, the fetcher recognizes that the sequential instructions following the branch are unnecessary. These sequential addresses are not accessed. Addresses at the branch target are accessed instead.

Therefore, placing an unconditional branch just before the start of a sensitive address space (*for example*, at the “end” of a memory area that borders an I/O device) guarantees that addresses in the sensitive area are not speculatively fetched.

24.7.2.4 Suggested Locations of Memory-Mapped Hardware

Table 24-6 shows two address regions of the IOP 480 CPU. Suppose a system designer can map all I/O devices and all ROM and SRAM devices, *for example*, anywhere into either region. The choices made by the designer can prevent speculative accesses to the Memory-Mapped I/O devices.

Table 24-6. Example Memory Mapping

0x7800 0000 – 0x7FFF FFFF (SGR bit 15)	128 MB Region 2
0x7000 0000 – 0x77FF FFFF (SGR bit 14)	128 MB Region 1

A simple way to avoid the problem of cacheable instruction fetches colliding with I/O devices meant to be non-cacheable would be to map all ROM and SRAM devices into Region 2, and all I/O devices into Region 1.

Thus, addresses in Region 1 should be accessed only by non-cacheable load/store instructions accessing I/O devices; no speculative fetches should occur. Region 1 could be set as Guarded in the SGR; no performance penalty would result, since by design there is no possibility of prefetching from Region 1. Accesses to Region 2 would be for code and program data. Speculative fetches in Region 2 can never access addresses in Region 1. Note that this hardware organization makes the use of the SGR to protect Region 1 redundant and optional.

The use of these regions could be reversed (code in Region 1 and I/O devices in Region 2), if Region 2 is set as Guarded in the SGR. Prefetching from the top of Region 1 could attempt to speculatively access the bottom of Region 2, but Guarding prevents a speculative access from occurring. The performance penalty is slight, under the assumption that code infrequently executes near the top of Region 1.

24.7.3 Summary

In summary, software should take the following actions to prevent speculative accesses to sensitive data areas, if the sensitive data areas are not in guarded storage:

- Protect against accesses to “random” values in the LR or CTR on **blr** or **bctr** branches following **rfi**, **rfci**, or **sc** instructions by putting appropriate instructions before or after the **rfi**, **rfci**, or **sc** instruction. See Section 24.7.2.1, “Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction,” on page 24-26.
- Protect against “running past” the end of memory into a bordering I/O device by putting an unconditional branch at the end of the memory area. See Section 24.7.2.3, “Fetching Past an Unconditional Branch,” on page 24-26.
- Recognize that a maximum of five Lwords (20 bytes) can be prefetched past an unresolved conditional branch, either down the target path or the sequential path. See Section 24.7.1.1, “Prefetch Distance Down an Unresolved Branch Path,” on page 24-24.
- Of course, software should not code branches with known unsafe targets (either instruction counter-relative or LR- or CTR-based), on the assumption that they are “protected” by guaranteeing that the unsafe direction is “not-taken.” The prefetcher can assume that if a branch “might” be taken, it is safe to fetch down the target path.

24.8 PRIVILEGED MODE OPERATION

In PowerPC Architecture, several terms describe two operating modes that have different instruction execution privileges. When a processor is “privileged mode,” it can execute all instructions in the instruction set. This mode is also called the “supervisor state.” The other mode, in which certain instructions cannot be executed, is called the “user mode,” or “problem state.” These terms are used in pairs:

Table 24-7. Instruction Execution Privileges and Operating Modes

Privileged	Nonprivileged
Privileged Mode	User Mode
Supervisor State	Problem State

The architecture uses the PR in the Machine Status Register (MSR) to controls the execution mode. When MSR[PR] = 1, the processor is in user mode (problem state); when MSR[PR] = 0, the processor is in privileged mode (supervisor state).

24.8.1 MSR Bits and Exception Handling

Attempting to execute a privileged instruction while MSR[PR] = 1 causes a privileged violation program exception (see Section 11.7.10, “Program Exceptions,” on page 11-24). The IOP 480 CPU does not execute the instruction, and the least-significant 16 bits of the program counter are loaded with 0x0700, the address of an exception processing routine.

The current value of the MSR[PR] bit is saved in the SRR1/SRR3 (along with all the other MSR bits) upon any interrupt, and the MSR[PR] bit is set to 0, in all cases. This means that all exception handlers operate in privileged mode.

The Exception Syndrome Register (ESR) distinguishes different types of program exceptions. ESR[PPR] is set when the exception was caused by a privileged exception. Software is not required to clear this ESR bit.

24.8.2 Privileged Instructions

The following instructions are privileged and cannot be executed when MSR[PR] = 1:

Table 24-8. Privileged Instructions

dcbi	
dccci	
dcread	
icbt	
iccci	
icread	
mfdcr	
mfmsr	
mfspr	For all SPRs except CTR, LR, TBHU, TBLU, XER. See Section 24.8.3
mtdcr	
mtmsr	
mtspr	For all SPRs except CTR, LR, XER. See Section 24.8.3
rfci	
rfi	
wrtee	
wrteei	

24.8.3 Privileged SPRs

All SPRs are privileged, except for the LR, the CTR, the TBHU, the TBLU, and the XER Except for moves to and from non-privileged SPRs, attempts to execute **mfspr** and **mtspr** instructions while in user mode result in privileged violation program exceptions.

In a **mfspr** or **mtspr** instruction, the 10-bit SPRN field specifies the SPR number of the source or destination SPR. The SPRN field contains two five-bit subfields, $SPRN_{0:4}$ and $SPRN_{5:9}$. The assembler handles the unusual register number encoding to generate the SPRF field. In the *machine code* for the **mfspr** and **mtspr** instructions, the SPRN subfields are *reversed* (ending up as $SPRF_{5:9}$ and $SPRF_{0:4}$) for compatibility with the POWER Architecture.

In the PowerPC Architecture, SPR numbers having a 1 in the most-significant bit of the SPRF field are privileged.

The following example illustrates how SPR numbers appear in assembler language coding and in machine coding of the **mfspr** and **mtspr** instructions.

In assembler language coding, SRR0 is SPR 26. Note that the assembler handles the unusual register number encoding to generate the SPRF field.

```
mfspr r5,26
```

When the SPR number is considered as a binary number (0b00000 11010), the most-significant bit is 0. However, the machine code for the instruction reverses the subfields, resulting in the following SPRF field: 0b11010 00000. The most-significant bit is 1; SRR0 is privileged.

When an SPR number is considered as a hexadecimal number, the second digit of the three-digit hexadecimal number indicates whether an SPR is privileged. If the second digit is odd (1, 3, 5, 7, 9, B, D, F), the SPR is privileged.

For example, the SPR number of SRR0 is 26 (0x01A). The second hexadecimal digit is odd; SRR0 is privileged. In contrast, the LR is SPR 8 (0x008); the second hexadecimal digit is not odd; the LR is nonprivileged.

24.8.4 Privileged DCRs

The **mtdcr** and **mfdcr** instructions themselves are privileged, in all cases. All DCRs are privileged.

24.9 SYNCHRONIZATION

The IOP 480 CPU supports the synchronization operations of PowerPC Architecture. The following book, chapter, and section numbers refer to related information in *The PowerPC Architecture: A Specification for a New Family of RISC Processors*:

- Book II, Section 1.8.1, “Storage Access Ordering” and “Enforce In-order Execution of I/O”
- Book III, Section 1.7, “Synchronization”
- Book III, Chapter 7, “Synchronization Requirements for Special Registers and Lookaside Buffers”

24.9.1 Context Synchronization

The context of a program is the environment (*for example*, privilege and relocation) in which the program executes. Context is controlled by the content of certain registers, such as the Machine State Register (MSR), and includes the content of all GPRs and SPRs.

An instruction or event is "context synchronizing" if it satisfies the following requirements:

1. All instructions that *precede* a context synchronizing operation must complete in the context that existed *before* the context synchronizing operation.
2. All instructions that *follow* a context synchronizing operation must complete in the context that exists *after* the context synchronizing operation.

Such instructions and events are called "context synchronizing operations." In the IOP 480 CPU, these include most interrupts and the **isync**, **rfci**, **rfi**, and **sc** instructions.

However, "context" specifically excludes the contents of memory. A context synchronizing operation does not guarantee that subsequent instructions observe the memory context established by previous instructions. To guarantee memory access ordering in the IOP 480 CPU, one must use either an **eieio** instruction or a **sync** instruction. For the IOP 480 CPU, the **eieio** and **sync** instructions are implemented identically. See Section 24.9.3, "Storage Synchronization," on page 24-31.

The contents of DCRs are not considered as part of the processor "context" managed by a context synchronizing operation. DCRs are peripherals of a processor, and are analogous to memory-mapped registers. Their context is managed in a manner similar to that of memory contents.

Finally, implementations of the PowerPC Architecture can exempt the machine check exception from context synchronization control. If the machine check exception is exempted, an instruction that *precedes* a context synchronizing operation can cause a machine check exception *after* the context synchronizing operation occurs and additional instructions have completed.

The following scenarios use psuedocode examples to illustrate these limitations of context synchronization. Subsequent text explains software can further guarantee "storage ordering."

1. Consider the following instruction sequence:

   STORE non-cacheable to address XYZ
   isync
   XYZ instruction

   In this sequence, the **isync** instruction does not guarantee that the XYZ instruction is fetched after the STORE has occurred to memory. There is no guarantee which XYZ instruction executes; either the old version or new (stored) version might.

2. This assumes that the IOP 480 CPU is part of a standard product that uses DCRs to provide bus region control using DCR:

   STORE non-cacheable to address XYZ
   isync
   MTDCR to change a bus region containing XYZ

   In this sequence, there is no guarantee that the STORE occurs before the **mtdcr** instruction changing the bus region control DCR. The STORE could fail because of a configuration error.

How can software ensure that the contents of memory and DCRs are synchronized in the instruction stream? The **eieio** instruction or the **sync** instruction perform this task. These instructions guarantee storage ordering; all memory accesses that precede **eieio** or **sync** affect the results of all subsequent memory accesses. Neither **eieio** nor **sync** guarantee that instruction prefetching follows the **eieio** or **sync**. The instructions do not cause the prefetch queues to be purged and instructions to be refetched. See Section 24.9.3, "Storage Synchronization," on page 24-31 for more information about **sync** and **eieio**.

Instruction cache state is part of context. A context synchronization operation is required to guarantee instruction cache access ordering.

Consider the following instruction sequence, which is required for self-modifying code:

STORE	Change data cache contents.
dcbst	Flush the new data cache contents to memory.
sync	Guarantee that **dcbst** completes before subsequent instructions begin.
icbi	Context changing operation; invalidates instruction cache contents.
isync	Context synchronizing operation; causes refetch using new instruction cache context text and new memory context, due to the previous sync.

Similarly, if software wishes to ensure that all storage accesses are complete before executing a **mtdcr** to change a bus region (Example 2), the software must issue a **sync** after all storage accesses and before the **mtdcr**. Likewise, if the software is to ensure that all instruction fetches after the **mtdcr** use the new bank register contents, the software must issue an **isync**, after the **mtdcr** and before the first instruction that should be fetched in the new context.

The **isync** instruction guarantees that all subsequent instructions are fetched and executed using the context established by all previous instructions. The **isync** instruction is a context synchronizing operation; **isync** causes all prefetched instructions to be discarded and refetched.

The following example illustrates the use of **isync** with debug exceptions:

mtdbcr	Set up an instruction address compare (IAC) event.
isync	Wait for the new Debug Control Register (DBCR) context to be established.
XYZ	This instruction is at the IAC address; an **isync** was necessary to guarantee that the IAC event happens at the execution of this instruction.

24.9.2 Execution Synchronization

For completeness, consider the definition of execution synchronizing as it relates to context synchronization. Execution synchronization is architecturally a subset of context synchronization.

Execution synchronization guarantees that the following requirement is met:

> All instructions that *precede* an execution synchronizing operation must complete in the context that existed *before* the execution synchronizing operation.

The following requirement need not be met:

> All instructions that *follow* an execution synchronizing operation must complete in the context that exists *after* the execution synchronizing operation.

Execution synchronization ensures that preceding instructions execute in the old context; subsequent instructions might execute in either the new or old context (indeterminate). The IOP 480 CPU provides three execution synchronizing operations: the **eieio**, **mtmsr**, and **sync** instructions.

Because **mtmsr** is execution synchronizing, it guarantees that previous instructions complete using the old MSR value. (Consider the previous example of using **mtmsr** to change the Endian mode.) However, to guarantee that subsequent instructions use the new MSR value, we have to insert a context synchronization operation, such as **isync**.

The PowerPC Architecture requires MSR[EE] (the external interrupt bit) to be, in effect, execution synchronizing: if a **mtmsr** turns on the EE bit, and an external interrupt is pending, the exception must be taken before the instruction that follows **mtmsr** is executed. However, the **mtmsr** instruction is not a context synchronizing operation, so the IOP 480 CPU does not, *for example*, discard prefetched instructions and refetch. Note that the wrtee and wrteei instructions can change the value of MSR[EE], but are not execution synchronizing.

Finally, while **sync** and **eieio** are execution synchronizing, they are also more restrictive in their requirement of memory ordering. Stating that an operation is execution synchronizing does not imply storage ordering. This is an additional specific requirement of **sync** and **eieio**.

24.9.3 Storage Synchronization

The **sync** instruction guarantees that all previous storage references complete with respect to the IOP 480 CPU before the **sync** instruction completes (therefore, before any subsequent instructions begin to execute). The **sync** instruction is execution synchronizing.

Consider the following use of **sync**:

stw	Store to I/O device.
sync	Wait for store to actually complete off chip.
mtdcr	Reconfigure device.

The **eieio** instruction guarantees the order of storage accesses. All storage accesses that precede **eieio** complete before any storage accesses that follow the instruction, as in the following example:

stb X	Store to I/O device, address X; this resets a status bit in the device.
eieio	Guarantee **stb** X completes before next instruction.
lbz Y	load from I/O device, address Y; this is the status register updated by **stb** X. **eieio** was necessary, because the read and write addresses are different, but affect one other.

The IOP 480 CPU implements both **sync** and **eieio** identically, in the manner described above for **sync**. In the PowerPC Architecture, **sync** can function across all processors in a multiprocessor environment; **eieio** functions only within its executing processor. The IOP 480 CPU is a uniprocessor; in this implementation, **sync** does not guarantee memory ordering across multiprocessors.

24.10 INSTRUCTION SET

The IOP 480 CPU instruction set contains instructions defined in the PowerPC Architecture and instructions specific to the IBM PowerPC 400 family of embedded controllers.

Section 28, “IOP 480 CPU Instruction Set,” contains detailed descriptions of each instruction, including psuedocode. Appendix A, “IOP 480 CPU Instruction Summary,” alphabetically lists each instruction and extended mnemonic and provides a short-form description. Appendix B, “IOP 480 CPU Instructions by Category,” provides short-form descriptions of instructions, grouped by the instruction categories listed in Table 24-10.

Table 24-10 also summarizes IOP 480 CPU instruction set functions by categories. Instructions within each category are described in subsequent sections.

24.10.1 Instructions Specific to IBM PowerPC Embedded Controllers

To support functions required in embedded real-time applications, the IBM PowerPC 400 family of embedded controllers defines instructions that are not defined in the PowerPC Architecture.

Table 24-9 lists the instructions specific to IBM PowerPC embedded controllers. Programs using these instructions are not portable to PowerPC implementations that are not part of the IBM PowerPC 400 family of embedded controllers.

Table 24-9. Instructions Specific to IBM PowerPC-Embedded Controllers

dccci dcread iccci icbt icread	mfdcr mtdcr rfci tlbre tlbsx tlbsx. tlbwe wrtee wrteei

24.10.2 Storage Reference Instructions

Load and store instructions transfer data between memory and the GPRs. These instructions operate on bytes, words, and Lwords. Storage reference instructions also support loading or storing multiple registers, character strings, and byte-reversed data.

Table 24-11 shows the storage reference instructions in the IOP 480 CPU.

Table 24-10. IOP 480 CPU Instruction Set Functional Summary

Storage Reference	load, store
Arithmetic and Logical	add, subtract,negate, multiply, divide, and, andc, or, orc, xor, nand, nor, xnor, sign extension, count leading zeros
Comparison	compare, compare logical, compare immediate
Branch	branch, branch conditional, branch to LR, branch to CTR
CR Logical	crand, crandc, cror, crorc, crnand, crnor, crxor, crxnor, move CR field
Rotate/Shift	rotate and insert, rotate and mask, shift left, shift right
Cache Control	invalidate, touch, zero, flush, store, read
Interrupt Control	write to external interrupt enable bit, move to/from MSR, return from interrupt, return from critical interrupt
Processor Management	system call, synchronize, trap, move to/from DCRs, move to/from SPRs, move to/from CR

Table 24-11. Storage Reference Instructions

Loads					Stores			
Byte	**Word Algebraic**	**Word**	**Multiple and String**	**Lword**	**Byte**	**Word**	**Multiple and String**	**Lword**
lbz lbzu lbzux lbzx	lha lhau lhaux lhax	lhbrx lhz lhzu lhzux lhzx	lmw lswi lswx	lwarx lwbrx lwz lwzu lwzux lwzx	stb stbu stbux stbx	sth sthbrx sthu sthux sthx	stmw stswi stswx	stw stwbrx stwu stwux stwx stwcx.

Table 24-12. Arithmetic and Logical Instructions

Arithmetic						Logical		
add add. addo addo. addc addc. addco addco. adde adde. addeo addeo.	addi addic addic. addis addme addme. addmeo addmeo. addze addze. addzeo addzeo.	divw divw. divwo divwo. divwu divwu. divwuo divwuo.	mulhw mulhw. mulhwu mulhwu. mulli mullw mullw. mullwo mullwo. neg neg. nego nego.	subf subf. subfo subfo. subfc subfc. subfco subfco. subfe subfe. subfeo subfeo.	subfic subme subme. submeo submeo. subfze subfze. subfzeo subfzeo.	and and. andc andc. andi. andis. cntlzw cntlzw.	eqv eqv. extsb extsb. extsh extsh. nand nand.	nor nor. or or. orc orc. ori oris xor xor. xori xoris

24.10.3 Arithmetic and Logical Instructions

Arithmetic operations are performed on integer or ordinal operands stored in registers. Instructions that perform operations on two operands are defined in a three-operand format; an operation is performed on the operands, which are stored in two registers. The result is placed in a third register. Instructions that perform operations on one operand are defined in a two-operand format; the operation is performed on the operand in a register and the result is placed in another register. Several instructions also have immediate formats in which an operand is a field in the instruction.

Most arithmetic and logical instructions can set the Condition Register (CR) based on the result of the instruction. The instructions having mnemonics ending in . (period) are the forms that set the CR.

Table 24-12 on page 24-32 lists the arithmetic and logical instructions in the IOP 480 CPU.

24.10.4 Compare Instructions

These instructions perform arithmetic or logical comparisons between two operands and set the CR.

Table 24-13 lists the comparison instructions in the IOP 480 CPU.

Table 24-13. Compare Instructions

Arithmetic	Logical
cmp cmpi	cmpl cmpli

24.10.5 Branch Instructions

These instruction unconditionally or conditionally branch to any address. Conditional branch instructions can test condition codes set by a previous instruction and branch accordingly. Conditional branch instructions can also decrement and test the Count Register as part of branch determination, and can save the return address in the Link Register.The target address for a branch can be a displacement from the current instruction address or an absolute address, or contained in the link or count registers.

Table 24-14 lists the branch instructions in the IOP 480 CPU.

Table 24-14. Branch Instructions

Unconditional	Conditional
b ba bl bla	bc bca bcl bcla bcctr bcctrl bclr bclrl

24.10.6 Condition Register Logical Instructions

These instructions combine the results of several comparisons without incurring the overhead of conditional branching. Code performance can significantly improve if multiple conditions are tested before a branch decision.

Table 24-15 lists the condition register logical instructions in the IOP 480 CPU.

Table 24-15. Condition Register Logical Instructions

crand crandc creqv crnand	crnor cror crorc crxor mcrf

24.10.7 Rotate and Shift Instructions

These instructions rotate or shift operands stored in the GPRs. Rotate instructions can also mask rotated operands.

Table 24-16 lists the rotate and shift instructions in the IOP 480 CPU.

Table 24-16. Rotate and Shift Instructions

Rotate	Shift
rlwimi rlwimi. rlwinm rlwinm. rlwnm rlwnm.	slw slw. sraw sraw. srawi srawi. srw srw.

24.10.8 Cache Control Instructions

These instructions indirectly control the contents of the data and instruction caches. Users can fill, flush, invalidate, and zero blocks (16-byte lines) in the data cache. Users can invalidate and fill individual lines in the instruction cache, and invalidate congruence classes in both caches.

Table 24-17 lists cache control instructions in the IOP 480 CPU.

Table 24-17. Cache Control Instructions

Data Cache	Instruction Cache
dcba dcbf dcbi dcbst dcbt dcbtst dcbz dccci dcread	icbi icbt iccci icread

24.10.9 Interrupt Control Instructions

These instructions move data between GPRs and the MSR, return from interrupts, and enable or disable maskable external interrupts.

Table 24-18 lists the interrupt control instructions in the IOP 480 CPU.

Table 24-18. Interrupt Control Instructions

mfmsr mtmsr rfi rfci wrtee wrteei

24.10.10 TLB Management Instructions

The TLB management instructions read and write entries of the TLB array in the MMU, search the TLB array for an entry which translates a given address, invalidate all TLB entries, and synchronize TLB updates with other processors.

Table 24-19 lists the TLB management instructions in the IOP 480 CPU

Table 24-19. TLB Management Instructions

tlbia tlbre tlbsx tlbsx. tlbsync tlbwe

24.10.11 Processor Management Instructions

These instructions move data between the GPRs and control registers in the IOP 480 CPU, and provide traps, system calls, and synchronization controls.

Table 24-20 lists the processor management instructions in the IOP 480 CPU.

Table 24-20. Processor Management Instructions

eieio isync sync	mcrxr mfcr mfdcr mfspr	mtcrf mtdcr mtspr sc tw twi

24.10.12 Extended Mnemonics

In addition to mnemonics for instructions supported directly by hardware, the PowerPC Architecture defines numerous *extended mnemonics*.

An extended mnemonic translates directly into the mnemonic of a hardware instruction, typically with carefully specified operands. *For example*, the PowerPC Architecture does not define a “shift right Lword immediate” instruction, because the “rotate left Lword immediate then AND with mask,” (**rlwinm**) instruction, can accomplish the same result:

rlwinm RA,RS,32–n,n,31

However, because the required operands are not obvious, the PowerPC Architecture defines an extended mnemonic:

srwi RA,RS,n

Extended mnemonics transfer the problem of remembering complex or frequently used operand combinations to the assembler, and can more clearly reflect a programmer's intentions. Thus, programs can be more readable.

See the following sections and appendices for lists of the extended mnemonics:

- Section 28, "IOP 480 CPU Instruction Set," lists extended mnemonics under the associated hardware instruction mnemonics.
- Appendix A, "IOP 480 CPU Instruction Summary," lists extended mnemonics alphabetically, along with the hardware instruction mnemonics.
- Table B-4 in Appendix B, "IOP 480 CPU Instructions by Category," lists all extended mnemonics.

25 IOP 480 CPU CACHE OPERATIONS

The IOP 480 CPU incorporate two internal caches, an Instruction Cache Unit (ICU) and a Data Cache Unit (DCU).

The ICU controls instruction accesses to main memory and, if an instruction cache array is implemented, stores frequently used instructions to reduce the overhead of instruction transfers between the instruction queue and external memory, minimizing access latency for frequently executed instructions. The DCU controls data accesses to main memory and, if a cache array is implemented, stores frequently used data to reduce the overhead of data transfers between the GPRs and external memory, minimizing access latency for frequently used data. Instructions and data can be accessed in the cache much faster than in main memory.

The ICU and DCU feature:

- Line fills in target-Lword-first, sequential, or any other order
- A separate bypass path to handle instructions and data in cache-inhibited memory, and to improve performance during line fills
- Cache line locking

The DCU features byte-writeability to improve the performance of byte and word operations, and supports write-back and write-through write strategies.

The IOP 480 CPU differ visibly in the size of their implemented cache arrays, which store cached instructions and data.

Table 25-1. Cache Array Size by Core

Core	ICU Cache Array Size	DCU Cache Array Size
PowerPC RISC	4 KB	2 KB

Section 25.1, “ICU and DCU Organization,” on page 25-1, describes the organization of the ICU and DCU.

25.1 ICU AND DCU ORGANIZATION

The ICU and DCU contain control logic and, possibly, cache arrays. The control logic, which handles data transfers between the cache units, main memory, and the RISC core, differs significantly between the ICU and DCU. The ICU and DCU cache arrays, which (when implemented) store instructions and data from main memory, respectively, are almost identical. (The DCU array adds a “dirty” bit to mark modified lines.)

The ICU and DCU cache arrays are two-way set-associative. In both cache units, a cache line can be in one of two locations in the cache array. The two locations are members of a set of locations. Each set is divided into two ways, way A and way B; a cache line can be located in either way. Each way is organized as n lines of four Lwords each, where n is the cache size, in kilobytes, multiplied by 32. For example, a 2 KB cache array contains 64 lines.

Table 25-2. ICU and DCU Cache Array Organization

Tags (Two-Way Set)		Cache Lines (Two-Way Set)	
Way A	Way B	Way A	Way B
$A_{0:m-1}$ Line 0	$A_{0:m-1}$ Line 0	Line 0	Line 0
$A_{0:m-1}$ Line 1	$A_{0:m-1}$ Line 1	Line 1	Line 1
• • •	• • •	• • •	• • •
$A_{0:m-1}$ Line *n*–2	A_{0m-1} Line *n*–2	Line *n*–2	Line *n*–2
$A_{0:m-1}$ Line *n*–1	$A_{0:m-1}$ Line *n*–1	Line *n*–1	Line *n*–1

Cache lines are addressed using a tag field and an index. The tag fields are also two-way set-associative. As shown in Table 25-2 on page 25-1, the tag fields in ways A and B store address bits $A_{0:m-1}$ for each cache line; m is the number of address bits that specify the tag field. The remaining address bits ($A_{m:27}$) serve as an index to the cache array. The two cache lines that correspond with the same line index are called a congruence class.

Table 25-3 shows the values of m and n for various cache array sizes.

Table 25-3. Cache Sizes, Tag Fields, and Lines

Cache Size	m (Tag Field Bits)	n (Cache Array Lines)
0 KB	—	—
2 KB	22	64
4 KB	21	128

When the ICU or DCU requests a cache line from main memory (an operation called a cache line fill), a least-recently-used (LRU) policy determines which cache line way receives the requested line. The index, determined by the instruction or data address, selects a congruence class. Within a congruence class, the most recently accessed line (in either way A or way B) is retained and the LRU bit in the associated tag array marks the other line as LRU. The LRU line then receives the requested instruction or data Lwords. After the cache line fill, the LRU bit is set to identify as LRU the line opposite the line just filled.

To determine a real cache array size, use an **mfspr** instruction to read the Instruction Cache Debug Data Register (ICDBDR) after a system reset, before executing an **icread** instruction.

The size information is in the following ICDBDCR fields:

0:3	ICSZ	ICU Array Size 0 KB 2 KB 4 KB 8 KB 16 KB
4:7	DCSZ	DCU Array Size 0 KB 2 KB 4 KB 8 KB 16 KB

The effective size of the cache arrays can be changed, using fields in the Cache Debug Control Register (CDBCR), to model the performances of various cache array sizes. Section 25.7, “ICU and DCU Performance Modeling,” on page 25-14, describes how effective cache array sizes are changed. Note that effective cache array sizes cannot exceed the corresponding actual cache array sizes.

25.2 ICU OVERVIEW

The ICU manages data transfers between external cacheable memory and the instruction queue in the execution unit.

Figure 25-1 shows the relationship between the ICU and instruction queue.

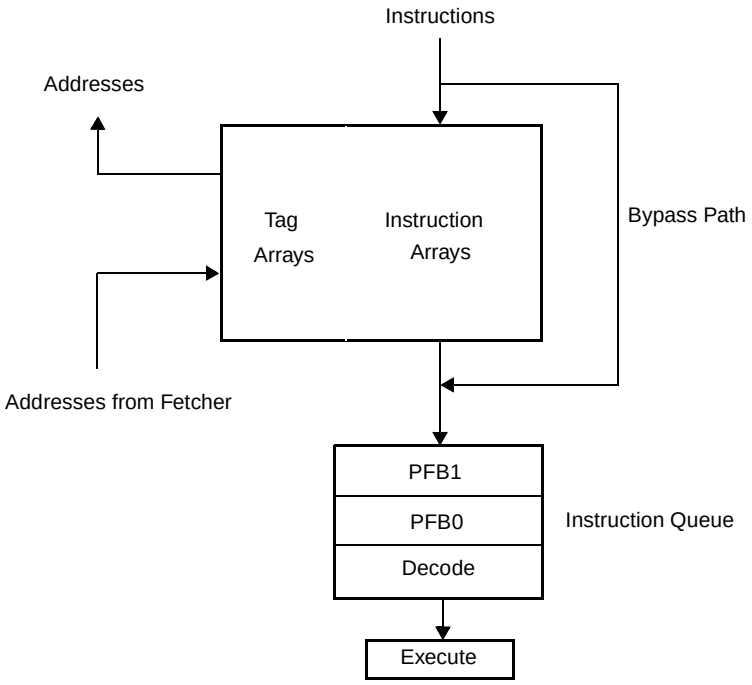


Figure 25-1. Instruction Flow

The bypass path handles instructions in cache-inhibited memory and improves performance during line fill operations. If a request from the fetcher obtains an entire line from memory, the queue does not have to wait for the entire line to reach the cache. The target Lword (the Lword requested by the fetcher) is sent on the bypass path to the queue while the line fill proceeds, even if the selected line fill order is not target-Lword-first.

Instructions from cacheable memory regions are copied into the instruction cache, from which they can be accessed by the fetcher far more quickly than they can be obtained from memory. Cache lines are loaded either target-Lword-first or sequentially, or in any order. Target-Lword-first fills start at the requested Lword, continue to the end of the line, and then wrap to fill the remaining Lwords at the beginning of the line. Sequential fills start at the first Lword of the cache line and proceed sequentially to the last Lword of the line.

Cache line fills always run to completion, even if the instruction stream branches away from the rest of the line. As requested instructions are received, they go to the fetcher before the line fills in the cache. The filled line is always placed in the ICU; if an external memory subsystem error occurs during the fill, the line is marked as invalid. During a clock cycle, the ICU can send one instruction to the fetcher.

25.2.1 Instruction Cacheability Control

When instruction address translation is disabled (the IR field of the Machine State Register (MSR) is 0), instruction cacheability is controlled by the Instruction Cache Cacheability Register (ICCR). Each bit in the ICCR (ICCR[S0:S31]) controls the cacheability of a 128 MB region (see Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13). If ICCR[Sn] = 1, caching is enabled for the specified region.

When instruction address translation is enabled (MSR[IR] = 1), instruction cacheability is controlled by the I storage attribute in the translation lookaside buffer (TLB) entry for the memory page. If TLB_entry[I] = 1, caching is inhibited; otherwise caching is enabled. Cacheability is controlled separately for each page, which can range in size from 1 KB to 16 MB. Section 27.3, “Translation Lookaside Buffer (TLB),” on page 27-2, describes the TLB.

The performance of the IOP 480 CPU is significantly lower while executing in cache-inhibited regions.

Following system reset, address translation is disabled and all ICCR bits are reset to 0 so that no memory regions are cacheable. Before regions can be designated as cacheable in the ICCR, it is necessary to execute the **iccci** instruction n times (once for each congruence class in the cache array). This invalidates all congruence classes before enabling the cache. The ICCR can then be reconfigured appropriately and the ICU can begin normal operation.

25.2.2 ICU Coherency

The ICU does not “snoop” external memory or the DCU. Programmers must follow special procedures for ICU synchronization when self-modifying code is used or if a peripheral device updates memory containing instructions.

The following code example illustrates the necessary steps for self-modifying code. This example assumes that *addr1* is both data and instruction cacheable.

```
stw     regN, addr1     # the data in regN is to become an
                        instruction at addr1
dcbst   addr1           # forces data from the data cache to
                        memory
sync                    # wait until the data actually reaches
                        the memory
icbi    addr1           # the previous value at addr1 might
                        already be in the instruction cache;
                        invalidate it in the cache
isync                   # the previous value at addr1 may
                        already have been pre-fetched into the
                        queue; invalidate the queue so that the
                        instruction must be re-fetched
```

25.2.3 DCU Overview

The DCU manages data transfers between external cacheable memory and the general-purpose registers in the execution unit.

A bypass path handles data operations in cache-inhibited memory and improves performance during line fill operations.

Data from cacheable memory regions are copied into lines in the data cache, either target-Lword-first or sequentially, or in any other order. Target-Lword-first fills start at the requested Lword, continue to the end of the line, and then wrap to fill the remaining Lwords at the beginning of the line. Sequential fills start at the first Lword of the cache line and proceed sequentially to the last Lword of the line. In both types of fills, the line is marked valid when the fourth Lword is filled.

GPRs receive the requested byte, word, or Lword of data immediately upon being received from main storage using a cache bypass mechanism. As requested data is received, it is forwarded to the target at the same time the data is written to the cache. The filled line is always placed in the DCU. The DCU can send a byte, word, or Lword to the target in two clock cycles.

Cache flushing (copying data in the cache that has been updated by the processor to main storage) and filling (loading requested data from main storage into the cache) are triggered by load, store and cache control instructions executed by the processor. Cache flushes are always sequential, starting at the first Lword of the cache block and proceeding sequentially to the end of the block.

Cache lines are always completely flushed or filled, even if the program does not request the rest of the bytes in the line, or if a bus error occurs after a bus interface unit accepts the request for the line fill. If a bus error occurs during a line fill, the line is filled and the data is marked valid. However, the line can contain invalid data, and a machine check exception occurs.

The DCU supports byte-writeability to improve the performance of byte and word store operations.

25.2.4 DCU Write Strategies

DCU operations can use write-back or write-through strategies to maintain coherency with external cacheable memory.

The write-back strategy updates only the data cache, not external memory, during store operations. Only modified data lines are flushed to external memory, and then only when necessary to free up locations for incoming lines, or when lines are explicitly flushed using **dcbf** or **dcbst** instructions. Cache flushes are always sequential, starting at the first Lword of the cache block and proceeding sequentially to the end of the block. The write-back strategy minimizes the amount of external bus activity and avoids unnecessary contention for the external bus between the ICU and the DCU.

The write-back strategy is contrasted with the write-through strategy, in which stores are written simultaneously to the cache and to external memory. A write-through strategy can simplify maintaining coherency between cache and memory. However, because the DCU cannot accept a new command until such a store completes in external memory, performance is generally slower.

When data address translation is disabled (MSR[DR] = 0), the write strategy is controlled by the Data Cache Write-thru Register (DCWR). Each bit in the DCWR (DCWR[W0:W31]) controls the write strategy of a 128 MB storage region (see Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13). If DCWR[Wn] = 0, the write-back strategy is enabled for the specified region; if DCWR[Wn] = 1, the write-through strategy is enabled.

When data address translation is enabled (MSR[IR] = 1), the write strategy is controlled by the W storage attribute in the TLB entry for the memory page. If TLB_entry[W] = 0, the write-back write strategy is selected. If TLB_entry[W] = 1, the write-through write strategy is selected. The write strategy is controlled separately for each page, which can range in size from 1 KB to 16 MB. Section 27.3, “Translation Lookaside Buffer (TLB),” on page 27-2, describes the TLB.

Note: *PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.*

The DCU can control whether a cache line is allocated in the cache on store misses. The Cache Debug Control Register (CDBCR) write-on-allocate (WOA) bit controls the allocate-on-write policy. If CDBCR[WOA] = 0, store misses cause a line fill. If CDBCR[WOA] = 1, store misses do not cause a line fill, but result in a non-cacheable store.

25.2.5 Data Cacheability Control

When data address translation is disabled (MSR[DR] = 0), data cacheability is controlled by the Data Cache Cacheability Register (DCCR). Each bit in the DCCR (DCCR[S0:S31]) controls the cacheability of a 128 MB region (see Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13). If DCCR[Sn] = 1, caching is enabled for the specified region.

When data address translation is enabled (MSR[DR] = 1), data cacheability is controlled by the I bit in the TLB entry for the memory page. If TLB_entry[I] = 1, caching is inhibited; otherwise caching is enabled. Cacheability is controlled separately for each page, which can range in size from 1 KB to 16 MB. Section 27.3, “Translation Lookaside Buffer (TLB),” on page 27-2, describes the TLB.

The performance of the IOP 480 CPU is significantly lower while executing in cache-disabled regions.

Following system reset, address translation is disabled and all DCCR bits are reset to 0 so that no memory regions are cacheable. Before regions can be designated as cacheable in the DCCR, it is necessary to execute the **dccci** instruction n times (once for each congruence class in the cache array). This invalidates all congruence classes before enabling the cache. The DCCR can then be reconfigured appropriately and the ICU can begin normal operation.

Note: *If a data block corresponding to the effective address (EA) exists in the cache, but the EA is non-cacheable, loads and stores (including **dcbz**) to that address are considered programming errors (the cache block should previously have been flushed). The only instructions that can legitimately access such an EA in the data cache are the cache management instructions **dcbf**, **dcbi**, **dcbst**, **dcbt**, **dcbtst**, **dccci**, and **dcread**.*

25.2.6 DCU Coherency

The DCU does not provide snooping. Application programs must carefully use cache-inhibited regions and cache control instructions to ensure proper operation of the cache in systems where external devices can update memory.

25.3 CACHE INSTRUCTIONS

For detailed descriptions of the instructions described in the following sections, see Section 28, “IOP 480 CPU Instruction Set.”

In the instruction descriptions, the term “block” is synonymous with cache line. A block is the unit of storage operated on by all cache block instructions.

25.3.1 ICU Instructions

The following instructions control instruction cache operations:

icbi — Instruction Cache Block Invalidate

Invalidates a cache block. If the data cache unlock exception is enabled, a data storage exception occurs, regardless of whether the target line is locked.

icbt — Instruction Cache Block Touch

Initiates a block fill, enabling a program to begin a cache block fetch before the program needs an instruction in the block. The program can subsequently branch to the instruction address and fetch the instruction without incurring a cache miss. This is a privileged-mode instruction.

iccci — Instruction Cache Congruence Class Invalidate

Invalidates a congruence class (in the IOP 480 CPU, both ways). This is a privileged-mode instruction.

icread — Instruction Cache Read

Reads either an instruction cache tag entry or an instruction Lword from an instruction cache line, typically for debugging. Bits in the CDBCR control instruction behavior (See “Cache Control and Debugging Features” on page 25-7.). This is a privileged-mode instruction.

25.3.2 IDCU Instructions

Data cache flushes and fills are triggered by load, store and cache control instructions. Cache control instructions are provided to fill, flush, or invalidate cache blocks.

The following instructions control data cache operations:

dcba — Data Cache Block Allocate

Speculatively establishes a line in the cache and marks the line as modified. If the line is not currently in the cache, the line is established and marked as modified without actually filling the line from external memory. If the line is marked as either non-cacheable or write-through, or if the target is not in the cache and the data cache lock-out exception is enabled, dcba is treated as a no-op.
If dcba references a non-cacheable address, or CDBCR[DLXE] = 1 and MSR = 1 (which would otherwise cause a data storage exception), dcba is treated as a no-op.
If dcba references a cacheable address, write-through required (which would otherwise cause an alignment exception), or CDBCR[DLXE] = 1 (which would otherwise cause a data storage exception), dcba is treated as a no-op.

dcbf — Data Cache Block Flush

Flushes a line, if found in the cache and marked as modified, to external memory; the line is then marked invalid. If the line is found in the cache and is not marked modified, the line is marked invalid but is not flushed. This operation is performed regardless of whether the address is marked cacheable. If the data cache unlock exception is enabled, a data storage exception occurs, regardless of whether the target line is locked.

dcbi — Data Cache Block Invalidate

Invalidates a block, if found in the cache, regardless of whether the address is marked cacheable. Any modified data is not flushed to memory. This is a privileged-mode instruction.

dcbst — Data Cache Block Store

Stores a block, if found in the cache and marked as modified, into external memory; the block is not invalidated but is no longer marked as modified. If the block is marked as not modified in the cache, no operation is performed. This operation is performed regardless of whether the address is marked cacheable.

dcbt — Data Cache Block Touch

Fills a block with data, if the address is cacheable and the data is not already in the cache. If the address is non-cacheable, this instruction is a no-op.

dcbtst — Data Cache Block Touch for Store

Implemented identically to the dcbt instruction in the IOP 480 CPU for compatibility with compilers and other tools.

dcbz — Data Cache Block Set to Zero

Fills a line in the cache with zeros and marks the line as modified. If the line is not currently in the cache (and the address is marked as cacheable and non-write-through), the line is established, filled with zeros, and marked as modified without actually filling the line from external memory. If the line is marked as either non-cacheable or write-through, an alignment exception results. If the target is not in the cache and the data cache lock-out exception is enabled, a data storage exception occurs.

dccci — Data Cache Congruence Class Invalidate

Invalidates a congruence class (in the IOP 480 CPU, both ways). This is a privileged-mode instruction.

dcread — Data Cache Read

Reads either a data cache tag entry or a data Lword from a data cache line, typically for debugging. Bits in the CDBCR control instruction behavior. (See “Cache Control and Debugging Features” on page 25-7.) This is a privileged-mode instruction.

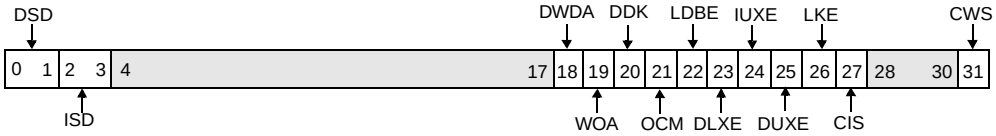
25.4 CACHE CONTROL AND DEBUGGING FEATURES

Registers and instructions are provided to control cache operation and to resolve debug cache problems. For ICU debug, the **icread** instruction and the Instruction Cache Debug Data Register (ICDBDR) are provided (see Section 25.4.1, “ICU Debugging,” on page 25-9). For DCU debug, the **dcread** instruction is provided (see Section 25.4.2, “DCU Debugging,” on page 25-10).

The CDBCR controls the behavior of the **icread** and the **dcread** instructions.

Register 25-1 illustrates the CDBCR bits.

Register 25-1. Cache Debug Control Register (CDBCR)



0:1	DSD	DCU Size Disable 00 The connected cache size is the real cache size. 01 The connected cache size is the real cache size/2. 10 The connected cache size is the real cache size/4. 11 The connected cache size is the real cache size/8.	See Table 25-5 for more information on bit settings for various real and effective cache sizes.
2:3	ISD	ICU Size Disable 00 The connected cache size is the real cache size. 01 The connected cache size is the real cache size/2. 10 The connected cache size is the real cache size/4. 11 The connected cache size is the real cache size/8.	See Table 25-5 for more information on bit settings for various real and effective cache sizes.
4:17		***Reserved***	
18	DWDA	Delayed Write Data Acknowledge 0 Normal processor write data acknowledge 1 Write data acknowledge is delayed one processor clock cycle	See Section 25.6.5, “Core Clock Frequency and Write Data Acknowledge,” on page 25-14.
19	WOA	Write-on-Allocate 0 All store misses result in a line fill. 1 Store misses do not cause a line fill, but result in a non-cacheable store.	
20	DDK	Disable Data-side Compression 0 Use K storage attribute to specify data compression 1 Disable data-side compression, regardless of K attribute	
21	OCM	Instruction-side OCM (IOCM) Mode 0 IOCM is presented only with cacheable fetches 1 IOCM is presented with cacheable and non-cacheable fetches	
22	LDBE	Load Debug Enable 0 Load data is invisible on data-side OCM. 1 Load data is visible on data-side OCM.	
23	DLXE	DCU Lock-out Exception Enable 0 DCU lock-out exception is disabled. 1 DCU lock-out exception is enabled.	
24	IUXE	ICU Unlock Exception Enable 0 ICU unlock exception is disabled. 1 ICU unlock exception is enabled.	
25	DUXE	DCU Unlock Exception Enable 0 DCU unlock exception is disabled. 1 DCU unlock exception is enabled.	
26	LKE	Lock Enable 0 Line locking is disabled. 1 Line locking is enabled.	
27	CIS	Cache Information Select 0 Information is cache data. 1 Information is cache tag.	
28:30		***Reserved***	

25.4.1 ICU Debugging

The **icread** instruction enables the reading of the instruction cache entries for the congruence class specified by $EA_{m::27}$, where m is the number of address bits in the tag field. The cache information is read into the ICDBDR; from there it can subsequently be moved, using a **mfspr** instruction, into a GPR.

Register 25-2 illustrates the ICDBDR bits.

If CDBCR[CIS] = 0, the data is an Lword of ICU data from the addressed line, specified by $EA_{28:29}$. If CDBC[CWS] = 0, the data is from the A-way; otherwise; the data from the B-way.

If CDBCR[CIS] = 1, the cache information is the cache tag. If CDBCR[CWS] = 0, the tag is from the A-way; otherwise, the tag is from the B-way.

ICU tag information is placed into the ICDBDR as shown.

Register 25-2. Instruction Cache Debug Data Register (ICDBDR)

0:31		Instruction cache information from **icread**	

0:*m*–1	TAG	Cache Tag	See Table 25-1 for information on the size of this variable-length field.
m:24		***Reserved***	The size of this field depends on the size of the tag field.
25	LK	Cache Line Lock 0 Unlocked 1 Locked	
26		***Reserved***	
27	V	Cache Line Valid 0 Not valid 1 Valid	
28:30		***Reserved***	
31	LRU	Least Recently Used (LRU) 0 A-way LRU 1 B-way LRU	

The instruction pipeline does not wait for data from an **icread** instruction to arrive before attempting to use the contents the ICDBCR.

The following code sequence ensures proper results:

- icread r5,r6# read cache information
- isync# ensure completion of icread
- mficdbdr r7# move information to GPR

25.4.2 DCU Debugging

The **dcread** instruction provides a debugging tool for reading the data cache entries for the congruence class specified by $EA_{m:27}$, where m is the number of address bits in the tag field. The cache information is read into a GPR.

If CDBCR[CIS] = 0, the data is an Lword of DCU data from the addressed line, specified by $EA_{28:29}$. If CDBC[CWS] = 0, the data is from the A-way; otherwise; the data is from the B-way.

If CDBCR[CIS] = 1, the cache information is the cache tag. If CDBC[CWS] = 0, the tag is from the A-way; otherwise the tag is from the B-way.

DCU tag information is placed into the GPR as shown:

0:*m*–1	TAG	Cache Tag	Tag size is determined by CDBCR[DSD]. See Section 25.7, “ICU and DCU Performance Modeling,” on page 25-14, for more information.
m:24		***Reserved***	The size of this field depends on the size of the TAG field.
25	LK	Cache Line Locked 0 Unlocked 1 Locked	
26	D	Cache Line Dirty 0 Not dirty 1 Dirty	
27	V	Cache Line Valid 0 Not valid 1 Valid	
28:30		***Reserved***	
31	LRU	Least Recently Used (LRU) 0 A-way LRU 1 B-way LRU	

Note: *A “dirty” cache line is one which has been accessed by a store instruction after it was established, and can be inconsistent with external memory.*

25.5 CACHE LINE LOCKING

Lines in the ICU and DCU can be locked. Locked lines remain in the cache arrays until they are explicitly unlocked. Locking frequently used instructions and data in the respective cache units can ensure that the instructions and data are available as quickly as possible.

If a line is locked in a congruence class in the ICU or DCU, that line does not participate in the LRU replacement for that congruence class. Typically, on a cache miss, the LRU unlocked line is replaced.

Except for the **dcba**, **dcbt**, **dcbtst**, **dcbz**, and **icbt** instructions, a load, store, or cache instruction that misses in the cache, and references a locked-out congruence class (a congruence class in which both lines are locked), behaves as if the instruction references non-cacheable storage. The **dcbt**, **dcbtst**, and **icbt** instructions do not update the cache in this case, but the PLB appears to have performed a line fill. The **dcba** and **dcbz** instructions behave as described in Section 25.5.2, “Unlocking Lines in the ICU and DCU,” on page 25-11.

25.5.1 Locking Lines in the ICU and DCU Cache Arrays

Line locking is controlled by the CDBCR[LKE] bit (see Figure 25-1). To lock lines in the cache units, an instruction sequence, executing in Privileged mode, sets CDBCR[LKE] = 1 to enable locking, and establishes cache lines as needed, using a **dcba**, **dcbt**, **dcbtst**, **dcbz**, or **icbt** instruction as appropriate to establish and lock each line. Locked lines are marked as locked in the appropriate cache tag array. The instruction sequence can then set CDBCR[LKE] = 0, disabling line locking, and return the processor to user mode, making the locked lines available to application code.

Some instructions that establish cache lines in the DCU are non-privileged. However, the enabling and disabling of cache line locking requires the use of the privileged **mtcdbcr** instruction (actually, **mtcdbcr** is an extended mnemonic for the **mtspr** instruction). Note that cache line locking should be performed thoughtfully; limiting the cache line locking window in a privileged code sequence as described is recommended.

Note: *The* **mtcdbcr** *extended mnemonic is not context- or execution-synchronizing. Software must place appropriate synchronization instructions before and after an* **mtcdbcr** *to ensure that the locking instructions execute in the proper context.*

25.5.2 Unlocking Lines in the ICU and DCU

Several instructions unlock locked cache lines: the privileged instructions **dccci**, **dcbi**, and **iccci**; and the non-privileged instructions **dcba**, **dcbf**, **dcbz**, and **icbi**.

In user mode, the behavior of the non-privileged unlocking cache instructions depends upon the settings of the CDBCR[DLXE], CDBCR[IUXE], and CDBCR[DUXE] bits. The following bits are associated with data cache instructions:

- **dcba**

  The CDBCR[DLXE] field (DLXE stands for Data-cache Lock-out Exception Enable) controls whether an attempt to replace an locked data cache line causes dcba to be treated as a no-op.

  Assume that the dcba instruction hits in the cache, that the W (write-through) and I (cacheability) storage attributes for the target line are 0, and that the target line is a valid cache line. Assume further that the target line is locked.

  If the DLXE field is enabled (CDBCR[DLXE] = 1), dcba is treated as a no-op.

  If the DLXE field is disabled (CDBCR[DLXE] = 0), the target line is unlocked and removed. A new line is established. If CDBCR[LKE] = 1, the newly established line is locked.

- **dcbf**

  The CDBCR[DUXE] field (DUXE stands for Data-cache Unlock Exception Enable) controls whether an attempt to flush a locked data cache line causes an exception.

  If the field is disabled (CDBCR[DUXE] = 0), the target line is flushed and unlocked, and no exception is taken. If the field is enabled (CDBCR[DUXE] = 1), **dcbf** causes a data storage exception. The exception occurs regardless of whether the target line is locked.

- **dcbz**

  The CDBCR[DLXE] field controls whether an attempt to replace an locked data cache line causes an exception.

Assume that the **dcbz** instruction hits in the cache, that the W and I storage attributes for the target are 0, and that the target is a valid cache line. Assume further that the target line is locked.

If the DLXE field is enabled (CDBCR[DLXE] = 1), a data storage exception occurs if the target is not in the cache.The exception occurs regardless of whether the target congruence class is locked-out.

If the DLXE field is disabled (CDBCR[DLXE] = 0), the target line is unlocked and removed. A new line is established. If CDBCR[LKE] = 1, the newly established line is locked. No exception is taken.

If **dcbz** references a non-cacheable address (alignment exception), and CDBCR[DLXE] = 1 (data storage exception), the alignment exception takes priority.

If **dcbz** references a cacheable address, write-through required (alignment exception), and CDBCR[DLXE] = 1 (data storage exception), the data storage exception takes priority.

- **icbi**

  The CDBCR[IUXE] field (IUXE stands for Instruction-cache Unlock Exception Enable) controls whether an attempt to invalidate a locked instruction cache line causes an exception.

  If the field is disabled (CDBCR[IUXE] = 0), the target line is invalidated and unlocked, and no exception is taken. If the filed is enabled (CDBCR[IUXE] = 1), a data storage exception occurs. The exception occurs regardless of whether the target line is locked.

  In Privileged mode, no data storage exceptions occur when the privileged and non-privileged cache instructions execute, regardless of the setting of the locked-out and unlocked exception bits in the CDBCR.

- **dcba**

  dcba replaces the LRU unlocked line in a congruence class, regardless of whether the congruence class is locked out.

- **dcbz**

  dcbz replaces the LRU unlocked line in a congruence class, regardless of whether the congruence class is locked out.

25.6 DCU PERFORMANCE

DCU performance depends upon the application and the design of the attached external bus controller, but, in general, cache hits complete in two cycles without stalling the CPU pipeline. Under certain conditions and limitations of the DCU, the pipeline stalls (stops executing instructions) until the DCU completes current operations.

Several factors affect DCU performance, including:

- Pipeline stalls
- DCU priority
- Simultaneous cache operations
- Sequential cache operations

25.6.1 Pipeline Stalls

The CPU issues commands for cache operations to the DCU. If the DCU can immediately perform the requested cache operation, no pipeline stall occurs. In some cases, however, the DCU cannot immediately perform the requested cache operation, and the pipeline stalls until the DCU can perform the pending cache operation.

A load miss or a load to a storage region marked as non-cacheable always stalls the pipeline until the load is aborted or the requested load data becomes available to the CPU, enabling it to resume instruction execution.

Other pipeline stalls occur when the CPU issues a command for a cache operation while one of the following cache operations, previously requested, is in progress:

- A line fill resulting from a store
- Two pending line flushes
- A **dcbt** instruction
- A store to a region of memory marked as write-through
- On-chip memory (OCM) presenting the DSOCM_dcuStoreAck_Hold signal

A store miss stalls the pipeline only when the CPU issues a request for a cache operation while the line fill resulting from the store miss is in progress. When the line fill finishes, instruction execution resumes.

Multiple line flushes can stall the pipeline. The DCU can flush up to two lines to memory before stalling the pipeline when the CPU issues another command for a cache operation before the second line flush begins.

When a **dcbt** instruction results in a cache miss and is followed by another cache operation, the pipeline stalls until the line fill resulting from the **dcbt** miss finishes. Then, instruction execution resumes.

When a storage region is marked as write-through, all stores, when followed immediately by another cache operation, stall the pipeline until the cycle after the PLB acknowledges the store data. This results in at least a one-cycle delay to complete the write through. However, if a store to a storage region marked as write-through misses, and the device supplying the data for the resulting line fill has a store queue, the store queue holds the store data until the line fill finishes. In this case, the write-through appears to complete before the line fill.

The pipeline stalls when OCM (if implemented in the processor core) asserts DSOCM_dcuStoreAck_Hold for loads/stores. For loads that are held, the pipeline stalls until the load is completed or aborted. If a store is held and an immediately subsequent cache operation is requested, the pipeline stalls until the store finishes and OCM deasserts DSOCM_dcuStoreAck_Hold.

25.6.2 Cache Operation Priorities

The DCU uses a priority signal, DCU_plbPriority, to improve performance when pipeline stalls occur. When the pipeline is stalled because of a data cache operation, the DCU asserts the priority signal to the external bus. DCU_plbPriority tells the external bus that the DCU requires immediate service, and is valid only when the data cache is requesting access to the PLB interface. DCU_plbPriority is asserted for all loads that require external data, or when the data cache is requesting the external bus and stalling an operation that is being presented to the data cache.

Table 25-4 on page 25-14 provides examples of when priority is asserted and de-asserted.

25.6.3 Simultaneous Cache Operations

Some cache operations can occur simultaneously to improve DCU performance. For example, when a line must be allocated in the data cache and another line flushed to memory, the operations can be performed simultaneously. A line fill caused by a store miss to a storage region marked as write-through, and a load to a non-cacheable storage region marked followed by a line flush, can also be performed simultaneously. Support for simultaneous operations depends upon the configuration of the external bus.

25.6.4 Sequential Cache Operations

Some common cache operations, when performed sequentially, can limit DCU performance: sequential loads/stores to non-cacheable storage regions, sequential line fills, and sequential line flushes.

In the case of sequential cache hits, the most commonly occurring operations, the DCU loads or stores data every two cycles. In such cases, the DCU does not affect performance.

However, when a load from a non-cacheable storage region is accepted by the PLB in the same cycle as the load request, and the load data is presented in the next cycle, three cycles are required to complete the load.

The following equation determines the number of cycles required to complete a series of sequential loads from non-cacheable storage regions:

$$\text{Total cycles} = (2 + (car + cgd)) \times \text{Number of loads}$$

where *car* is the number of cycles required to accept the load request (if the request is accepted in the same cycle, *car* = 0) and *cgd* is the number of cycles required to get the load data after the request is accepted.

Similarly, when a store to a non-cacheable storage region is accepted and the store data is presented in the same cycle, at least three cycles are required to complete the store.

Table 25-4. Priority Changes with Different Data Cache Operations

Instruction	Priority	Next Instruction	Priority
Any load	1	N/A	N/A
Any store	0	Any other cache operation when the line fill was not accepted on the PLB interface	1
dcbf	0	Any cache hit	0
dcbf/dcbst	0	Load non-cache when requesting access to the PLB interface	1
dcbf/dcbst	0	Any line fill when **dcbf/dcbst** requests access to the PLB interface	1
dcbf/dcbst	0	**dcbf/dcbst** when the first **dcbf/dcbst** requests access to the PLB interface	1
dcbt	0	Any cache hit	0
dcbt	0	Any other cache operation when a line fill was not accepted on the PLB interface	1
dcbi/dccci/dcbz	0	N/A	N/A

The following equation determines the number of cycles required to complete a series of sequential stores to storage regions marked as non-cacheable:

$$\text{Total cycles} = (3 + (car + cad)) \times \text{Number of stores}$$

where *car* is the number of cycles required to accept the store request (if the request is accepted in the same cycle, *car* = 0) and *cad* is the number of cycles required to acknowledge the store data after the request is accepted.

Sequential line fills can limit DCU performance. Line fills occur when a load/store or **dcbt** instruction misses in the cache. Sequential line flushes from the DCU to main memory also limit DCU performance. Flushes occur when a line fill replaces a valid line that is marked dirty (modified), or when a **dcbf** instruction flushes a specific line. If two flushes are pending, the DCU stalls any new data cache operations until the first flush finishes and the second begins.

25.6.5 Core Clock Frequency and Write Data Acknowledge

To sustain performance at the maximum core clock frequency for some bus interface designs, CDBCR[DWDA] must be set to 1, which delays write data acknowledge by one core clock cycle. This, in turn, causes line flushes and non-cacheable stores to take an additional core clock cycle.

Setting CDBCR[DWDA] = 1 relaxes the timing requirements for the PLB_dcuWrDAck signal, enabling system designs to sustain the maximum core clock frequency at the expense of slower line flushes and non-cacheable stores even when a bus interface design cannot meet the timing requirements for PLB_dcuWrDAck. When CDBCR[DWDA] = 1, PLB_dcuWrDAck is latched before being used by the DCU control logic.

25.7 ICU AND DCU PERFORMANCE MODELING

The effective size of the ICU and DCU can be changed to model how various cache sizes affect performance, depending on the settings of the CDBCR[ISD] and CDBCR[DSD] fields.

To model smaller cache sizes, the size of the tag field (specified by m) is increased; one or more bits that were formerly part of the index are treated as bits in the tag field. The number of bits treated differently depends upon the desired cache size. Note that if the effective cache size and the real cache size are the same, tag and index bits are not treated differently.

Table 25-5 shows how the CDBCR[ISD] and CDBCR[DSD] fields determine effective ICU and DCU sizes, respectively. An “x” in the table indicates a don’t care.

Table 25-5. CDBCR[DSD], CDBCR[ISD], and Effective Cache Size

Real Cache Size	CDBCR[DSD]	CDBCR[ISD]	Effective Cache Size	Tag Field	Index	*m*
16 KB	00	00	16 KB	0:18	19:27	19
	01	01	8 KB	0:19	20:27	20
	10	10	4 KB	0:20	21:27	21
	11	11	2 KB	0:21	22:27	22
8 KB	00	00	8 KB	0:19	20:27	20
	01	01	4 KB	0:20	21:27	21
	1x	1x	2 KB	0:21	22:27	22
4 KB	00	00	4 KB	0:20	21:27	21
	x1, 1x	x1, 1x	2 KB	0:21	22:27	22
2 KB	xx	xx	2 KB	0:21	22:27	22

26 IOP 480 CPU DEBUGGING AND JTAG FACILITIES

The debug facilities of the IOP 480 CPU include support for:

- Debug modes for debugging during hardware and software development
- Debug events that allow developers to control the debug process

Debug registers control the debug modes and debug events. The debug registers are accessed through software running on the processor or through a JTAG debug port. The debug interface is the JTAG debug port. The JTAG debug port can also be used for board test.

The debug modes, events, controls, and interface provide a powerful combination of debug facilities for a wide range of hardware and software development tools.

26.1 DEVELOPMENT TOOL SUPPORT

The OS Open Real-Time Operating System Debugger product from IBM is an example of an operating system-aware debugger, implemented using software traps.

The RISCWatch product from IBM is an example of a development tool that uses the external debug mode, debug events, and the JTAG debug port to implement a hardware and software development tool.

26.2 DEBUG MODES

There are two debug modes in the IOP 480 CPU; each supports a type of debug tool commonly used in embedded systems development:

- Internal debug mode, which supports ROM monitors
- External debug mode, which supports emulators

Both modes can be enabled simultaneously; the Debug Control Register (DBCR) controls the internal and external debug modes.

26.2.1 Internal Debug Mode

Internal debug mode supports access to all architected processor resources, setting hardware and software breakpoints, and monitoring processor status. In this mode, debug events generate debug exceptions, which can interrupt normal program flow so that monitor software can collect processor status and alter processor resources.

Internal debug mode relies on exception handling software at a dedicated interrupt vector and an external communications path to debug software problems. This mode, used while the processor executes instructions, enables debugging of operating system or application programs.

In this mode, debugger software is accessed through a communications port, such as a serial port, external to the processor core.

26.2.2 External Debug Mode

External debug mode provides access to all architected processor resources and supports stopping, starting, and stepping the processor, setting hardware and software breakpoints, and monitoring processor status. In this mode, debug events cause the processor to become architecturally frozen. While the processor is frozen, normal instruction execution stops and all architected processor resources can be accessed and altered. External bus activity continues in external debug mode.

The JTAG mechanism can also pass instructions to the processor for execution, allowing a JTAG debug tool to display and alter processor resources, including memory.

The JTAG mechanism prevents the occurrence of a privileged exception when a privileged instruction is executed while the processor is in user mode.

Storage access control by a memory management unit (MMU) remains in effect while in external debug mode; the debugger may need to modify MSR or TLB values to access protected memory.

External debug mode relies only on internal processor resources, so it can be used to debug both system hardware and software problems. This mode can also be used for software development on systems without a control program, or to debug control program problems.

Access to the processor, while in external debug mode, is through the JTAG debug port.

26.3 PROCESSOR CONTROL

The IOP 480 CPU provides the following debug facilities for processor control. Not all facilities are available in all debug modes.

Instruction Step Processor is stepped one instruction at a time, while stopped, using JTAG debug port.

Instruction Stuff While the processor is stopped, instructions can be stuffed into the processor and executed using JTAG debug port.

Halt Processor can be stopped by activating an external halt signal on an external event, such as a logic analyzer trigger. This signal freezes the processor architecturally. While frozen, normal instruction execution stops and all architected processor resources can be accessed and altered using JTAG debug port. Normal execution resumes when the halt signal is deactivated.

Stop Processor can be stopped using JTAG debug port. Activating a stop causes the processor to become architecturally frozen. While frozen, normal instruction execution stops and architected processor resources can be accessed and altered using JTAG debug port. Normal execution resumes when this bit is reset.

Reset External reset signal, JTAG debug port, or Debug Control Register (DBCR) can request a reset of the processor. JTAG debug port can request core, chip, and system resets.

Debug Events A debug event triggers a debug operation. The operation depends on the debug mode. For more information and a list of debug events, see Section 26.5, “Debug Events,” on page 26-2.

Freeze Timers JTAG debug port or the DBCR can control timer resources. The timers can be enabled to run, freeze always, or freeze on a debug event.

Trap Instructions The trap instructions **tw** and **twi** can be used, with debug events, to implement software breakpoints.

26.4 PROCESSOR STATUS

The processor execution status, exception status, and most recent reset can be monitored.

Execution Status JTAG debug port can monitor processor execution status to determine whether the processor is stopped, waiting, or running.

Exception Status JTAG debug port can monitor the status of pending synchronous exceptions.

Most Recent Reset JTAG debug port can read the Debug Status Register (DBSR) to determine the type of the most recent reset.

26.5 DEBUG EVENTS

Debug events, enabled by the DBCR and recorded in the DBSR, trigger debug operations. A debug event occurs when an event listed in Table 26-1 on page 26-3 is detected. The debug operation is performed after the debug event.

In internal debug mode, the processor generates a debug exception when a debug event occurs. In external debug mode, the processor stops when a debug event occurs. When internal and external debug mode are both enabled, the processor also stops on any debug event. When external and internal debug mode are both disabled, debug events are recorded in the DBSR, but no action is taken.

26.6 DEBUG REGISTERS

Several debug registers, available to debug tools running on the processor, are not intended for use by application code. Debug tools control debug resources such as debug events. Application code that uses debug resources can cause the debug tools to fail, as well as other unexpected results, such as program hangs and processor resets.

Application code should not use the debug resources, including the debug registers.

26.6.1 Debug Control Register (DBCR)

DBCR can enable debug events, reset the processor, control timer operation during debug events, enable JTAG interrupts, and set the processor debug mode.

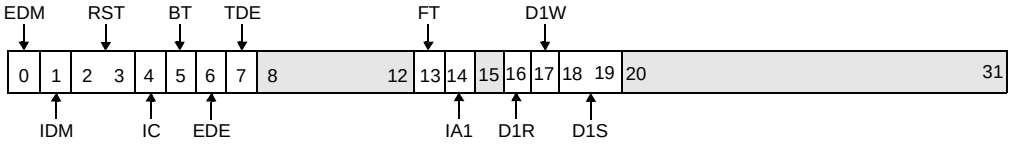
Application code should not use DBCR.

Table 26-1lists debug events. Register 26-1 illustrates the DBCR bits.

Table 26-1. Debug Events

Event	Description
Exception	This debug event occurs after an exception. Exception debug events always include the non-critical class of exceptions unless in internal debug mode. When only internal debug mode is enabled, the critical class of exceptions are not included. This debug event is disabled if the Machine State Register (MSR) field MSR[DE] = 0 and the DBCR field DBCR[IDM] = 1.
Branch Taken	This debug event occurs before the execution of a branch instruction that is determined to be taken. This debug event is disabled if MSR[DE] = 0 and DBCR[IDM] = 1.
Data Address Compare (DAC)	This debug event occurs before the execution of an instruction that accesses a data address that matches the contents of the Data Address Compare Register (DAC1). The DBCR can set the DAC to occur on reads/writes from byte addresses, or from any byte within Word, Lword, or Qword addresses.
Instruction Address Compare (IAC)	This debug event occurs before the execution of an instruction at an address that matches the contents of the Instruction Address Compare Register (IAC1).
Instruction Completion	This debug event occurs after the completion of any instruction. This debug event is disabled if MSR[DE] = 0 and DBCR[IDM] = 1.
Trap	This debug event occurs before the execution of a trap instruction where the conditions are such that the trap occurs.
Unconditional	This debug event occurs immediately upon being set by the JTAG debug port or the XXX_cpuUncondDebugEvent signal. This signal supports user-defined debug events, using ASIC logic external to the IOP 480 CPU.

Register 26-1. Debug Control Register (DBCR)



0	EDM	External Debug Mode 0 Disable 1 Enable	
1	IDM	Internal Debug Mode 0 Disable 1 Enable	
2:3	RST	Reset 00 No action 01 Core reset 10 Chip reset 11 System reset	
		Note: *Writing 01, 10, or 11 to this field causes a processor reset request.*	
4	IC	Instruction Completion Debug Event 0 Disable 1 Enable	Instruction completion does not cause a debug event if MSR[DE] = 0 in internal Debug mode
5	BT	Branch Taken Debug Event 0 Disable 1 Enable	Branch taken does not cause a debug event if MSR[DE] = 0 in internal Debug mode
6	EDE	Exception Debug Event 0 Disable 1 Enable	Critical exceptions do not cause debug events if MSR[DE] = 0 in internal Debug mode
7	TDE	TRAP Debug Event 0 Disable 1 Enable	
8:12		***Reserved***	
13	FT	Freeze timers on debug event 0 Free-run timers 1 Freeze timers	
14	IA1	Instruction Address Compare 1 Enable 0 Disable 1 Enable	
15		***Reserved***	
16	D1R	DAC Read Enable 0 Disable 1 Enable	
17	D1W	DAC Write Enable 0 Disable 1 Enable	
18:19	D1S	DAC Size 00 Compare all bits 01 Ignore the least significant bit (LSB) 10 Ignore the two LSBs 11 Ignore the four LSBs	 Exact address compare Byte within word address compare Byte within Lword address compare Qword address compare
20:31		***Reserved***	

26.6.1.1 DAC Compare Size Field (DBCR[D1S]) Note

The data address compare (DAC) debug event can be set to react to any byte in a larger block of memory, in addition to reacting to an exact address match. The DAC Compare Size field (DBCR[D1S]) allows DAC debug events to react to any byte in a word, Lword, or line of four words (Qword).

The address for a DAC is the effective address (EA) of a storage reference instruction.

As an example, suppose that a DAC debug event should react to byte 3 of an Lword-aligned target. A DAC set for exact compare would not recognize a reference to that byte by load/store Lword or load/store word instructions, because the byte address is not the EA of such instructions. In such a case, the D1S field must be set for a wider capture range (for example, to ignore the two least significant bits (LSBs) if Lword operations to the misaligned byte are to be detected). The wider capture range results in excess debug events (events that are within the specified capture range, but reflect byte operations in addition to the desired byte). Such excess debug events must be handled by software.

26.6.2 Debug Status Register (DBSR)

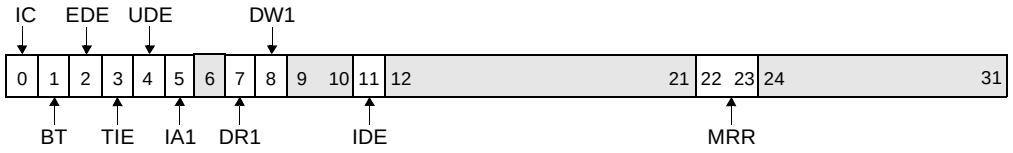
The DBSR contains status on debug events and the most recent reset; the status is obtained by reading the DBSR. The status bits are normally set by debug events or by any of the three reset types.

Clearing the DBSR fields is performed by writing an Lword to the DBSR, using the **mtdbsr** extended mnemonic, having a 1 in all bit positions to be cleared and a 0 in the all other bit positions. The data written to the DBSR is not direct data, but a mask. A 1 clears the bit and a 0 has no effect.

Application code should not use the DBSR.

Register 26-2 illustrates the DBSR bits.

Register 26-2. Debug Status Register (DBSR)



0	IC	Instruction Completion Debug Event 0 Event didn't occur 1 Event occurred
1	BT	Branch Taken Debug Event 0 Event didn't occur 1 Event occurred
2	EDE	Exception Debug Event 0 Event didn't occur 1 Event occurred
3	TIE	TRAP Instruction Debug Event 0 Event didn't occur 1 Event occurred
4	UDE	Unconditional Debug Event 0 Event didn't occur 1 Event occurred
5	IA1	IAC1 Debug Event 0 Event didn't occur 1 Event occurred
6		***Reserved***
7	DR1	DAC Read Debug Event 0 Event didn't occur 1 Event occurred
8	DW1	DAC Write Debug Event 0 Event didn't occur 1 Event occurred
9:10		***Reserved***
11	IDE	Imprecise Debug Event 0 Event didn't occur 1 Event occurred
12:21		***Reserved***
22:23	MRR	Most Recent Reset 00 No reset has occurred since last cleared by software. 01 Core reset 10 Chip reset 11 System reset This field is set to a value, indicating the type of reset, when a reset occurs.
24:31		***Reserved***

26.6.3 Data Address Compare Register (DAC1)

The IOP 480 CPU can take a debug event upon storage or cache references to an address specified in the DAC register. The address in the DAC is the address of an operand (effective address, or EA) of a storage reference or cache instruction.The fields DBCR[D1R] and DBCR[D1W] control the DAC Read and DAC Write debug events, respectively.

The address in DAC specifies an exact byte EA for a DAC debug event. However, one may want to take a debug event on any byte within a word (*that is*, ignore the least significant bit (LSB) of the DAC); on any byte within an Lword (that is, ignore the two LSbs of DAC), or on any byte within a line of four Lwords (*that is*, ignore four LSbs of DAC). The DBCR[D1S] field controls the addressing options.

Register 26-4 illustrates the DAC1 bits.

Register 26-3. Data Address Compare Register (DAC1)

0:31		Data Address Compare (DAC) byte address	DBCR[D1S] determines byte, word, or Lword usage.

26.6.3.1 Data Address Compare (DAC) Applied to Cache Instructions

Some cache instructions may cause DAC debug events. There are several special cases.

Table 26-2 summarizes possible DAC debug events by cache instruction.

Architecturally, the dcbi and dcbz instructions are “stores.” These instructions can change data, or cause the loss of data by invalidating a dirty line. Therefore, they can cause DAC-Write debug events.

The **dccci** instruction also could be considered a “store” because it can change data by invalidating a dirty line. However, **dccci** is not address-specific; it affects an entire congruence class regardless of the operand address of the instruction. Because it is not address-specific, **dccci** does not cause DAC-Write debug events.

Table 26-2. DAC Applied to Cache Instructions

Instruction	Possible DAC Debug Event	
	DAC-Read	DAC-Write
icbi	Yes	No
icbt	Yes	No
iccci	No	No
icread	No	No
dcba	No	Yes
dcbf	No	Yes
dcbi	No	Yes
dcbst	No	Yes
dcbt	Yes (if cacheable)	No
dcbtst	Yes (if cacheable)	No
dcbz	No	Yes
dccci	No	No
dcread	No	No

Architecturally, the **dcbt**, **dcbtst**, **dcbf,** and **dcbst** instructions are “loads.” These instructions do not change data. Flushing or storing a cache line from the cache is not architecturally a “store” because a store had already updated the cache; the **dcbf** or **dcbst** instruction only updates the copy in main memory.

The **dcbt** and **dcbtst** instructions can cause DAC-Read debug events independent of cacheability.

If data translation is enabled and the effective address (EA) of either **dcbt** or **dcbtst** encounters a data storage exception due to a TLB miss or a zone fault, the instruction becomes a no-op. Under this condition, these instructions still can cause DAC-Read debug events.

Although **dcbf** and **dcbst** are architecturally “loads,” these instructions can create DAC-Write (but not DAC-Read) debug events. In a debug environment, the fact that external memory is being written is the event of interest.

Although **dcread** and **dccci** are not address-specific (they affect a congruence class regardless of the instruction operand address), and are considered “loads,” on the IOP 480 CPU they do not cause DAC debug events.

All ICU operations (**icbi**, **icbt**, **iccci**, and **icread**) are architecturally treated as "loads." **icbi** and **icbt** cause DAC debug events. **iccci** and **icread** do not cause DAC debug events on the IOP 480 CPU.

26.6.3.2 DAC Applied to String Instructions

An **stswx** instruction with a string length of 0 is a no-op. The **lswx** instruction with the string length equal to 0 does not alter the RT contents with undefined data, as allowed by the PowerPC Architecture. Neither **stswx** nor **lswx** with zero length causes a DAC debug event because storage is not accessed by these instructions.

26.6.4 Instruction Address Compare Register (IAC1)

The IOP 480 CPU can take a debug event upon an attempt to execute an instruction from an address. The address, which must be Lword-aligned, is defined in the IAC register. The DBCR[IA1] field of the DBCR controls the Instruction Address Compare (IAC) debug event.

Register 26-4 illustrates the IAC1 bits.

26.7 DEBUG INTERFACE

The IOP 480 CPU processor provides a JTAG interface to support hardware and software test and debug. Typically, the interface connects to a debug port external to the IOP 480 CPU; the debug port is typically connected to a JTAG connector on a processor board.

26.7.1 IEEE 1149.1 Test Access Port (JTAG Debug Port)

The IEEE 1149.1 Test Access Port (TAP), commonly called the JTAG (Joint Test Action Group) debug port, is an architectural standard described in IEEE Standard 1149.1–1990, *IEEE Standard Test Access Port and Boundary Scan Architecture*. The standard describes a method for accessing internal chip facilities using a four- or five-signal interface.

The JTAG debug port, originally designed to support scan-based board testing, is enhanced to support the attachment of debug tools. The enhancements, which comply with the IEEE 1149.1 specifications for vendor-specific extensions, are compatible with standard JTAG hardware for boundary-scan system testing.

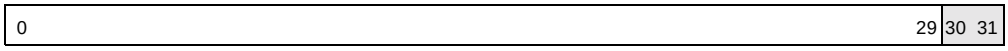
JTAG Signals	JTAG debug port implements the four required JTAG signals, TCLK, TMS, TDI, and TDO, and the optional $\overline{\text{TRST}}$ signal.
JTAG Clock Requirements	The frequency of the TCLK signal can range from DC to one-half of the internal chip clock frequency.
JTAG Reset Requirements	JTAG debug port logic is reset at the same time as a system reset. Upon receiving $\overline{\text{TRST}}$, the JTAG TAP controller returns to the Test-Logic Reset state.

26.7.1.1 JTAG Connector

A 16-pin male 2x8 header connector is suggested as the JTAG debug port connector for chips that incorporate the IOP 480 CPU. This connector definition matches the requirements of the RISCWatch debugger from IBM. The connector is shown in Figure 26-1 on page 26-9 and the signals are shown in Table 26-3 on page 26-9. The connector should be placed as close as possible to the chip containing the IOP 480 CPU to ensure signal integrity.

Position 14 does not contain a pin.

Register 26-4. Instruction Address Compare Register (IAC1)



0:29		Instruction Address Compare Lword address	Omit two low-order bits of complete address.
30:31		***Reserved***	

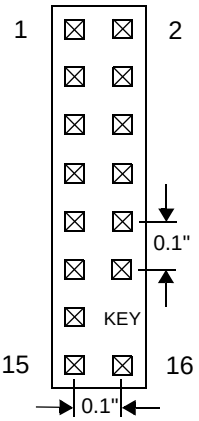


Figure 26-1. JTAG Connector Physical Layout (Top View)

Table 26-3. JTAG Connector Signals

Pin	I/O	Signal	Description
1	O	TDO	JTAG Test Data Out
2		No connect (NC)	***Reserved***
3	I	TDI[1]	JTAG Test Data In
4		$\overline{\text{TRST}}$	JTAG Reset
5		NC	***Reserved***
6		+POWER[2]	Processor Power OK
7	I	TCK[3]	JTAG Test Clock
8		NC	***Reserved***
9	I	TMS[1]	JTAG Test Mode Select
10		NC	***Reserved***
11	I	HALT[3]	Processor Halt
12		NC	***Reserved***
13		NC	***Reserved***
14		Key	The pin at this position should be removed
15		NC	***Reserved***
16		GND	Ground

1. A 10K ohm pull-up resistor can be connected to this signal to reduce chip power consumption. The pull-up resistor is not required.

2. The +POWER signal, sourced from the target development board, indicates whether the processor is operating. This signal does not supply power to the RISCWatch hardware or to the processor. The active level on this signal can be +5V or +3.3V (the IOP 480 CPU may have +5V or +3.3V I/O, but the processor must be powered by +3.3 V). A series resistor (1K ohm or less) should be used to provide short circuit current-limiting protection.

3. A 10K ohm pull-up resistor must be connected to these signals to ensure proper chip operation when these inputs are not used.

26.7.1.2 JTAG Instructions

The JTAG debug port provides the standard **extest, sample/preload,** and **bypass** instructions. Invalid instructions behave as the **bypass** instruction. There are three private instructions.

Table 26-4. JTAG Instructions

Instruction	Code	Comments
Extest	0000	IEEE 1149.1 standard
Sample/Preload	0001	IEEE 1149.1 standard
JTAG 5	0101	Private
JTAG 7	0111	Private
JTAG B	1011	Private
Bypass	1111	IEEE 1149.1 standard

26.7.1.3 JTAG Boundary Scan

Boundary Scan Description Language (BSDL), IEEE 1149.1b-1994, is a supplement to IEEE 1149.1-1990 and IEEE 1149.1a-1993 *Standard Test Access Port and Boundary-Scan Architecture*. BSDL, a subset of the IEEE 1076-1993 Standard VHSIC Hardware Description Language (VHDL), allows a rigorous description of testability features in components which comply with the standard. It is used by automated test pattern generation tools for package interconnect tests and by electronic design automation (EDA) tools for synthesized test logic and verification. BSDL supports robust extensions that can be used for internal test generation and to write software for hardware debug and diagnostics.

The primary components of BSDL include the logical port description, the physical pin map, the instruction set, and the boundary register description.

The logical port description assigns symbolic names to the pins of a chip. Each pin has a logical type of in, out, inout, buffer, or linkage that defines the logical direction of signal flow.

The physical pin map correlates the logical ports of the chip to the physical pins of a specific package. A BSDL description can have several physical pin maps; each map is given a unique name.

Instruction set statements describe the bit patterns that must be shifted into the Instruction Register to place the chip in the various test modes defined by the standard. Instruction set statements also support descriptions of instructions that are unique to the chip.

The boundary register description lists each cell or shift stage of the Boundary Register. Each cell has a unique number; the cell numbered 0 is the closest to the Test Data Out (TDO) pin and the cell with the highest number is closest to the Test Data In (TDI) pin. Each cell contains additional information, including: cell type, logical port associated with the cell, logical function of the cell, safe value, control cell number, disable value, and result value.

27 IOP 480 CPU MEMORY MANAGEMENT

The IOP 480 CPU has a 4 GB address space, which is presented as a flat address space.

The IOP 480 CPU memory management unit (MMU) performs address translation and protection functions. With appropriate system software, the MMU supports:

- Translation of the 4-GB logical address space into physical addresses
- Independent enabling of instruction and data translation and protection
- Page-level access control using the translation mechanism
- Software control of page replacement strategy
- Additional virtual-mode control of protection using zones
- Real-mode write protection

27.1 OVERVIEW

The instruction and integer units generate effective addresses (EAs) for instruction fetches and data accesses, respectively. (An EA is a 32-bit address formed by adding an index or displacement to a base address.) Instruction EAs are for sequential instruction fetches, and for fetches causing changes in program flow (branches and interrupts). Data EAs are for load/store and cache control instructions. The MMU translates EAs into real addresses; the instruction cache unit (ICU) and data cache unit (DCU) use real addresses to access memory.

The IOP 480 CPU MMU supports demand-paged virtual memory and other management schemes that depend on precise control of logical to physical address mapping and flexible memory protection. Translation misses and protection faults cause precise exceptions. Sufficient information is available to correct the fault and restart the faulting instruction.

The MMU divides logical storage into pages. The page represents the granularity of logical address translation and protection control. Eight page sizes (1 KB, 4 KB, 16 KB, 64 KB, 256 KB, 1 MB, 4 MB, and 16 MB) are simultaneously supported. For logical-to-physical address translation to be performed, a valid entry for the page containing the logical address must be in the translation lookaside buffer (TLB). Addresses for which no TLB entry exists cause TLB-miss exceptions.

27.2 ADDRESS TRANSLATION

Bits in the Machine State Register (MSR) control translation. The MSR[IR] (instruction relocate) bit controls translation for instruction accesses. The MSR[DR] (data relocate) bit controls the translation mechanism for data accesses. These bits, specified independently, can be changed at any time by a program in supervisor state. Note that all exceptions clear MSR[IR, DR] and place the processor in the supervisor state. Subsequent discussion about translation and protection assumes that MSR[IR, DR] are set appropriately.

The processor references memory when it fetches an instruction, and when it executes load/store, branch, and cache control instructions. An EA references a memory location. When translation is enabled, the EA is translated into a real address, as illustrated in Figure 27-1. The ICU or DCU uses the real address for the access. (When translation is not enabled, the EA is already a real address.)

In address translation, the EA is combined with an 8-bit process ID (PID) to create a 40-bit virtual address. The virtual address is compared to all of the TLB entries. A matching entry, if found in the TLB, supplies the real address for the storage reference. Figure 27-1 illustrates the process.

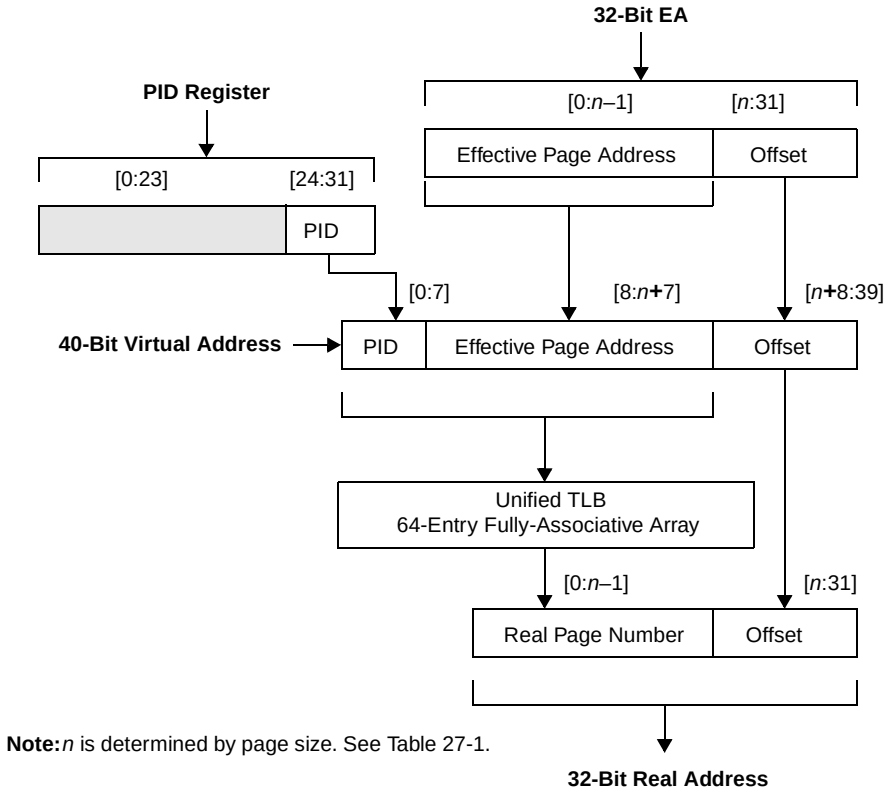


Figure 27-1. Effective to Real Address Translation Flow

27.3 TRANSLATION LOOKASIDE BUFFER (TLB)

The TLB is hardware that controls translation, protection, and storage attributes. The instruction and data units share a unified 64-entry, fully associative TLB (a TLB in which any page entry (TLB entry) can be placed anywhere in the TLB). TLB entries are maintained under software control. System software determines the TLB entry replacement strategy and the format and use of page state information. A TLB entry contains the information required to identify the page, to specify translation and protection controls, and to specify the storage attributes.

27.3.1 Unified TLB

The unified TLB (UTLB) contains 64 entries; each has a TLBHI (tag) portion and a TLBLO (data) portion. TLBHI contains 36 bits; TLBLO contains 32 bits. When translation is enabled, the UTLB tag portion compares some or all of $EA_{0:21}$ with some or all of the effective page number $EPN_{0:21}$, based on the size bits $SIZE_{0:2}$. The 64 entries are simultaneously checked for a match. If an entry matches, the corresponding data portion of the UTLB provides the real page number (RPN), access control bits (ZSEL, EX, WR), and storage attributes (W, I, M, G, E, K). A programming error occurs if multiple entries match during a TLB look-up. The results of such a look-up are undefined. Figure 27-2 on page 27-3 illustrates the UTLB.

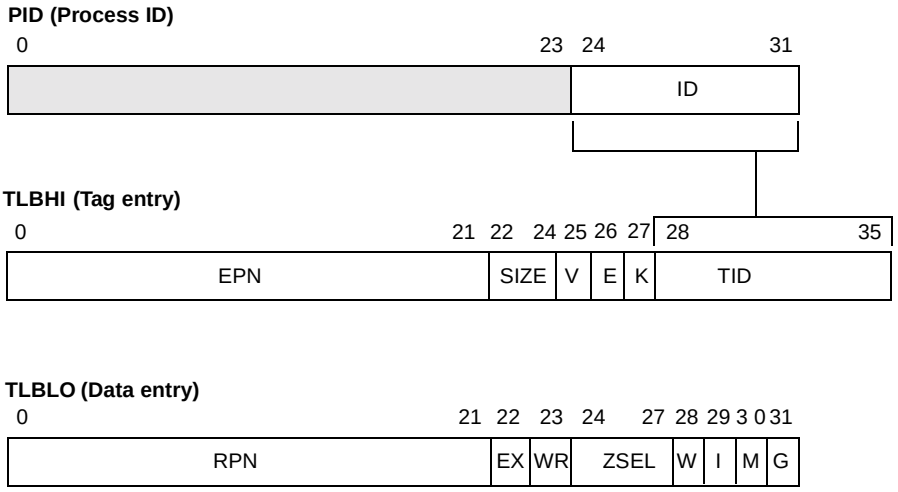


Figure 27-2. TLB Entries

The virtual address space is extended by adding an 8-bit translation ID (TID) loaded from the Process ID (PID) register during a TLB access. The PID identifies one of 255 unique software entities (usually a process or thread). TBL_entry[TID] is compared to the PID during TLB look-up.

Tag and data entries are written by copying data from GPRs and the PID, using the **tlbwe** instruction. Tag and data entries are read by copying data to GPRs and the PID, using the **tlbre** instruction. Software can search for specific entries using the **tlbsx** instruction.

27.3.2 Unified TLB Fields

Each TLB entry describes a page that is eligible for translation and access controls. Fields in the TLB entry fall into four categories:

- Information required to identify the page to the hardware translation mechanism
- Control information specifying the translation
- Access control information
- Storage attribute control information

27.3.3 Page Identification Fields

When an EA is presented to the MMU for processing, the MMU applies several selection criteria to each TLB entry to select the appropriate entry. Although it is possible to place multiple entries into the TLB to match a specific EA and PID, this is considered a programming error and the results are undefined. The following fields in the TLB entry identify the page. Except as noted, all comparisons must succeed to validate an entry for subsequent processing.

EPN (effective page number, 22 bits)

Compared to some number of the $EA_{0:21}$ bits presented to the MMU.

The exact comparison depends on the page size, as specified in Table 27-1.

Table 27-1. TLB Fields Related to Page Size

Page Size	Size Field	*n* Bits Compared	EPN to EA Comparison	RPN Bits Set to 0
1 KB	000	22	$EPN_{0:21} \leftrightarrow EA_{0:21}$	—
4 KB	001	20	$EPN_{0:19} \leftrightarrow EA_{0:19}$	$RPN_{20:21}$
16 KB	010	18	$EPN_{0:17} \leftrightarrow EA_{0:17}$	$RPN_{18:21}$
64 KB	011	16	$EPN_{0:15} \leftrightarrow EA_{0:15}$	$RPN_{16:21}$
256 KB	100	14	$EPN_{0:13} \leftrightarrow EA_{0:13}$	$RPN_{14:21}$
1 MB	101	12	$EPN_{0:11} \leftrightarrow EA_{0:11}$	$RPN_{12:21}$
4 MB	110	10	$EPN_{0:9} \leftrightarrow EA_{0:9}$	$RPN_{10:21}$
16 MB	111	8	$EPN_{0:7} \leftrightarrow EA_{0:7}$	$RPN_{8:21}$

SIZE (page size, 3 bits)

Selects one of the eight page sizes, 1 KB–16 MB, listed in Table 27-1 on page 27-3.

V (valid, 1 bit)

Indicates whether a TLB entry is valid and can be used for translation.

A valid TLB entry implies read access, unless overridden by zone protection. TLB_entry[V] can be written using a **tlbwe** instruction. The **tlbia** instruction invalidates all TLB entries.

TID (translation ID, 8 bits)

Loaded from the PID register during a **tlbwe** operation. The TID value, which extends the EA, is compared with the PID value during a TLB access (see Table 27-2 on page 27-9). The TID provides a convenient way to associate a translation with one of 255 unique software entities, typically a process or thread maintained by system software. Setting TLB_entry[TID] = 0000 disables TID-PID comparison and identifies a TLB entry as valid for all processes; the value of the PID register is irrelevant.

27.3.3.1 Translation Field

When a TLB entry is identified as matching an EA (and possibly the PID), TLB_entry[RPN] defines how the EA is translated.

RPN (real page number, 22 bits)

Replaces some, or all, of $EA_{0:21}$, depending on page size. For example, a 16 KB page uses $EA_{0:17}$ for comparison. The translation mechanism replaces $EA_{0:17}$ with $TLB\_entry[RPN]_{0:17}$ to form the physical address, and $EA_{18:31}$ becomes the real page offset, as illustrated in Figure 27-1 on page 27-2.

Note: *Software must set all unused bits of RPN (as determined by page size) to 0. See Table 27-1 on page 27-3.*

27.3.3.2 Access Control Fields

Several access controls are available in the UTLB entries.

ZSEL (zone select, 4 bits)

Selects one of 16 zone fields (Z0—Z15) from the Zone Protection Register (ZPR). The ZPR field bits can modify the access protection specified by the TLB_entry[V, EX, WR] bits of a TLB entry. Zone protection is described in detail in Section 27.7.1.4, “Zone Protection,” on page 27-10.

EX (execute enable, 1 bit)

When set (TLB_entry[EX] = 1), allows instruction execution at addresses within a page. ZPR settings can override TLB_entry[EX]; see Section 27.7.1.4, “Zone Protection,” on page 27-10, for more information.

WR (write-enable 1 bit)

When set (TLB_entry[WR] = 1), permits store operations to addresses in a page. ZPR settings can override TLB_entry[WR]; see Section 27.7.1.4, “Zone Protection,” on page 27-10.

27.3.3.3 Storage Attribute Fields

TLB entries contain bits that set and provide information about the storage control attributes. Four of the attributes (W, I, M, and G) are defined in the PowerPC Architecture. The E storage attributes are defined in the IBM PowerPC Embedded Environment. The K attribute is implementation-specific.

W (write-through, 1 bit)

When set (TLB_entry[W] = 1), stores are specified as write-through. If data in the referenced page is in the data cache, a store updates the cached copy of the data and the external memory location. Contrast this with a write-back strategy, which updates memory only when a cache line is flushed.

In real mode, the Data Cache Write-thru Register (DCWR) controls the setting of the W storage attribute.

Note that the PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited. It is considered a programming error to use these memory models; the results are boundedly undefined.

I (caching inhibited, 1 bit)

When set (TLB_entry[I] = 1), a memory access is completed by referencing the location in main memory, bypassing the cache arrays. During the access, the accessed location is not put into the cache arrays.

In real mode, the Instruction Cache Cacheability Register (ICCR) and Data Cache Cacheability Register (DCCR) control the setting of the I storage registers. In these registers, the polarity of the bit is

reversed; 1 indicates that a storage control region is cacheable, rather than caching inhibited).

The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited. It is considered a programming error to use these memory models; the results are boundedly undefined.

It is considered a programming error if the target location of a load/store, **dcbz**, or fetch access to caching inhibited storage is in the cache; the results are boundedly undefined. It is *not* considered a programming error for the target locations of other cache control instructions to be in the cache when caching is inhibited.

M (memory coherent, 1 bit)

For some implementations that support multiprocessing, the M storage attribute improves the performance of memory coherency management. The IOP 480 CPU does not provide multi-processor support or hardware support for data coherency; the M bit is implemented, but has no effect.

G (guarded, 1 bit)

When set (TLB_entry[G] = 1), indicates that the hardware cannot speculatively access the location for pre-fetching or out-of-order load access. The G storage attribute is typically used to protect memory-mapped I/O from inadvertent access. Attempted execution of an instruction from a guarded address, while instruction translation is enabled, results in an instruction storage exception.

An instruction fetch to a guarded region does not occur until the execution pipeline is empty, guaranteeing that the access is necessary and therefore not speculative. For this reason, performance is degraded when executing out of guarded regions, and software should avoid unnecessarily marking regions of instruction storage as guarded.

In real mode, the Storage Guarded Register (SGR) controls the setting of the G storage attribute.

K (compression, 1 bit)

When set (TLB_entry[K] = 1), indicates that data in the associated page is stored in compressed mode.

In real mode, the Storage Compression Register (SKR) controls the setting of the K storage attribute.

E (endian, 1 bit)

When set (TLB_entry[E] = 1), indicates that data in the associated page is stored in true Little Endian format.

In real mode, the Storage Little-Endian Register (SLER) controls the setting of the E storage attribute.

27.3.4 Shadow Instruction TLB

To enhance performance, four instruction-side TLB entries are kept in a four-entry fully-associative shadow array. This array, called the instruction TLB (ITLB), helps to avoid TLB contention between instruction accesses to the TLB and load/store operations. Replacement and invalidation of the ITLB entries is managed by hardware during routine execution (see Section 27.3.4.2, “ITLB Consistency,” on page 27-6 for details).

The ITLB can be considered a level-1 instruction-side TLB; the UTLB serves as the level-2 instruction-side TLB and the level-1 data-side TLB. The ITLB is used only by instruction fetches for storing instruction address translations. Each ITLB entry contains the translation information for a page. The processor uses the ITLB for address translation of instruction accesses when MSR[IR] = 1.

27.3.4.1 ITLB Accesses

The instruction unit accesses the ITLB independently of the rest of the MMU. ITLB accesses are transparent to the executing program, except that ITLB hits contribute to higher overall instruction throughput by allowing data translations to occur in parallel. Therefore, when instruction accesses hit in the ITLB, the address translation mechanisms in the UTLB are available for use by data accesses simultaneously.

The ITLB requests a new entry from the UTLB when an ITLB miss occurs. A two-cycle penalty occurs at each ITLB miss that is also a UTLB hit; the penalty is larger if it is also a UTLB miss, or if there is contention for the UTLB from the data side. A round-robin replacement algorithm replaces existing entries with new entries.

Figure 27-3 illustrates the relationship of the ITLB and UTLB in address translation.

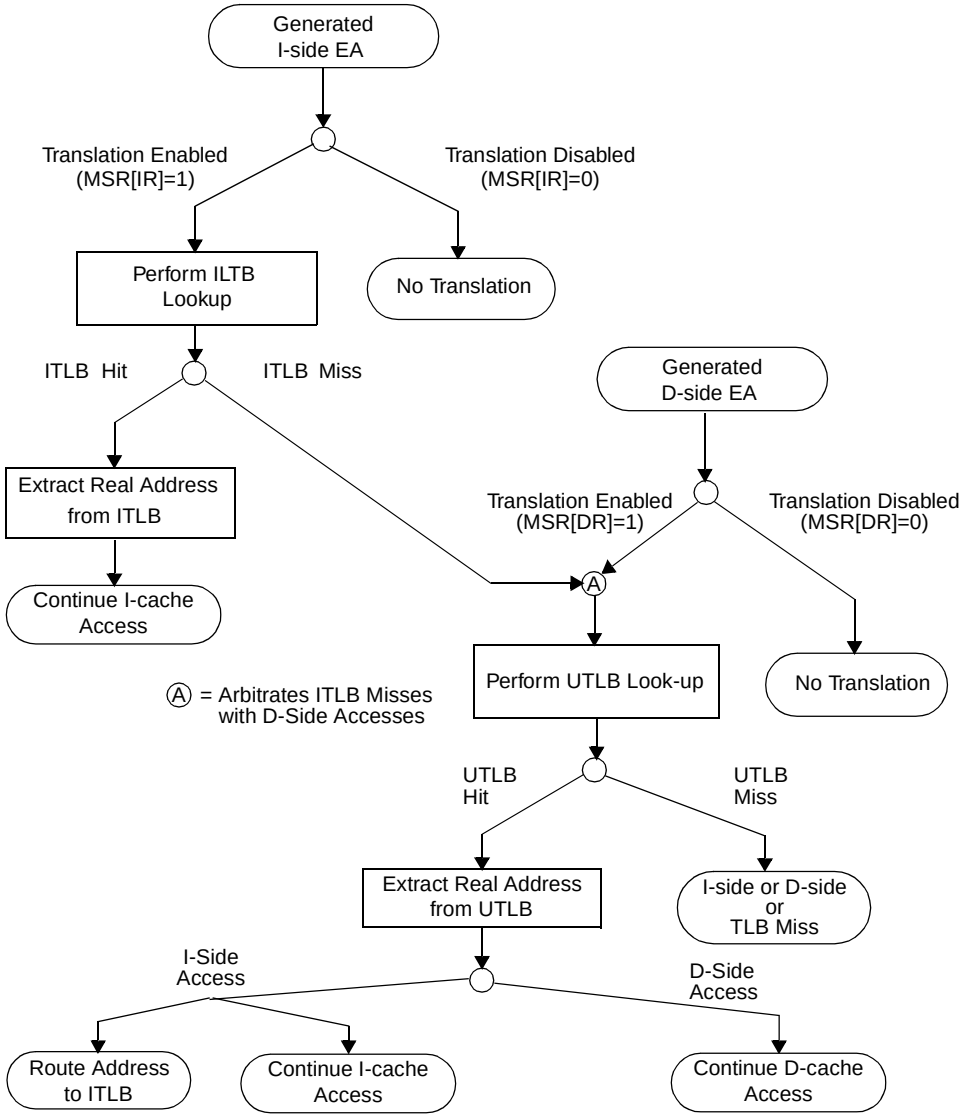


Figure 27-3. ITLB/UTLB Address Resolution

27.3.4.2 ITLB Consistency

The processor invalidates the entire ITLB contents when the following context-synchronizing events occur, to help to maintain ITLB integrity.

- **isync** instruction
- Processor context switch (all interrupts, **rfi**, **rfci**)

If software updates a translation/protection mechanism (UTLB, PID, ZPR, or MSR) and must synchronize these updates with the ITLB, the *software* must perform the necessary context synchronization.

A typical example is the manipulation of the TLB by an operating system within an interrupt handler for a TLB miss. Upon entry to the interrupt handler, the ITLB is invalidated and translation is disabled. If the operating system simply made the TLB updates and returned from the handler (using **rfi**), no additional explicit software action would be required to synchronize the ITLB.

If, instead, the operating system re-enables translation within the handler, and then performs TLB updates within the handler, those updates would not be effective in the ITLB until **rfi** is executed to return from the handler. For those TLB updates to be reflected in

the ITLB *within* the handler, an **isync** must be issued after TLB updates finish. Failure to properly synchronize the ITLB can cause unexpected behavior.

Note: *As a rule, follow software manipulation of translation mechanism (if performed while translation is active) with a context-synchronizing operation (usually **isync**).*

27.4 TLB-RELATED EXCEPTIONS

The processor relies on exception processing to implement paged virtual memory, and to enforce protection of specified memory pages.

When an enabled exception (an interrupt) occurs, the processor clears MSR[IR, DR]. Therefore, at least at the beginning of all exception handlers, the processor operates in real mode for instruction accesses and data accesses. Note that when address translation is disabled (MSR[IR] = 0 and MSR[DR] = 0) for an instruction fetch or load/store, the EA is treated as the real address and is passed directly to the memory subsystem (including cache units) as a direct address. Such direct addresses bypass all memory protection checks performed by the MMU.

MMU accesses can result in the following exceptions:

- Data storage exception
- Instruction storage exception
- Data TLB miss exception
- Instruction TLB miss exception
- Program exception

27.4.1 Data Storage Exception

A data storage exception occurs when data translation is active, and the desired access to the EA is not permitted for one of the following reasons:

1. In the problem state

- **icbi**, load/store, **dcbz**, or **dcbf** having an EA having ZPR[Zn] = 00. (In this case, **dcbt** and **dcbtst** no-op, rather than cause an exception. Privileged instructions cannot cause data storage exceptions.)
- Stores, or **dcbz**, to an EA having TLB_entry[WR] = 0 and ZPR[Zn] ≠ 11. (The privileged instructions **dcbi** and **dccci** are treated as “stores”, but cause program exceptions, rather than data storage exceptions.)

2. In the supervisor state

- Data store, **dcbi**, **dcbz**, or **dccci** to an EA having TLB_entry[WR] = 0 and ZPR[Zn] other than 11 or 10.

Section 27.7.1.4, “Zone Protection,” on page 27-10, describes zone protection in detail. See Section 11.7.7, “Instruction Storage Exception,” on page 11-22, for a detailed discussion of the data storage exception.

27.4.2 Instruction Storage Exception

An instruction access exception occurs when instruction translation is active, and the processor attempts to execute an instruction at an EA for which fetch access is not permitted, for any of the following reasons:

1. In the problem state

- Instruction fetch from an EA with ZPR[Zn] = 00.
- Instruction fetch from an EA having TLB_entry[EX] = 0 and ZPR[Zn] ≠ 11.
- Instruction fetch from an EA having TLB_entry[G] = 1.

2. In the supervisor state

- Instruction fetch from an EA having TLB_entry[EX] = 0 and ZPR[Zn] other than 11 or 10.
- Instruction fetch from an EA having TLB_entry[G] = 1.

See Section 27.7.1.4, “Zone Protection,” on page 27-10, for a detailed discussion of zone protection. See Section 11.7.7, “Instruction Storage Exception,” on page 11-22, for a detailed discussion of the instruction storage exception.

27.4.3 Data TLB Miss Exception

A data TLB miss exception is generated if data translation is enabled and a valid TLB entry matching the EA and PID is not present. The exception applies to data access instructions and cache operations (excluding cache touch instructions).

See Section 11.7.15, “Data TLB Miss Exception,” on page 11-27, for a detailed discussion.

27.4.4 Instruction TLB Miss Exception

The instruction TLB miss exception is generated if instruction translation is enabled and execution is attempted for an instruction for which a valid TLB entry matching the EA and PID for the instruction fetch is not present.

See Section 11.7.16, “Instruction TLB Miss Exception,” on page 11-27, for a detailed discussion.

27.4.5 Program Exception

When the TIE_cpuMmuEn signal is tied to 0, the TLB instructions (**tlbia**, **tlbre**, **tlbsx**, **tlbsync**, and **tlbwe**) are treated as illegal instructions. When execution of any of these instructions occurs under this circumstance, a program exception results.

When TIE_cpuMmuEn is tied to 0, MSR[IR, DR] = 0.

Note: *When TIE_cpuMmuEn is tied to 0, MSR[IR, DR] = 0 upon execution of an* **rfi** *or* **rfci** *instruction, even if an exception handler sets MSR[IR] = 1 or MSR[DR] = 1 in Save/Restore Register 0 (SRR0) or SRR3.*

See Section 11.7.16, “Instruction TLB Miss Exception,” on page 11-27, for a detailed discussion.

27.5 TLB MANAGEMENT

The processor does not imply any format for the page tables or the page table entries. Software has significant flexibility in implementing a custom replacement strategy. For example, software can “lock” TLB entries that correspond to frequently used storage, so that those entries are never cast out of the TLB, and TLB miss exceptions to those pages never occur.

TLB management is performed in software with some hardware assist, consisting of:

- Storage of the missed EA in the Save/Restore Register 0 (SRR0) register for an instruction-side miss, or in the Data Exception Address Register (DEAR) for a data-side miss.
- Instructions for reading, writing, searching, and invalidating the TLB, as described briefly in the following subsections. See Section 28, “IOP 480 CPU Instruction Set,” for detailed instruction descriptions.

27.5.1 TLB Search Instructions (tlbsx/tlbsx.)

tlbsx instruction locates entries in the TLB, to find the TLB entry associated with an exception, or to locate candidate entries to cast out. **tlbsx** searches the UTLB array for a matching entry. The EA is the value to be matched; EA = (RA|0)+(RB).

If the TLB entry is found, its index is placed in $RT_{26:31}$. RT can then serve as the source register for a **tlbre** or **tlbwe** instruction to read or write the entry, respectively. If no match is found, the contents of RT are undefined.

tlbsx. sets the Condition Register (CR) bit $CR0_{EQ}$. The value of $CR0_{EQ}$ depends on whether an entry is found: $CR0_{EQ}$ = 1 if an entry is found; $CR0_{EQ}$ = 0 if no entry is found.

27.5.2 TLB Read/Write Instructions (tlbre/tlbwe)

TLB entries can be accessed for reading and writing by **tlbre** and **tlbwe**, respectively. Separate extended mnemonics are available for the TLBHI (tag) and TLBLO (data) portions of a TLB entry.

27.5.3 TLB Invalidate Instruction (tlbia)

tlbia sets TLB_entry[V] = 0 to invalidate all TLB entries. All other TLB entry fields remain unchanged.

Using **tlbwe** to set TLB_entry[V] = 0 invalidates a specific TLB entry.

27.5.4 TLB Sync Instruction (tlbsync)

tlbsync guarantees that all TLB operations have completed for all processors in a multi-processor system. The IOP 480 CPU provides no multiprocessor support, so this instruction performs no function. The instruction is included to facilitate code portability.

27.6 RECORDING PAGE REFERENCES AND CHANGES

When system software manages virtual memory, the software views physical memory as a collection of pages. Each page is associated with at least one TLB entry. To manage memory effectively, system software often must know whether a particular page has been referenced or modified. Note that this involves more than knowing whether a particular

TLB entry was used to reference or alter memory, because multiple TLB entries can translate to the same page.

When system software manages a demand-paged environment, and the software needs to replace the contents of a page with other data, previously referenced pages (accessed for any purpose) are more likely to be maintained than pages that were never referenced. If the contents of a page must be replaced, and data contained in that page was modified, system software generally must write the contents of the modified page to the backing store before replacing its contents. System software must maintain records to control the environment.

Similarly, when system software manages TLB entries, the software often must know whether a particular TLB entry was referenced. When the system software must select a TLB entry to cast, previously referenced entries are more likely to be maintained than entries which were never referenced. System software must also maintain records for this purpose.

The IOP 480 CPU does not provide hardware reference or change bits, but TLB miss exceptions and data storage exceptions enable system software to maintain reference information for TLB entries and their associated pages, respectively.

First, the TLB entries are built, with each TLB_entry[V, WR] = 0. System software retains the index and EPN of each entry.

The first attempt by application code to access a page causes a TLB miss exception, because its TLB entry is marked invalid. The TLB miss handler records the reference to the TLB entry (and to the associated page) in a table, then sets TLB_entry[V] = 1. (Note that TLB_entry[V] can be considered a reference bit for the TLB entry.) Subsequent read accesses to the page associated with the TLB entry proceed normally.

In the example just given for recording TLB entry references, the first write access to the page using the TLB entry, after the entry is made valid, causes a data storage exception (write access was turned off.). The TLB miss handler records the write to the page in a table, then sets TLB_entry[WR] = 1. (Note that TLB_entry[WR] can be considered a change bit for the page.) Subsequent write accesses to the page proceed normally.

27.7 ACCESS PROTECTION

The IOP 480 CPU provides virtual-mode access protection. The TLB entry enables system software to control general access for programs in the problem state, and control write and execute permissions for all pages. The TLB entry can specify zone protection that can override the other access control mechanisms supported in the TLB entries.

TLB entry and zone protection methods also support access controls for cache operation and string loads/stores.

27.7.1 Access Protection Mechanisms in the TLB

For MMU access protection to be in effect, one or both of MSR[IR] or MSR[DR] must be active. MSR[IR] enables protection on instruction fetches, which are inherently read-only. MSR[DR] enables protection on data accesses (loads/stores).

27.7.1.1 General Access Protection

The translation ID (TLB_entry[TID]) provides the first level of MMU access protection. This 8-bit field, if non-zero, is compared to the contents of TLB_entry[PID] (see Table 27-2 on page 27-9). These fields must match (after successful comparison of EA) in a valid TLB entry if any access is to be allowed. In typical use, it is assumed that a program in the supervisor state (such as a real-time operating system) sets the PID before enabling a program in the problem state program that is subject to access control.

If TLB_entry[TID] = 0000, the associated memory page is accessible to all programs, regardless of their PID. This enables multiple processes to share common code and data. The common area is still subject to all other access protection mechanisms.

Table 27-2. Process ID (PID)

0:23		***Reserved***
24:31		Process ID

27.7.1.2 Execute Permissions

If MSR[IR] = 1, instruction fetches are subject to MMU translation and are afforded MMU access protection. Fetches are inherently read-only, so write protection is not needed. Instead, using TLB_entry[EX], a memory page is marked as executable (contains instructions) or not executable (contains data or memory-mapped control hardware).

If an instruction is pre-fetched from a memory page for which TLB_entry[EX] = 0, the instruction is tagged as an error. If the processor subsequently attempts to execute this instruction, an instruction storage exception results. This exception is precise with respect to the attempted execution. If the fetcher discards the instruction without attempting to execute it, no exception results.

Zone protection can alter execution protection.

27.7.1.3 Write Permissions

If MSR[DR] = 1, data loads and stores are subject to MMU translation and are afforded MMU access protection. The existence of a TLB entry describing a memory page implies read access; write access is controlled by TLB_entry[WR].

If a store (including **dcbz**, **dcbi**, or **dccci**) is made to an EA having TLB_entry[WR] = 0, a data storage exception results. This exception is precise.

Zone protection can alter write protection (see Section 27.7.1.4, “Zone Protection,” on page 27-10). In addition, only zone protection can prevent read access of a page defined by a TLB entry.

27.7.1.4 Zone Protection

Each TLB entry contains a 4-bit zone select (ZSEL) field. A zone is an arbitrary identifier for grouping TLB entries (memory pages) for purposes of protection. As many as16 different zones may be defined. Any zone can have any number of member pages.

Each zone is associated with a 2-bit field (Z0–Z15) in the ZPR. The values of the field define how protection is applied to all pages that are member of that zone. Changing the value of the ZPR field can alters the protection attributes of all pages in the zone. Without ZPR, the change would require finding, reading, altering, and rewriting the TLB entry for each page in a zone, individually. The ZPR provides a much faster means of altering the protection for groups of memory pages.

The ZSEL values 0–15 select ZPR fields Z0–Z15, respectively.

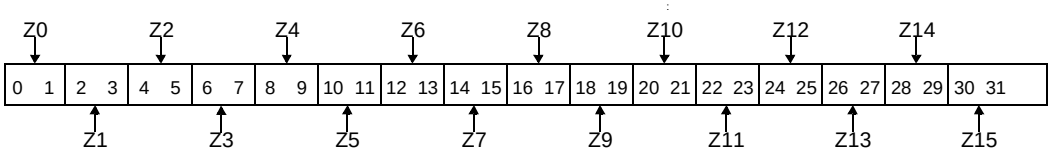
While it is common for TLB_entry[EX, WR] to be identical for all member pages in a group, this is not required. The ZPR field alters the protection defined by TLB_entry[EX] and TLB_entry[WR], on a page-by-page basis, as shown in Table 27-1 on page 27-11. An application program (presumed to be running in the problem state) can have execute and write permissions as defined by TLB_entry[EX] and TLB_entry[WR] for the individual pages, or no access (denies loads, as well as stores and execution), or complete access.

Setting ZPR[Zn] = 00 for a ZPR field is the only way to deny read access to a page defined by an otherwise valid TLB entry. TLB_entry[EX] and TLB_entry[WR] do not support read protection. Note that the **icbi** instruction is considered a load with respect to access protection; executed in user-mode, it causes a data storage exception if MSR[DR] = 1 and ZPR[Zn] = 00 is associated with the EA.

For a given ZPR field value, a program in supervisor state always has equal or greater access than a program in the problem state. System software can never be denied read (load) access for a valid TLB entry.

Register 27-1 illustrates the ZPR bits.

Register 27-1. Zone Protection Register (ZPR)



0:1	Z0	TLB page access control for all pages in this zone; the TLB bits EX and WR are bits that translate an effective address and PID.	
		In Problem State (MSR[PR] = 1): 00 No access 01 Access controlled by EX and WR 10 Access controlled by EX and WR 11 Accessed as if EX and WR are set	In Supervisor State (MSR[PR] = 0): 00 Access controlled by EX and WR 01 Access controlled by EX and WR 10 Access as if EX and WR are set 11 Accessed as if EX and WR are set
2:3	Z1	See the description of Z0.	
4:5	Z2	See the description of Z0.	
6:7	Z3	See the description of Z0.	
8:9	Z4	See the description of Z0.	
10:11	Z5	See the description of Z0.	
12:13	Z6	See the description of Z0.	
14:15	Z7	See the description of Z0.	
16:17	Z8	See the description of Z0.	
18:19	Z9	See the description of Z0.	
20:21	Z10	See the description of Z0.	
22:23	Z11	See the description of Z0.	
24:25	Z12	See the description of Z0.	
26:27	Z13	See the description of Z0.	
28:29	Z14	See the description of Z0.	
30:31	Z15	See the description of Z0.	

27.7.2 Access Protection for Cache Instructions

Architecturally the instructions **dcba**, **dcbi**, and **dcbz** are treated as “stores” since they can change data (or cause loss of data by invalidating a dirty line).

If data translation is enabled, and TLB_entry[WR] = 0, **dcbi** and **dcbz** can cause data storage exceptions. **dcbz** can cause a data storage exception when executed in user mode and ZPR[Zn] = 00; **dcbi** cannot, because it is a privileged instruction.

The **dcba** instruction enables “speculative” line establishment in the cache arrays; the established lines do not cause a line fill. Because the effects of **dcba** are speculative, exceptions that would otherwise result when ZPR[Zn] = 00 or TLB_entry[WR] = 0 do not occur. In such cases, **dcba** is treated as a no-op.

The **dccci** instruction can also be considered a “store” since it can change data by invalidating a dirty line; however, **dccci** is not address-specific (it affects an entire congruence class regardless of the operand address of the instruction). To restrict possible damage from an instruction which can change data and yet avoids the protection mechanism, the **dccci** instruction is privileged.

If data translation is enabled, **dccci** can cause data storage exceptions when TLB_entry[WR] = 0; the operand is treated as if it were address-specific. **dccci** cannot cause a data storage exception when ZPR[Zn] = 00, because it is a privileged instruction.

Because **dccci** can cause data storage and TLB-miss exceptions, use of **dccci** is not recommended when MSR[DR] = 1; if it is used, take care that the specific operand address does not cause an exception.

Architecturally, **dcbt** and **dcbtst** are treated as “loads” because they do not change data; they cannot cause data storage exceptions when TLB_entry[WR] = 0.

The cache block touch instructions are considered “speculative” loads; therefore, if a data storage exception would otherwise result from the execution of **dcbt** or **dcbtst** when ZPR[Zn] = 00, the instruction

is treated as a no-op and the exception does not occur. Similarly, TLB miss exceptions do not occur for these instructions.

Architecturally, **dcbf** and **dcbst** are treated as “loads”. Flushing or storing a line from the cache is not architecturally considered a “store” because a store was performed to update the cache, and **dcbf** or **dcbst** only update main memory. Therefore, neither **dcbf** nor **dcbst** can cause data storage exceptions when TLB_entry[WR] = 0. Because neither instruction is privileged, they can cause data storage exceptions when ZPR[Zn] = 00 and data translation is enabled.

dcread is a “load” from a non-specific address, and is privileged. Therefore, it cannot cause data storage exceptions when ZPR[Zn] = 00 or TLB_entry[WR] = 0.

icbi and **icbt** are considered “loads” and cannot cause data storage exceptions when TLB_entry[WR] = 0. **icbi** can cause data storage exceptions when ZPR[Zn] = 00. Because **icbt** is privileged, it cannot cause data storage exceptions when ZPR[Zn] = 00.

The **iccci** instruction cannot change data; an instruction cache line cannot be dirty. The **iccci** instruction is privileged and considered a load. It does not cause data storage exceptions when ZPR[Zn] = 00 or TLB_entry[WR] = 0.

Because **iccci** can cause a TLB miss exception, use of **iccci** is not recommended when MSR[DR] = 1; if it is used, take care that a specific operand address does not cause an exception.

icread is considered a “load” from a non-specific address, and is privileged. Therefore, it cannot cause data storage exceptions when ZPR[Zn] = 00 or TLB_entry[WR] = 0.

Table 27-3 summarizes the conditions under which the cache control instructions can cause data storage exceptions.

27.7.3 Access Protection for String Instructions

The **stswx** instruction with string length equal to zero (XER[TBC] = 0) is a no-op.

When data translation is enabled and XER[TBC] = 0, neither **lswx** nor **stswx** can cause TLB miss exceptions, or data storage exceptions when ZPR[Z*n*] = 0 or TLB_entry[WR] = 0.

Table 27-3. Protection Applied to Cache Control Instructions

Instruction	Possible Data Storage Exception	
	when ZPR[Zn] = 00	when TLB_entry[WR] = 0
dcba	No (instruction no-ops)	No (instruction no-ops)
dcbf	Yes	No
dcbi	No	Yes
dcbst	Yes	No
dcbt	No (instruction no-ops)	No
dcbtst	No (instruction no-ops)	No
dcbz	Yes	Yes
dccci	No	Yes
dcread	No	No
icbi	Yes	No
icbt	No	No
iccci	No	No
icread	No	No

27.8 REAL-MODE STORAGE ATTRIBUTE CONTROL

The PowerPC Architecture and the PowerPC Embedded Environment define several SPRs to control the following storage attributes in real mode: W, I, M, G, K, and E. Note that the K and E attributes are not defined in the PowerPC Architecture. The E attribute is defined in the IBM PowerPC Embedded Environment. The K attribute is implementation specific.

Six SPRs, called storage attribute control registers, control the various storage attributes when address translation is disabled. When address translation is enabled, these registers are ignored, and the storage attributes supplied by the TLB entry are used (see Section 27.3.2, “Unified TLB Fields,” on page 27-3).

These registers divide the 4 GB real address space into thirty-two 128 MB regions. In a storage attribute control register, bit 0 controls the lowest 128 MB region, bit 1 the next 128 MB region, and so on.

For detailed information on the function of the storage attributes, see Section 27.3.3.3, “Storage Attribute Fields,” on page 27-4.

The registers control storage attributes in real mode:

- The Data Cache Write-thru Register (DCWR) controls the W storage attribute.

  The DCWR controls write-through policy (the W storage attribute) in real mode (data translation disabled), for the data cache unit (DCU). Write-through is not applicable to the instruction cache unit (ICU).

  The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

- The Data Cache Cacheability Register (DCCR) controls the I storage attribute for data accesses and cache management instructions. Note that the polarity of the bits in this register is opposite to that of the I attribute (DCCR[Sn] = 1 enables caching, while TLB_entry[I] = 1 inhibits caching).

  Following any reset, all DCCR bits are set to 0. No memory regions are cacheable. Before memory regions can be designated as cacheable in the DCCR, it is necessary to execute the **dccci** instruction once for each congruence class in the DCU cache array. This procedure invalidates all congruence classes. The DCCR can then be reconfigured, and the DCU can begin normal operation.

  The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

- The Instruction Cache Cacheability Register (ICCR) controls the I storage attribute for instruction fetches. Note that the polarity of the bits in this register is opposite to that of the I attribute (ICCR[Sn] = 1 enables caching, while TLB_entry[I] 1 inhibits caching).

  Following any reset, all ICCR bits are set to 0. No memory regions are cacheable. Before memory regions can be designated as cacheable in the ICCR, it is necessary to execute the **dccci** instruction once for each congruence class in the DCU cache array. This procedure invalidates all congruence classes. The ICCR can then be reconfigured, and the ICU can begin normal operation.

  The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

- The Storage Guarded Register (SGR) controls the G storage attribute for instruction and data accesses.

  This attribute does not affect data accesses; the IOP 480 CPU does not perform speculative loads or stores.

  After any reset, the SGR is set to 0xFFFF FFFF, marking all of storage as guarded. For best performance, system software should clear the guarded attribute of appropriate regions as soon as possible. In instruction translate mode (MSR[IR] = 1), the G attribute comes from the TLB entry, and attempting to execute from a guarded region in translate mode causes an instruction storage exception. See Section 11.7.7, “Instruction Storage Exception,” on page 11-22, for more information.

- The Storage Compression Register (SKR) controls the K storage attribute for instruction and data accesses.

  This attribute determines whether storage is compressed.

  After any reset, all SKR bits are set to 0 (uncompressed).

- The Storage Little-Endian Register (SLER) controls the E storage attribute for instruction and data accesses.

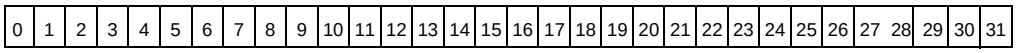
  This attribute determines the byte ordering of storage. Section 24.4, “Byte Ordering,” on page 24-13, provides a detailed description of byte ordering in the IBM PowerPC Embedded Environment.

  After any reset, all SLER bits are set to 0 (Big Endian).

Table 27-4 shows a generic storage attribute control register. The actual storage attribute control registers have the same bit numbers and address ranges.

$EA_{0:4}$ specify a storage control region. a bit from each storage attribute control register. For the DCWR, SGR, SKR, and SLER registers, the selected bits are directly used as the W, G, K, and E storage attributes, respectively.

Table 27-4. Storage Attribute Control Registers



Bit	Address Range	Bit	Address Range
0	0x0000 0000 – 0x07FF FFFF	16	0x8000 0000 – 0x87FF FFFF
1	0x0800 0000 – 0x0FFF FFFF	17	0x8800 0000 – 0x8FFF FFFF
2	0x1000 0000 – 0x17FF FFFF	18	0x9000 0000 – 0x97FF FFFF
3	0x1800 0000 – 0x1FFF FFFF	19	0x9800 0000 – 0x9FFF FFFF
4	0x2000 0000 – 0x27FF FFFF	20	0xA000 0000 – 0xA7FF FFFF
5	0x2800 0000 – 0x2FFF FFFF	21	0xA800 0000 – 0xAFFF FFFF
6	0x3000 0000 – 0x37FF FFFF	22	0xB000 0000 – 0xB7FF FFFF
7	0x3800 0000 – 0x3FFF FFFF	23	0xB800 0000 – 0xBFFF FFFF
8	0x4000 0000 – 0x47FF FFFF	24	0xC000 0000 – 0xC7FF FFFF
9	0x4800 0000 – 0x4FFF FFFF	25	0xC800 0000 – 0xCFFF FFFF
10	0x5000 0000 – 0x57FF FFFF	26	0xD000 0000 – 0xD7FF FFFF
11	0x5800 0000 – 0x5FFF FFFF	27	0xD800 0000 – 0xDFFF FFFF
12	0x6000 0000 – 0x67FF FFFF	28	0xE000 0000 – 0xE7FF FFFF
13	0x6800 0000 – 0x6FFF FFFF	29	0xE800 0000 – 0xEFFF FFFF
14	0x7000 0000 – 0x77FF FFFF	30	0xF000 0000 – 0xF7FF FFFF
15	0x7800 0000 – 0x7FFF FFFF	31	0xF800 0000 – 0xFFFF FFFF

28 IOP 480 CPU INSTRUCTION SET

Descriptions of the IOP 480 instructions follow. Each description contains the following elements:

- Instruction names (mnemonic and full)
- Instruction syntax
- Instruction format diagram
- Pseudocode description
- Prose description
- Registers altered
- Architecture notes identifying the associated PowerPC Architecture component

Where appropriate, instruction descriptions list invalid instruction forms and provide programming notes.

For a quick reference, Table 28-1 alphabetically lists instructions with concurrent page numbers.

Table 28-1. Alphabetical Instruction Listing with Page Number Cross-Reference

IOP 480 CPU Instruction Set

Table 28-1. Alphabetical Instruction Listing with Page Number Cross-Reference (Continued)

Table 28-1. Alphabetical Instruction Listing with Page Number Cross-Reference (Continued)

Table 28-1. Alphabetical Instruction Listing with Page Number Cross-Reference (Continued)

Instruction	Page Number
rfci	**28-119**
rfi	**28-120**
rlwimi	**28-121**
rlwinm	**28-122**
rlwnm	**28-124**
sc	**28-125**
slw	**28-126**
sraw	**28-127**
srawi	**28-128**
srw	**28-129**
stb	**28-130**
stbu	**28-131**
stbux	**28-132**
stbx	**28-133**
sth	**28-134**
sthbrx	**28-135**
sthu	**28-136**
sthux	**28-137**
sthx	**28-138**
stmw	**28-139**
stswi	**28-140**
stswx	**28-141**
stw	**28-143**
stwbrx	**28-144**
stwcx.	**28-145**
stwu	**28-146**
stwux	**28-147**

Table 28-1. Alphabetical Instruction Listing with Page Number Cross-Reference (Continued)

Instruction	Page Number
stwx	**28-148**
subf	**28-149**
subfc	**28-150**
subfe	**28-151**
subfic	**28-152**
subfme	**28-153**
subfze	**28-154**
sync	**28-155**
tlbia	**28-156**
tlbre	**28-157**
tlbsx	**28-159**
tlbsync	**28-160**
tlbwe	**28-161**
tw	**28-163**
twi	**28-166**
wrtee	**28-169**
wrteei	**28-170**
xor	**28-171**
xori	**28-172**
xoris	**28-173**

28.1 INSTRUCTION SET PORTABILITY

To support embedded real-time applications, the instruction sets of the IOP 480 CPU and other IBM PowerPC 400Series embedded controllers implement the IBM PowerPC Embedded Environment. This additional instruction set is not part of the PowerPC Architecture defined in *The PowerPC Architecture: A Specification for a New Family of RISC Processo*rs.

Programs using these instructions are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment. Table 28-2 lists instructions in the IBM PowerPC Embedded Environment that are implemented in the IOP 480 CPU.

Table 28-2. Instructions in the IBM PowerPC Embedded Environment

dccci	**mfdcr**
dcread	**mtdcr**
iccci	**rfci**
icbt	**tlbre**
icread	**tlbsx**
	tlbsx.
	tlbwe
	wrtee
	wrteei

28.2 INSTRUCTION FORMATS

For more detailed information about instruction formats, including a summary of instruction field usage and instruction format diagrams for the IOP 480 CPU, see Appendix A.3, “Instruction Formats,” on page A-36.

Instructions are four bytes long. Instruction addresses are always Lword-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions have an extended opcode in another field. The remaining instruction bits contain additional fields. All instruction fields belong to one of the following categories:

- Defined

  These instructions contain values, such as opcodes, that cannot be altered. The instruction format diagrams specify the values of defined fields.

- Variable

  These fields contain operands, such as general purpose register selectors and immediate values, that may vary from execution to execution. The instruction format diagrams specify the operands in variable fields.

- Reserved

  Bits in a ***reserved*** field should be set to 0. In the instruction format diagrams, reserved fields are shaded.

If any bit in a defined field does not contain the expected value, the instruction is illegal and an illegal instruction exception occurs. If any bit in a reserved field does not contain 0, the instruction form is invalid and its result is architecturally undefined. Unless otherwise noted, the IOP 480 CPU execute all invalid instruction forms without causing an illegal instruction exception.

28.3 PSEUDOCODE

The pseudocode that appears in the instruction descriptions provides a semi-formal language for describing instruction operations.

The pseudocode uses the following notation:

$\leftarrow$	Assignment.
$\wedge$	AND logical operator.
$\neg$	NOT logical operator.
$\vee$	OR logical operator.
$\oplus$	Exclusive-OR (XOR) logical operator.
+	*Twos* complement addition.
–	*Twos* complement subtraction, unary minus.
$\times$	Multiplication.
$\div$	Division yielding a quotient.

%	Remainder of an integer division; (33% 32) = 1.
||	Concatenation.
=, ≠	Equal, not equal relations.
<, >	Signed comparison relations.
$\overset{u}{<}$, $\overset{u}{>}$	Unsigned comparison relations.
if...then...else...	Conditional execution; if *condition* then a else *b*, where *a* and *b* represent one or more pseudocode statements. Indenting indicates the ranges of *a* and *b*. If *b* is null, the else does not appear.
do	Do loop. “to” and “by” clauses specify incrementing an iteration variable; “while” and “until” clauses specify terminating conditions. Indenting indicates the range of the loop.
leave	Leave innermost do loop or do loop specified in a leave statement.
n	Decimal number.
0xn	Hexadecimal number.
0bn	Binary number.
FLD	Instruction field.
FLD_b	Bit in a named instruction field.
$FLD_{b:b}$	Range of bits in a named instruction field.
$FLD_{b,b,\ldots}$	List of bits, by number or name, in a named instruction field.
REG_b	Bit in a named register.
$REG_{b:b}$	Range of bits in a named register.
$REG_{b,b,\ldots}$	List of bits, by number or name, in a named register.
REG[FLD]	Field in a named register.
REG[FLD, FLD . . .]	List of fields in a named register.
REG[FLD:FLD]	Range of fields in a named register.
GPR(r)	General Purpose Register (GPR) r, where $0 \le r \le 31$.
(GPR(r))	Contents of GPR r, where $0 \le r \le 31$.
DCR(DCRN)	Device Control Register (DCR) specified by the DCRF field in an **mfdcr** or **mtdcr** instruction.
SPR(SPRN)	SPR specified by the SPRF field in an **mfspr** or **mtspr** instruction.
RA, RB, . . .	GPRs.
(Rx)	Contents of a GPR, where x is A, B, S, or T.
(RA|0)	Contents of the register RA or 0, if the RA field is 0.
$c_{0:3}$	Four-bit object used to store condition results in compare instructions.
^{n}b	Bit or bit value *b* is replicated *n* times.
xx	Bit positions which are don’t cares.
CEIL(x)	Least integer ≥ x.

EXTS(x)	The result of extending x on the left with sign bits.
PC	Program counter.
RESERVE	Reserve bit; indicates whether a process has reserved a block of storage.
CIA	Current instruction address; the 32-bit address of the instruction being described by a sequence of pseudocode. This address is used to set the next instruction address (NIA). Does not correspond to any architected register.
NIA	Next instruction address; the 32-bit address of the next instruction to be executed. In pseudo-code, a successful branch is indicated by assigning a value to NIA. For instructions that do not branch, the NIA is CIA +4.
MS(addr, n)	Number of bytes represented by n at the location in main storage represented by addr.
EA	Effective address; the 32-bit address, derived by applying indexing or indirect addressing rules to the specified operand, that specifies an location in main storage.
ROTL((RS),n)	Rotate left; the contents of RS are shifted left the number of bits specified by *n*.
MASK(MB,ME)	Mask having 1s in positions MB through ME (wrapping if MB > ME) and 0's elsewhere.
instruction(EA)	Instruction operating on a data or instruction cache block associated with an EA.

The following table lists the pseudocode operators and their associativity in descending order of precedence:

Table 28-3. Operator Precedence

Operators	Associativity
REG_b, REG[FLD], function evaluation	Left to right
^{n}b	Right to left
$\neg$, – (unary minus)	Right to left
$\times$, $\div$	Left to right
+, –	Left to right
$\|\|$	Left to right
=, $\neq$, <, >, $\overset{u}{<}$, $\overset{u}{>}$	Left to right
$\wedge$, $\oplus$	Left to right
$\vee$	Left to right
$\leftarrow$	None

28.4 REGISTER USAGE

Each instruction description lists the registers altered by the instruction. Some register changes are explicitly detailed in the instruction description (for example, the target register of a load instruction). Other registers are changed, with the details of the change not included in the instruction description. This category frequently includes the Condition Register (CR) and the Fixed-point Exception Register (XER). For discussion of CR, see Section 24.2.3, "Condition Register (CR)," on page 24-7. For discussion of XER, see Section 24.2.2.3, "Fixed Point Exception Register (XER)," on page 24-5.

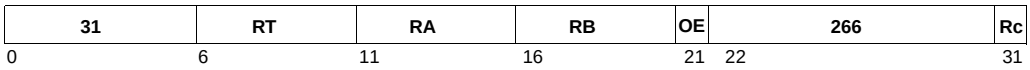
28.5 ALPHABETICAL INSTRUCTION LISTING

The following pages list the instructions available in the IOP 480 CPU in alphabetical order.

add

Add

add	**RT, RA, RB**	OE=0, Rc=0
add.	**RT, RA, RB**	OE=0, Rc=1
addo	**RT, RA, RB**	OE=1, Rc=0
addo.	**RT, RA, RB**	OE=1, Rc=1



(RT) ← (RA) + (RB)

The sum of the contents of register RA and the contents of register RB is placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

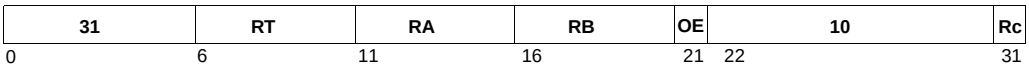
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

addc

Add Carrying

addc	**RT, RA, RB**	OE=0, Rc=0
addc.	**RT, RA, RB**	OE=0, Rc=1
addco	**RT, RA, RB**	OE=1, Rc=0
addco.	**RT, RA, RB**	OE=1, Rc=1



(RT) ← (RA) + (RB)
if (RA) + (RB) $\overset{u}{>}$ $2^{32} - 1$ then
    XER[CA] ← 1
else
    XER[CA] ← 0

The sum of the contents of register RA and register RB is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

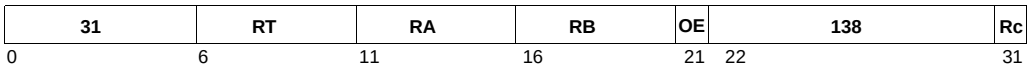
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

adde

Add Extended

adde	**RT, RA, RB**	OE=0, Rc=0
adde.	**RT, RA, RB**	OE=0, Rc=1
addeo	**RT, RA, RB**	OE=1, Rc=0
addeo.	**RT, RA, RB**	OE=1, Rc=1



(RT) ← (RA) + (RB) + XER[CA]
if (RA) + (RB) + XER[CA] $\overset{u}{>} 2^{32} - 1$ then
    XER[CA] ← 1
else
    XER[CA] ← 0

The sum of the contents of register RA, register RB, and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

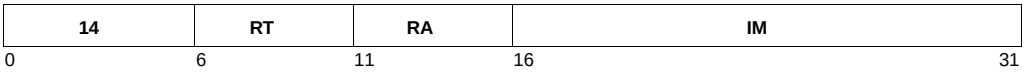
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

addi

Add Immediate

addi **RT, RA, IM**



(RT) ← (RA|0) + EXTS(IM)

If the RA field is 0, the IM field, sign-extended to 32 bits, is placed into register RT.

If the RA field is nonzero, the sum of the contents of register RA and the contents of the IM field, sign-extended to 32 bits, is placed into register RT.

Registers Altered

- RT

Programming Note

To place an immediate, sign-extended value into the GPR specified by the RT field, set the RA field to 0.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

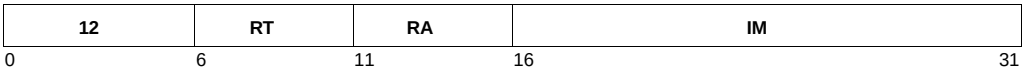
Table 28-4. Extended Mnemonics for addi

Mnemonic	Operands	Function	Other Registers Changed
lal	RT, D(RA)	Load address (RA ≠ 0); D is an offset from a base address that is assumed to be (RA). (RT) ← (RA) + EXTS(D) Extended mnemonic for addi RT,RA,D	
li	RT, IM	Load immediate. (RT) ← EXTS(IM) Extended mnemonic for addi RT,0,IM	
subi	RT, RA, IM	Subtract EXTS(IM) from (RA|0). Place result in RT. Extended mnemonic for addi RT,RA,–IM	

addic

Add Immediate Carrying

addic RT, RA, IM



(RT) ← (RA) + EXTS(IM)
if (RA) + EXTS(IM) $\overset{u}{\geq} 2^{32} - 1$ then
    XER[CA] ← 1
else
    XER[CA] ← 0

The sum of the contents of register RA and the contents of the IM field, sign-extended to 32 bits, is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

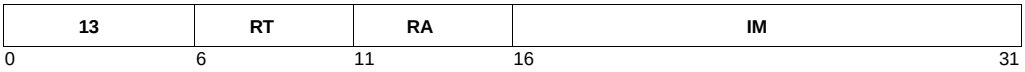
Table 28-5. Extended Mnemonics for addic

Mnemonic	Operands	Function	Other Registers Changed
subic	RT, RA, IM	Subtract EXTS(IM) from (RA) Place result in RT; place carry-out in XER[CA]. Extended mnemonic for addic RT,RA,–IM	

addic.

Add Immediate Carrying and Record

addic. **RT, RA, IM**



(RT) ← (RA) + EXTS(IM)
if (RA) + EXTS(IM) $\overset{u}{\geq} 2^{32} - 1$ then
    XER[CA] ← 1
else
    XER[CA] ← 0

The sum of the contents of register RA and the contents of the IM field, sign-extended to 32 bits, is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$

Programming Note

addic. is one of three instructions that implicitly update CR[CR0] without having an RC field. The other instructions are **andi.** and **andis.**

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

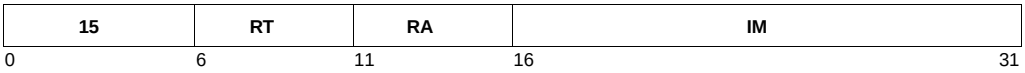
Table 28-6. Extended Mnemonics for addic.

Mnemonic	Operands	Function	Other Registers Changed
subic.	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT; place carry-out in XER[CA]. Extended mnemonic for addic. RT,RA,–IM	CR[CR0]

addis

Add Immediate Shifted

addis **RT, RA, IM**



(RT) ← (RA|0) + (IM || ^{16}0)

If the RA field is 0, the IM field is concatenated on its right with sixteen 0-bits and placed into register RT.

If the RA field is nonzero, the contents of register RA are added to the contents of the extended IM field. The sum is stored into register RT.

Registers Altered

- RT

Programming Note

An **addi** instruction stores a sign-extended 16-bit value in a GPR. An **addis** instruction followed by an **ori** instruction stores an arbitrary 32-bit value in a GPR, as shown in the following example:

```
addis    RT, 0, high 16 bits of value
ori      RT, RT, low 16 bits of value
```

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

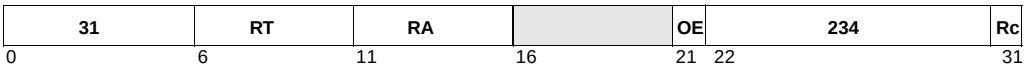
Table 28-7. Extended Mnemonics for addis

Mnemonic	Operands	Function	Other Registers Changed
lis	RT, IM	Load immediate shifted. (RT) ← (IM || ^{16}0) Extended mnemonic for addis RT,0,IM	
subis	RT, RA, IM	Subtract (IM || ^{16}0) from (RA|0). Place result in RT. Extended mnemonic for addis RT,RA,–IM	

addme

Add to Minus One Extended

addme	**RT, RA**	OE=0, Rc=0
addme.	**RT, RA**	OE=0, Rc=1
addmeo	**RT, RA**	OE=1, Rc=0
addmeo.	**RT, RA**	OE=1, Rc=1



(RT) ← (RA) + XER[CA] + (–1)
if (RA) + XER[CA] + 0xFFFF FFFF $\overset{u}{>}$ 2^{32} – 1 then
    XER[CA] ← 1
else
    XER[CA] ← 0

The sum of the contents of register RA, XER[CA], and –1 is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Invalid Instruction Forms

- Reserved fields

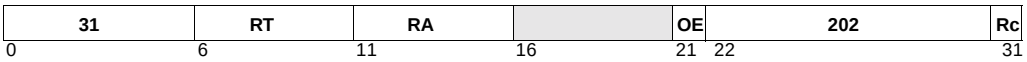
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

addze

Add to Zero Extended

addze	**RT, RA**	OE=0, Rc=0
addze.	**RT, RA**	OE=0, Rc=1
addzeo	**RT, RA**	OE=1, Rc=0
addzeo.	**RT, RA**	OE=1, Rc=1



(RT) $\leftarrow$ (RA) + XER[CA]
if (RA) + XER[CA] $\overset{u}{>}$ $2^{32} - 1$ then
    XER[CA] $\leftarrow$ 1
else
    XER[CA] $\leftarrow$ 0

The sum of the contents of register RA and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the add operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Invalid Instruction Forms

- Reserved fields

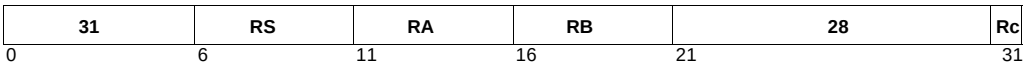
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

and

AND

and	**RA, RS, RB**	Rc=0
and.	**RA, RS, RB**	Rc=1



(RA) ← (RS) ∧ (RB)

The contents of register RS is ANDed with the contents of register RB and the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

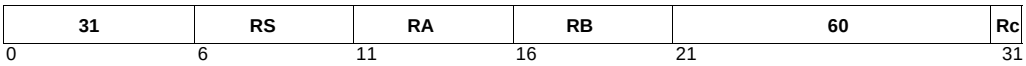
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

andc

AND with Complement

andc	**RA,RS,RB**	Rc=0
andc.	**RA,RS,RB**	Rc=1



(RA) $\leftarrow$ (RS) $\wedge$ $\neg$(RB)

The contents of register RS is ANDed with the ones complement of the contents of register RB; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

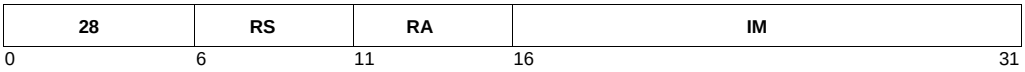
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

andi.

AND Immediate

andi. **RA, RS, IM**



(RA) $\leftarrow$ (RS) $\wedge$ ($^{16}0$ || IM)

The IM field is extended to 32 bits by concatenating 16 0-bits on its left. The contents of register RS is ANDed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$

Programming Note

The **andi.** instruction can test whether any of the 16 least-significant bits in a GPR are 1-bits.

andi. is one of three instructions that implicitly update CR[CR0] without having an Rc field. The other instructions are **addic.** and **andis.**.

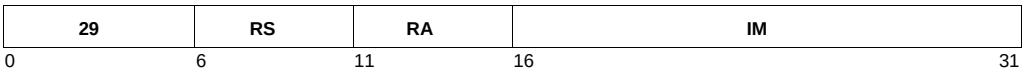
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

andis.

AND Immediate Shifted

andis. **RA, RS, IM**



(RA) $\leftarrow$ (RS) $\wedge$ (IM $\|$ $^{16}0$)

The IM field is extended to 32 bits by concatenating 16 0-bits on its right. The contents of register RS are ANDed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$

Programming Note

The **andis.** instruction can test whether any of the 16 most-significant bits in a GPR are 1-bits.

andis. is one of three instructions that implicitly update CR[CR0] without having an Rc field. The other instructions are **addic.** and **andi.**.

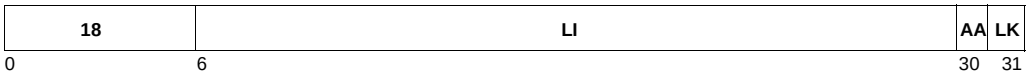
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

b

Branch

b	**target**	AA=0, LK=0
ba	**target**	AA=1, LK=0
bl	**target**	AA=0, LK=1
bla	**target**	AA=1, LK=1




```
If AA = 1 then
   LI ← target6:29
   NIA ← EXTS(LI || 20)
else
   LI ← (target – CIA)6:29
   NIA ← CIA + EXTS(LI || 20)
if LK = 1 then
   (LR) ← CIA + 4
PC ← NIA
```

The next instruction address (NIA) is the effective address of the branch. The NIA is formed by adding a displacement to a base address. The displacement is obtained by concatenating two 0-bits to the right of the LI field and sign-extending the result to 32 bits.

If the AA field contains 0, the base address is the address of the branch instruction, which is also the current instruction address (CIA). If the AA field contains 1, the base address is 0.

Program flow is transferred to the NIA.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- LR if LK contains 1

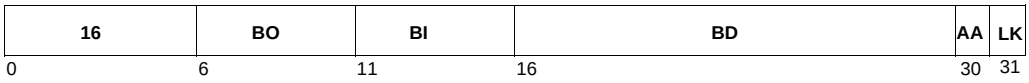
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

bc

Branch Conditional

bc	**BO, BI, target**	AA=0, LK=0
bca	**BO, BI, target**	AA=1, LK=0
bcl	**BO, BI, target**	AA=0, LK=1
bcla	**BO, BI, target**	AA=1, LK=1




```
if BO2 = 0 then
    CTR ← CTR – 1
if (BO2 = 1 ∨ ((CTR = 0) = BO3)) ∧ (BO0 = 1 ∨ (CRBI = BO1))  then
    if AA = 1 then
        BD ← target16:29
        NIA ← EXTS(BD || 20)
    else
        BD ← (target – CIA)16:29
        NIA ← CIA + EXTS(BD || 20)
else
    NIA ← CIA + 4
if LK = 1 then
    (LR) ← CIA + 4
PC ← NIA
```

If bit 2 of the BO field contains 0, the CTR is decremented.

The BI field specifies a bit in the CR to be used as the condition of the branch.

The next instruction address (NIA) is the effective address of the branch. The NIA is formed by adding a displacement to a base address. The displacement is obtained by concatenating two 0-bits to the right of the BD field and sign-extending the result to 32 bits.

If the AA field contains 0, the base address is the address of the branch instruction, which is also the current instruction address (CIA). If the AA field contains 1, the base address is 0.

The BO field controls options that determine when program flow is transferred to the NIA. The BO field also controls Branch Prediction, a performance-improvement feature. See Section 24.6.4, “BO Field on Conditional Branches,” on page 24-22, and Section 24.6.5, “Branch Prediction,” on page 24-23, for a complete discussion.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- CTR if BO_2 contains 0
- LR if LK contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 28-8. Extended Mnemonics for bc, bca, bcl, bcla

Mnemonic	Operands	Function	Other Registers Changed
bdnz	target	Decrement CTR; branch if CTR ≠ 0. Extended mnemonic for bc 16,0,target	
bdnza		Extended mnemonic for bca 16,0,target	
bdnzl		Extended mnemonic for bcl 16,0,target	(LR) ← CIA + 4.
bdnzla		Extended mnemonic for bcla 16,0,target	(LR) ← CIA + 4.
bdnzf	cr_bit, target	Decrement CTR; branch if CTR ≠ 0 AND CR_{cr_bit} = 0. Extended mnemonic for bc 0,cr_bit,target	
bdnzfa		Extended mnemonic for bca 0,cr_bit,target	
bdnzfl		Extended mnemonic for bcl 0,cr_bit,target	(LR) ← CIA + 4.
bdnzfla		Extended mnemonic for bcla 0,cr_bit,target	(LR) ← CIA + 4.
bdnzt	cr_bit, target	Decrement CTR; branch if CTR ≠ 0 AND CR_{cr_bit} = 1. Extended mnemonic for bc 8,cr_bit,target	
bdnzta		Extended mnemonic for bca 8,cr_bit,target	
bdnztl		Extended mnemonic for bcl 8,cr_bit,target	(LR) ← CIA + 4.
bdnztla		Extended mnemonic for bcla 8,cr_bit,target	(LR) ← CIA + 4.
bdz	target	Decrement CTR; branch if CTR = 0. Extended mnemonic for bc 18,0,target	
bdza		Extended mnemonic for bca 18,0,target	
bdzl		Extended mnemonic for bcl 18,0,target	(LR) ← CIA + 4.
bdzla		Extended mnemonic for bcla 18,0,target	(LR) ← CIA + 4.
bdzf	cr_bit, target	Decrement CTR; branch if CTR = 0 AND CR_{cr_bit} = 0. Extended mnemonic for bc 2,cr_bit,target	
bdzfa		Extended mnemonic for bca 2,cr_bit,target	
bdzfl		Extended mnemonic for bcl 2,cr_bit,target	(LR) ← CIA + 4.
bdzfla		Extended mnemonic for bcla 2,cr_bit,target	(LR) ← CIA + 4.

Table 28-8. Extended Mnemonics for bc, bca, bcl, bcla

Mnemonic	Operands	Function	Other Registers Changed
bdzt	cr_bit, target	Decrement CTR; branch if CTR = 0 AND CR_{cr_bit} = 1. Extended mnemonic for bc 10,cr_bit,target	
bdzta		Extended mnemonic for bca 10,cr_bit,target	
bdztl		Extended mnemonic for bcl 10,cr_bit,target	(LR) ← CIA + 4.
bdztla		Extended mnemonic for bcla 10,cr_bit,target	(LR) ← CIA + 4.
beq	[cr_field,] target	Branch if equal; use CR0 if cr_field is omitted. Extended mnemonic for bc 12,4∗cr_field+2,target	
beqa		Extended mnemonic for bca 12,4∗cr_field+2,target	
beql		Extended mnemonic for bcl 12,4∗cr_field+2,target	(LR) ← CIA + 4.
beqla		Extended mnemonic for bcla 12,4∗cr_field+2,target	(LR) ← CIA + 4.
bf	cr_bit, target	Branch if CR_{cr_bit} = 0. Extended mnemonic for bc 4,cr_bit,target	
bfa		Extended mnemonic for bca 4,cr_bit,target	
bfl		Extended mnemonic for bcl 4,cr_bit,target	LR
bfla		Extended mnemonic for bcla 4,cr_bit,target	LR
bge	[cr_field,] target	Branch if greater than or equal; use CR0 if cr_field is omitted. Extended mnemonic for bc 4,4∗cr_field+0,target	
bgea		Extended mnemonic for bca 4,4∗cr_field+0,target	
bgel		Extended mnemonic for bcl 4,4∗cr_field+0,target	LR
bgela		Extended mnemonic for bcla 4,4∗cr_field+0,target	LR
bgt	[cr_field,] target	Branch if greater than; use CR0 if cr_field is omitted. Extended mnemonic for bc 12,4∗cr_field+1,target	
bgta		Extended mnemonic for bca 12,4∗cr_field+1,target	
bgtl		Extended mnemonic for bcl 12,4∗cr_field+1,target	LR
bgtla		Extended mnemonic for bcla 12,4∗cr_field+1,target	LR

Table 28-8. Extended Mnemonics for bc, bca, bcl, bcla

Mnemonic	Operands	Function	Other Registers Changed
ble	[cr_field,] target	Branch if less than or equal; use CR0 if cr_field is omitted. Extended mnemonic for bc 4,4∗cr_field+1,target	
blea		Extended mnemonic for bca 4,4∗cr_field+1,target	
blel		Extended mnemonic for bcl 4,4∗cr_field+1,target	LR
blela		Extended mnemonic for bcla 4,4∗cr_field+1,target	LR
blt	[cr_field,] target	Branch if less than; use CR0 if cr_field is omitted. Extended mnemonic for bc 12,4∗cr_field+0,target	
blta		Extended mnemonic for bca 12,4∗cr_field+0,target	
bltl		Extended mnemonic for bcl 12,4∗cr_field+0,target	(LR) ← CIA + 4.
bltla		Extended mnemonic for bcla 12,4∗cr_field+0,target	(LR) ← CIA + 4.
bne	[cr_field,] target	Branch if not equal; use CR0 if cr_field is omitted. Extended mnemonic for bc 4,4∗cr_field+2,target	
bnea		Extended mnemonic for bca 4,4∗cr_field+2,target	
bnel		**Extended mnemonic for bcl 4,4∗cr_field+2,target**	(LR) ← CIA + 4.
bnela		Extended mnemonic for bcla 4,4∗cr_field+2,target	(LR) ← CIA + 4.
bng	[cr_field,] target	Branch if not greater than; use CR0 if cr_field is omitted. Extended mnemonic for bc 4,4∗cr_field+1,target	
bnga		Extended mnemonic for bca 4,4∗cr_field+1,target	
bngl		Extended mnemonic for bcl 4,4∗cr_field+1,target	(LR) ← CIA + 4.
bngla		Extended mnemonic for bcla 4,4∗cr_field+1,target	(LR) ← CIA + 4.
bnl	[cr_field,] target	Branch if not less than; use CR0 if cr_field is omitted. Extended mnemonic for bc 4,4∗cr_field+0,target	
bnla		Extended mnemonic for bca 4,4∗cr_field+0,target	
bnll		Extended mnemonic for bcl 4,4∗cr_field+0,target	(LR) ← CIA + 4.
bnlla		Extended mnemonic for bcla 4,4∗cr_field+0,target	(LR) ← CIA + 4.

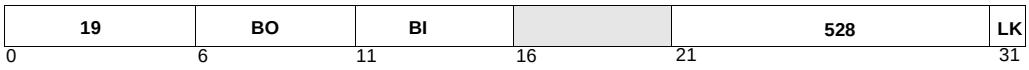
Table 28-8. Extended Mnemonics for bc, bca, bcl, bcla

Mnemonic	Operands	Function	Other Registers Changed
bns	[cr_field,] target	Branch if not summary overflow; use CR0 if cr_field is omitted. Extended mnemonic for bc 4,4∗cr_field+3,target	
bnsa		Extended mnemonic for bca 4,4∗cr_field+3,target	
bnsl		Extended mnemonic for bcl 4,4∗cr_field+3,target	(LR) ← CIA + 4.
bnsla		Extended mnemonic for bcla 4,4∗cr_field+3,target	(LR) ← CIA + 4.
bnu	[cr_field,] target	Branch if not unordered; use CR0 if cr_field is omitted. Extended mnemonic for bc 4,4∗cr_field+3,target	
bnua		Extended mnemonic for bca 4,4∗cr_field+3,target	
bnul		Extended mnemonic for bcl 4,4∗cr_field+3,target	(LR) ← CIA + 4.
bnula		Extended mnemonic for bcla 4,4∗cr_field+3,target	(LR) ← CIA + 4.
bso	[cr_field,] target	Branch if summary overflow; use CR0 if cr_field is omitted. Extended mnemonic for bc 12,4∗cr_field+3,target	
bsoa		**Extended mnemonic for bca 12,4∗cr_field+3,target**	
bsol		Extended mnemonic for bcl 12,4∗cr_field+3,target	(LR) ← CIA + 4.
bsola		Extended mnemonic for bcla 12,4∗cr_field+3,target	(LR) ← CIA + 4.
bt	cr_bit, target	Branch if CR_{cr_bit} = 1. Extended mnemonic for bc 12,cr_bit,target	
bta		Extended mnemonic for bca 12,cr_bit,target	
btl		Extended mnemonic for bcl 12,cr_bit,target	(LR) ← CIA + 4.
btla		Extended mnemonic for bcla 12,cr_bit,target	(LR) ← CIA + 4.
bun	[cr_field], target	Branch if unordered; use CR0 if cr_field is omitted. Extended mnemonic for bc 12,4∗cr_field+3,target	
buna		Extended mnemonic for bca 12,4∗cr_field+3,target	
bunl		Extended mnemonic for bcl 12,4∗cr_field+3,target	(LR) ← CIA + 4.
bunla		Extended mnemonic for bcla 12,4∗cr_field+3,target	(LR) ← CIA + 4.

bcctr

Branch Conditional to Count Register

bcctr	**BO, BI**	LK = 0
bcctrl	**BO, BI**	LK = 1




```
if BO2 = 0 then
   CTR ← CTR – 1
if (BO2 = 1 ∨ ((CTR = 0) = BO3)) ∧ (BO0 = 1 ∨ (CRBI = BO1))  then
   NIA ← CTR0:29 || 20
else
   NIA ← CIA + 4
if LK = 1 then
   (LR) ← CIA + 4
PC ← NIA
```

The BI field specifies a bit in the CR to be used as the condition of the branch.

The next instruction address (NIA) is the target address of the branch. The NIA is formed by concatenating the 30 most significant bits of the CTR with two 0-bits on the right.

The BO field controls options that determine when program flow is transferred to the NIA. The BO field also controls Branch Prediction, a performance-improvement feature. See Section 24.6.4, “BO Field on Conditional Branches,” on page 24-22, and Section 24.6.5, “Branch Prediction,” on page 24-23, for a complete discussion.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- CTR if BO_2 contains 0
- LR if LK contains 1

Invalid Instruction Forms

- Reserved fields
- If bit 2 of the BO field contains 0, the instruction form is invalid, but the pseudocode applies. If the branch condition is true, the branch is taken; the NIA is the contents of the CTR after it is decremented.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 28-9. Extended Mnemonics for bcctr, bcctrl

Mnemonic	Operands	Function	Other Registers Changed
bctr		Branch unconditionally to address in CTR. Extended mnemonic for bcctr 20,0	
bctrl		Extended mnemonic for bcctrl 20,0	(LR) ← CIA + 4.
beqctr	[cr_field]	Branch, if equal, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 12,4∗cr_field+2	
beqctrl		Extended mnemonic for bcctrl 12,4∗cr_field+2	(LR) ← CIA + 4.
bfctr	cr_bit	Branch, if CR_{cr_bit} = 0, to address in CTR. Extended mnemonic for bcctr 4,cr_bit	
bfctrl		Extended mnemonic for bcctrl 4,cr_bit	(LR) ← CIA + 4.
bgectr	[cr_field]	Branch, if greater than or equal, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 4,4∗cr_field+0	
bgectrl		Extended mnemonic for bcctrl 4,4∗cr_field+0	(LR) ← CIA + 4.
bgtctr	[cr_field]	Branch, if greater than, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 12,4∗cr_field+1	
bgtctrl		Extended mnemonic for bcctrl 12,4∗cr_field+1	(LR) ← CIA + 4.
blectr	[cr_field]	Branch, if less than or equal, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 4,4∗cr_field+1	
blectrl		Extended mnemonic for bcctrl 4,4∗cr_field+1	(LR) ← CIA + 4.
bltctr	[cr_field]	Branch, if less than, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 12,4∗cr_field+0	
bltctrl		Extended mnemonic for bcctrl 12,4∗cr_field+0	(LR) ← CIA + 4.
bnectr	[cr_field]	Branch, if not equal, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 4,4∗cr_field+2	
bnectrl		Extended mnemonic for bcctrl 4,4∗cr_field+2	(LR) ← CIA + 4.
bngctr	[cr_field]	Branch, if not greater than, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 4,4∗cr_field+1	
bngctrl		Extended mnemonic for bcctrl 4,4∗cr_field+1	(LR) ← CIA + 4.

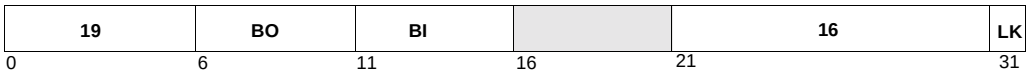
Table 28-9. Extended Mnemonics for bcctr, bcctrl

Mnemonic	Operands	Function	Other Registers Changed
bnlctr	[cr_field]	Branch, if not less than, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 4,4∗cr_field+0	
bnlctrl		Extended mnemonic for bcctrl 4,4∗cr_field+0	(LR) ← CIA + 4.
bnsctr	[cr_field]	Branch, if not summary overflow, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 4,4∗cr_field+3	
bnsctrl		Extended mnemonic for bcctrl 4,4∗cr_field+3	(LR) ← CIA + 4.
bnuctr	[cr_field]	Branch, if not unordered, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 4,4∗cr_field+3	
bnuctrl		Extended mnemonic for bcctrl 4,4∗cr_field+3	(LR) ← CIA + 4.
bsoctr	[cr_field]	Branch, if summary overflow, to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 12,4∗cr_field+3	
bsoctrl		Extended mnemonic for bcctrl 12,4∗cr_field+3	(LR) ← CIA + 4.
btctr	cr_bit	Branch if CR_{cr_bit} = 1 to address in CTR. Extended mnemonic for bcctr 12,cr_bit	
btctrl		Extended mnemonic for bcctrl 12,cr_bit	(LR) ← CIA + 4.
bunctr	[cr_field]	Branch if unordered to address in CTR; use CR0 if cr_field is omitted. Extended mnemonic for bcctr 12,4∗cr_field+3	
bunctrl		Extended mnemonic for bcctrl 12,4∗cr_field+3	(LR) ← CIA + 4.

bclr

Branch Conditional to Link Register

bclr	**BO, BI**	LK = 0
bclrl	**BO, BI**	LK = 1




```
if BO2 = 0 then
   CTR ← CTR – 1
if (BO2 = 1 ∨ ((CTR = 0) = BO3)) ∧ (BO0 = 1 ∨ (CRBI = BO1))  then
   NIA ← LR0:29 || 20
else
   NIA ← CIA + 4
if LK = 1 then
   (LR) ← CIA + 4
PC ← NIA
```

If bit 2 of the BO field contains 0, the CTR is decremented.

The BI field specifies a bit in the CR to be used as the condition of the branch.

The next instruction address (NIA) is the target address of the branch. The NIA is formed by concatenating the 30 most significant bits of the LR with two 0-bits on the right.

The BO field controls options that determine when program flow is transferred to the NIA. The BO field also controls Branch Prediction, a performance-improvement feature. See Section 24.6.4, “BO Field on Conditional Branches,” on page 24-22, and Section 24.6.5, “Branch Prediction,” on page 24-23, for a complete discussion.

If the LK field contains 1, then (CIA + 4) is placed into the LR.

Registers Altered

- CTR if BO_2 contains 0
- LR if LK contains 1

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 28-10. Extended Mnemonics for bclr, bclrl

Mnemonic	Operands	Function	Other Registers Changed
blr		Branch unconditionally to address in LR. Extended mnemonic for bclr 20,0	
blrl		Extended mnemonic for bclrl 20,0	(LR) ← CIA + 4.
bdnzlr		Decrement CTR; branch if CTR ≠ 0, to address in LR. Extended mnemonic for bclr 16,0	
bdnzlrl		Extended mnemonic for bclrl 16,0	(LR) ← CIA + 4.
bdnzflr	cr_bit	Decrement CTR; branch if CTR ≠ 0 AND CR_{cr_bit} = 0 to address in LR. Extended mnemonic for bclr 0,cr_bit	
bdnzflrl		Extended mnemonic for bclrl 0,cr_bit	(LR) ← CIA + 4.
bdnztlr	cr_bit	Decrement CTR; branch if CTR ≠ 0 AND CR_{cr_bit} = 1 to address in LR. Extended mnemonic for bclr 8,cr_bit	
bdnztlrl		Extended mnemonic for bclrl 8,cr_bit	(LR) ← CIA + 4.
bdzlr		Decrement CTR; branch if CTR = 0 to address in LR. Extended mnemonic for bclr 18,0	
bdzlrl		Extended mnemonic for bclrl 18,0	(LR) ← CIA + 4.
bdzflr	cr_bit	Decrement CTR; branch if CTR = 0 AND CR_{cr_bit} = 0 to address in LR. Extended mnemonic for bclr 2,cr_bit	
bdzflrl		***Extended mnemonic for*** **bclrl 2,cr_bit**	(LR) ← CIA + 4.
bdztlr	cr_bit	Decrement CTR; branch if CTR = 0 AND CR_{cr_bit} = 1 to address in LR. Extended mnemonic for bclr 10,cr_bit	
bdztlrl		Extended mnemonic for bclrl 10,cr_bit	(LR) ← CIA + 4.
beqlr	[cr_field]	Branch if equal to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 12,4∗cr_field+2	
beqlrl		Extended mnemonic for bclrl 12,4∗cr_field+2	(LR) ← CIA + 4.
bflr	cr_bit	Branch if CR_{cr_bit} = 0 to address in LR. Extended mnemonic for bclr 4,cr_bit	
bflrl		Extended mnemonic for bclrl 4,cr_bit	(LR) ← CIA + 4.

Table 28-10. Extended Mnemonics for bclr, bclrl

Mnemonic	Operands	Function	Other Registers Changed
bgelr	[cr_field]	Branch, if greater than or equal, to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 4,4∗cr_field+0	
bgelrl		Extended mnemonic for bclrl 4,4∗cr_field+0	(LR) ← CIA + 4.
bgtlr	[cr_field]	Branch, if greater than, to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 12,4∗cr_field+1	
bgtlrl		Extended mnemonic for bclrl 12,4∗cr_field+1	(LR) ← CIA + 4.
blelr	[cr_field]	Branch, if less than or equal, to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 4,4∗cr_field+1	
blelrl		Extended mnemonic for bclrl 4,4∗cr_field+1	(LR) ← CIA + 4.
bltlr	[cr_field]	Branch, if less than, to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 12,4∗cr_field+0	
bltlrl		Extended mnemonic for bclrl 12,4∗cr_field+0	(LR) ← CIA + 4.
bnelr	[cr_field]	Branch, if not equal, to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 4,4∗cr_field+2	
bnelrl		Extended mnemonic for bclrl 4,4∗cr_field+2	(LR) ← CIA + 4.
bnglr	[cr_field]	Branch, if not greater than, to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 4,4∗cr_field+1	
bnglrl		Extended mnemonic for bclrl 4,4∗cr_field+1	(LR) ← CIA + 4.
bnllr	[cr_field]	Branch, if not less than, to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 4,4∗cr_field+0	
bnllrl		Extended mnemonic for bclrl 4,4∗cr_field+0	(LR) ← CIA + 4.
bnslr	[cr_field]	Branch if not summary overflow to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 4,4∗cr_field+3	
bnslrl		Extended mnemonic for bclrl 4,4∗cr_field+3	(LR) ← CIA + 4.
bnulr	[cr_field]	Branch if not unordered to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 4,4∗cr_field+3	
bnulrl		Extended mnemonic for bclrl 4,4∗cr_field+3	(LR) ← CIA + 4.

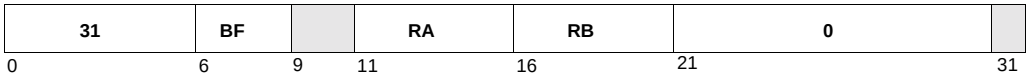
Table 28-10. Extended Mnemonics for bclr, bclrl

Mnemonic	Operands	Function	Other Registers Changed
bsolr	[cr_field]	Branch if summary overflow to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 12,4∗cr_field+3	
bsolrl		Extended mnemonic for bclrl 12,4∗cr_field+3	(LR) ← CIA + 4.
btlr	cr_bit	Branch if CR_{cr_bit} = 1 to address in LR. Extended mnemonic for bclr 12,cr_bit	
btlrl		Extended mnemonic for bclrl 12,cr_bit	(LR) ← CIA + 4.
bunlr	[cr_field]	Branch if unordered to address in LR; use CR0 if cr_field is omitted. Extended mnemonic for bclr 12,4∗cr_field+3	
bunlrl		Extended mnemonic for bclrl 12,4∗cr_field+3	(LR) ← CIA + 4.

cmp

Compare

cmp BF, 0, RA, RB



$c_{0:3} \leftarrow {}^{4}0$
if (RA) < (RB) then $c_0 \leftarrow 1$
if (RA) > (RB) then $c_1 \leftarrow 1$
if (RA) = (RB) then $c_2 \leftarrow 1$
$c_3 \leftarrow$ XER[SO]
n ← BF
CR[CRn] $\leftarrow c_{0:3}$

The contents of register RA are compared with the contents of register RB using a 32-bit signed compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- CR[CRn] where n is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Note

The PowerPC Architecture defines this instruction as **cmp BF,L,RA,RB**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for the IOP 480 CPU, use of the extended mnemonic **cmpw BF,RA,RB** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

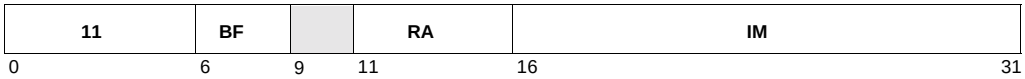
Table 28-11. Extended Mnemonics for cmp

Mnemonic	Operands	Function	Other Registers Changed
cmpw	[BF,] RA, RB	Compare Lword; use CR0 if BF is omitted. Extended mnemonic for cmp BF,0,RA,RB	

cmpi

Compare Immediate

cmpi BF, 0, RA, IM



$c_{0:3} \leftarrow {}^{4}0$
if (RA) < EXTS(IM) then $c_0 \leftarrow 1$
if (RA) > EXTS(IM) then $c_1 \leftarrow 1$
if (RA) = EXTS(IM) then $c_2 \leftarrow 1$
$c_3 \leftarrow$ XER[SO]
n ← BF
CR[CRn] ← $c_{0:3}$

The IM field is sign-extended to 32 bits. The contents of register RA are compared with the extended IM field, using a 32-bit signed compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

Registers Altered

- CR[CRn] where n is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Note

The PowerPC Architecture defines this instruction as **cmpi BF,L,RA,IM**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for IOP 480 CPU, use of the extended mnemonic **cmpwi BF,RA,IM** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

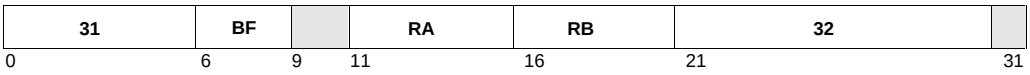
Table 28-12. Extended Mnemonics for cmpi

Mnemonic	Operands	Function	Other Registers Changed
cmpwi	[BF,] RA, IM	Compare Lword Immediate; use CR0 if BF is omitted. Extended mnemonic for cmpi BF,0,RA,IM	

cmpl

Compare Logical

cmpl BF, 0, RA, RB



$c_{0:3} \leftarrow {}^{4}0$
if (RA) $\overset{\geq}{=}$ (RB) then $c_0 \leftarrow 1$
if (RA) $\overset{\leq}{=}$ (RB) then $c_1 \leftarrow 1$
if (RA) = (RB) then $c_2 \leftarrow 1$
$c_3 \leftarrow$ XER[SO]
n $\leftarrow$ BF
CR[CRn] $\leftarrow c_{0:3}$

The contents of register RA are compared with the contents of register RB, using a 32-bit unsigned compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- CR[CRn] where n is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Notes

The PowerPC Architecture defines this instruction as **cmpl BF,L,RA,RB**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for the IOP 480 CPU, use of the extended mnemonic **cmplw BF,RA,RB** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

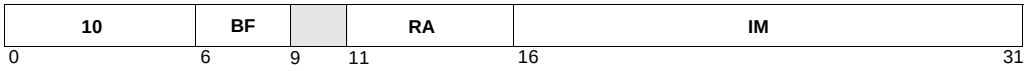
Table 28-13. Extended Mnemonics for cmpl

Mnemonic	Operands	Function	Other Registers Changed
cmplw	[BF,] RA, RB	Compare Logical Lword; use CR0 if BF is omitted. Extended mnemonic for cmpl BF,0,RA,RB	

cmpli

Compare Logical Immediate

cmpli BF, 0, RA, IM



$c_{0:3} \leftarrow {}^{4}0$
if (RA) $<^{u}$ ($^{16}0$ || IM) then $c_0 \leftarrow 1$
if (RA) $>^{u}$ ($^{16}0$ || IM) then $c_1 \leftarrow 1$
if (RA) = ($^{16}0$ || IM) then $c_2 \leftarrow 1$
$c_3 \leftarrow$ XER[SO]
n $\leftarrow$ BF
CR[CRn] $\leftarrow c_{0:3}$

The IM field is extended to 32 bits by concatenating 16 0-bits to its left. The contents of register RA are compared with IM using a 32-bit unsigned compare.

The CR field specified by the BF field is updated to reflect the results of the compare and the value of XER[SO] is placed into the same CR field.

Registers Altered

- CR[CRn] where n is specified by the BF field

Invalid Instruction Forms

- Reserved fields

Programming Note

The PowerPC Architecture defines this instruction as **cmpli BF,L,RA,IM**, where L selects operand size for 64-bit PowerPC implementations. For all 32-bit PowerPC implementations, L = 0 is required (L = 1 is an invalid form); hence for the IOP 480 CPU, use of the extended mnemonic **cmplwi BF,RA,IM** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

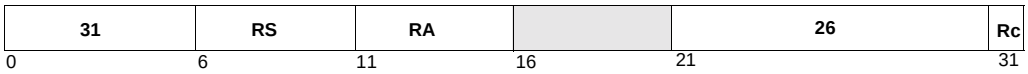
Table 28-14. Extended Mnemonics for cmpli

Mnemonic	Operands	Function	Other Registers Changed
cmplwi	[BF,] RA, IM	Compare Logical Lword Immediate; use CR0 if BF is omitted. Extended mnemonic for cmpli BF,0,RA,IM	

cntlzw

Count Leading Zeros Lword

cntlzw	**RA, RS**	Rc=0
cntlzw.	**RA, RS**	Rc=1




```
n ← 0
do while n < 32
    if (RS)n = 1 then leave
    n ← n + 1
(RA) ← n
```

The consecutive leading 0 bits in register RS are counted; the count is placed into register RA.

The count ranges from 0 through 32, inclusive.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

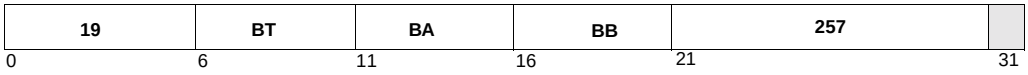
Invalid Instruction Forms

- Reserved fields

crand

Condition Register AND

crand **BT, BA, BB**



$CR_{BT} \leftarrow CR_{BA} \wedge CR_{BB}$

The CR bit specified by the BA field is ANDed with the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

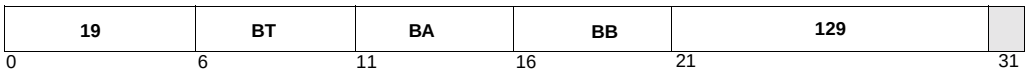
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

crandc

Condition Register AND with Complement

crandc **BT, BA, BB**



$CR_{BT} \leftarrow CR_{BA} \wedge \neg CR_{BB}$

The CR bit specified by the BA field is ANDed with the ones complement of the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

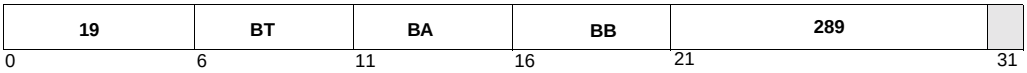
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

creqv

Condition Register Equivalent

creqv **BT, BA, BB**



$CR_{BT} \leftarrow \neg(CR_{BA} \oplus CR_{BB})$

The CR bit specified by the BA field is XORed with the CR bit specified by the BB field; the ones complement of the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

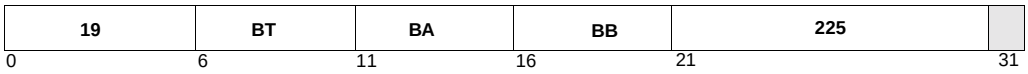
Table 28-15. Extended Mnemonics for creqv

Mnemonic	Operands	Function	Other Registers Changed
crset	bx	Condition register set. Extended mnemonic for creqv bx,bx,bx	

crnand

Condition Register NAND

crnand BT, BA, BB



$CR_{BT} \leftarrow \neg(CR_{BA} \wedge CR_{BB})$

The CR bit specified by the BA field is ANDed with the CR bit specified by the BB field; the ones complement of the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

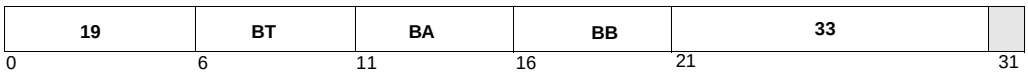
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

crnor

Condition Register NOR

crnor BT, BA, BB



$CR_{BT} \leftarrow \neg(CR_{BA} \vee CR_{BB})$

The CR bit specified by the BA field is ORed with the CR bit specified by the BB field; the ones complement of the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

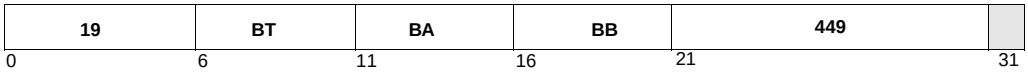
Table 28-16. Extended Mnemonics for crnor

Mnemonic	Operands	Function	Other Registers Changed
crnot	bx, by	Condition register not. Extended mnemonic for crnor bx,by,by	

cror

Condition Register OR

cror **BT, BA, BB**



$CR_{BT} \leftarrow CR_{BA} \vee CR_{BB}$

The CR bit specified by the BA field is ORed with the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

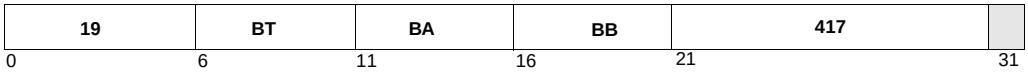
Table 28-17. Extended Mnemonics for cror

Mnemonic	Operands	Function	Other Registers Changed
crmove	bx, by	Condition register move. Extended mnemonic for cror bx,by,by	

crorc

Condition Register OR with Complement

crorc **BT, BA, BB**



$CR_{BT} \leftarrow CR_{BA} \vee \neg CR_{BB}$

The condition register (CR) bit specified by the BA field is ORed with the ones complement of the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

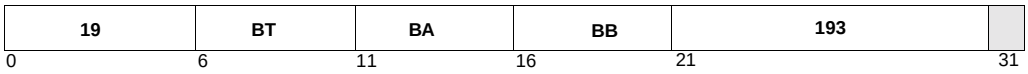
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

crxor

Condition Register XOR

crxor **BT, BA, BB**



$CR_{BT} \leftarrow CR_{BA} \oplus CR_{BB}$

The CR bit specified by the BA field is XORed with the CR bit specified by the BB field; the result is placed into the CR bit specified by the BT field.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

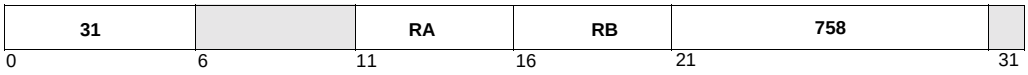
Table 28-18. Extended Mnemonics for crxor

Mnemonic	Operands	Function	Other Registers Changed
crclr	bx	Condition register clear. Extended mnemonic for crxor bx,bx,bx	

dcba

Data Cache Block Allocate

dcba **RA, RB**



EA ← (RA|0) + (RB)
DCBA(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block at the EA is in the data cache and the EA is marked as cacheable and non-write-through, the data in the cache block is architecturally undefined. For the IOP 480 CPU, the cache data block is set to 0.

If the data block at the EA is not in the data cache and the EA is marked as cacheable and not marked as write-through, a cache block is established and set to an architecturally-undefined value. Note that no data is read from main storage, as described in the programming note.

If the data block at the EA is marked as non-cacheable, a no-op occurs.

If the data block at the EA is in the data cache and marked as write-through, architecturally the data in the cache block can remain unmodified. Alternatively, the data block at the EA can be undefined in the data cache and in main storage. For the IOP 480 CPU, a no-op occurs.

If the data block at the EA is not in the data cache and marked as write-through, architecturally the instruction can establish a cache block and set the block to 0, or a no-op can occur. For the IOP 480 CPU, a no-op occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

Because the **dcba** instruction can establish an address in the data cache without copying the contents of that address from main storage, the address established can be invalid with respect to main storage. A subsequent operation may cause the address to be copied back to main storage, for example, to make room for a new cache block; a machine check exception could occur under these circumstances.

dcba provides a hint that a block of storage is soon to be stored or no longer needed; there is no need to retain the data in the block. Establishing the line in the cache, without reading from main storage, improves performance.

Exceptions

This instruction does not cause data storage exceptions (cache line locking or protection) or data TLB-miss exceptions. If conditions occur that would otherwise cause such an exception, **dcba** is treated as a no-op.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more information.

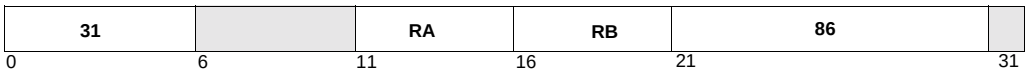
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbf

Data Cache Block Flush

dcbf **RA, RB**



EA ← (RA|0) + (RB)
DCBF(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block corresponding to the EA is in the data cache and marked as modified (stored into), the data block is copied back to main storage and then marked invalid in the data cache. If the data block is not marked as modified, it is simply marked invalid in the data cache. The operation is performed whether or not the EA is marked as cacheable.

If the data block at the EA is not in the data cache, no operation is performed.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Exceptions

This instruction is considered a “load” with respect to data storage exceptions (for the IOP 480 CPU, these are cache line locking exceptions). See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more information.

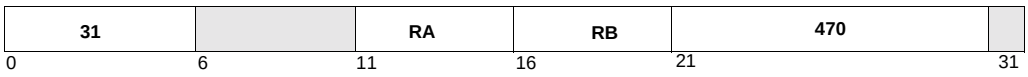
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbi

Data Cache Block Invalidate

dcbi RA, RB



EA ← (RA|0) + (RB)
DCBI(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block at the EA is in the data cache, the data block is marked invalid, regardless of whether or not the EA is marked as cacheable. If modified data existed in the data block prior to the operation of this instruction, that data is lost.

If the data block at the EA is not in the data cache, no operation is performed.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

Execution of this instruction is privileged.

Exceptions

This instruction is considered a “store” with respect to data storage exceptions. See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more information.

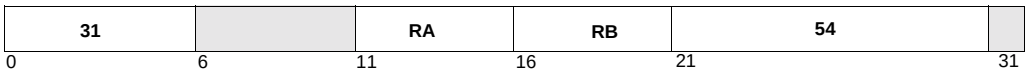
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

dcbst

Data Cache Block Store

dcbst RA, RB



EA ← (RA|0) + (RB)
DCBST(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0, and is the contents of register RA otherwise.

If the data block at the EA is in the data cache and marked as modified, the data block is copied back to main storage and marked as unmodified in the data cache.

If the data block at the EA is in the data cache, and is not marked as modified, or if the data block at the EA is not in the data cache, no operation is performed.

The operation specified by this instruction is performed whether or not the EA is marked as cacheable.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Exceptions

This instruction is considered a “load” with respect to data storage exceptions. See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more Information.

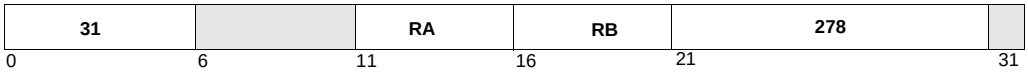
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbt

Data Cache Block Touch

dcbt RA, RB



EA ← (RA|0) + (RB)
DCBT(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

If the data block at the EA is not in the data cache and the EA is marked as cacheable, the block is read from main storage into the data cache.

If the data block at the EA is in the data cache, or if the EA is marked as non-cacheable, no operation is performed.

This instruction is not allowed to cause data storage exceptions or data TLB miss exceptions. If execution of the instruction would cause such an exception, then no operation is performed, and no exception occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

The **dcbt** instruction allows a program to begin a cache block fetch from main storage before the program needs the data. The program can later load data from the cache into registers without incurring the latency of a cache miss.

Exceptions

This instruction is considered a “load” with respect to data storage exceptions. See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more Information.

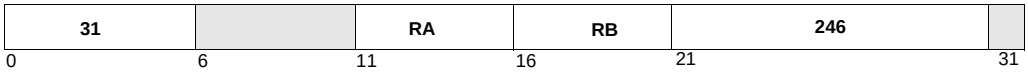
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbtst

Data Cache Block Touch for Store

dcbtst RA, RB



EA ← (RA|0) + (RB)
DCBTST(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block at the EA is not in the data cache and the EA address is marked as cacheable, the data block is loaded into the data cache.

If the EA is marked as non-cacheable, or if the data block at the EA is in the data cache, no operation is performed.

This instruction is not allowed to cause data storage exceptions or data TLB miss exceptions. If execution of the instruction would cause such an exception, then no operation is performed, and no exception occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

The **dcbtst** instruction allows a program to begin a cache block fetch from main storage before the program needs the data. The program can later store data from GPRs into the cache block, without incurring the latency of a cache miss.

Architecturally, **dcbtst** brings data into the cache in “Exclusive” mode, which allows the program to alter the cached data. “Exclusive” mode is part of the MESI protocol for multi-processor systems, and is not implemented. The implementation of the **dcbtst** instruction is identical to the implementation of the **dcbt** instruction.

Exceptions

This instruction is considered a “load” with respect to data storage exceptions. See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more Information.

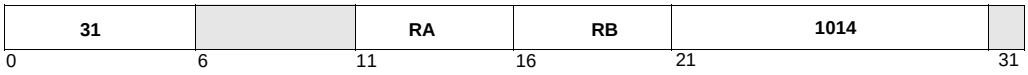
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dcbz

Data Cache Block Set to Zero

dcbz RA, RB



EA ← (RA|0) + (RB)
DCBZ(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the data block at the EA is in the data cache and the EA is marked as cacheable and non-write-through, the data in the cache block is set to 0.

If the data block at the EA is not in the data cache and the EA is marked as cacheable and non-write-through, a cache block is established and set to 0. Note that nothing is read from main storage, as described in the programming note.

If the data block at the EA is marked as either write-through or as non-cacheable, an alignment exception occurs.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

Because the **dcbz** instruction can establish an address in the data cache without copying the contents of that address from main storage, the address established may be invalid with respect to the storage subsystem. A subsequent operation may cause the address to be copied back to main storage, for example, to make room for a new cache block; a machine check exception could occur under these circumstances.

If **dcbz** is attempted to an EA which is marked as non-cacheable, the software alignment exception handler should emulate the instruction by storing zeros to the block in main storage. If a data block corresponding to the EA exists in the cache, but the EA is non-cacheable, stores (including **dcbz**) to that address are considered programming errors (the cache block should previously have been flushed).

If the EA is marked as write-through, the software alignment exception handler should emulate the instruction by storing zeros to the block in main storage. An EA that is marked as write-through required should also be marked as cacheable; when **dcbz** is attempted to such an address, the alignment exception handler should maintain coherency of cache and memory.

Exceptions

An alignment exception occurs if the EA is marked as non-cacheable or as write-through.

This instruction is considered a “store” with respect to data storage exceptions (for the IOP 480 CPU, these are cache line locking exceptions). See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more information.

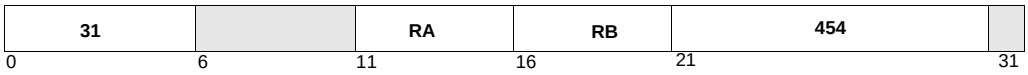
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

dccci

Data Cache Congruence Class Invalidate

dccci RA, RB



EA ← (RA|0) + (RB)
DCCCI(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

Both cache lines in the congruence class specified by $EA_{m:27}$ (where m is the number of bits in the cache array tag field) are invalidated, whether or not they match the EA. If modified data existed in the cache congruence class before the operation of this instruction, that data is lost.

The operation specified by this instruction is performed whether or not the EA is marked as cacheable.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

This instruction is intended for use in the power-on reset routine to invalidate the entire data cache tag array before enabling the data cache. A series of **dccci** instruction should be executed, one for each congruence class. Cacheability can then be enabled.

Exceptions

This instruction is considered a “store” with respect to data storage exceptions. See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

The execution of a **dccci** instruction can cause a data TLB miss exception, at the specified EA, regardless of the non-specific intent of that EA.

This instruction does not cause data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more Information.

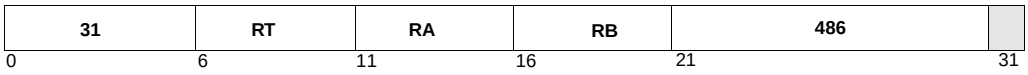
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

dcread

Data Cache Read

dcread RT, RA, RB



EA ← (RA|0) + (RB)
if ((CDBCR[CIS] = 0) ∧ (CDBCR[CWS] = 0)) then (RT) ← (d-cache data, way A)
if ((CDBCR[CIS] = 0) ∧ (CDBCR[CWS] = 1)) then (RT) ← (d-cache data, way B)
if ((CDBCR[CIS] = 1) ∧ (CDBCR[CWS] = 0)) then (RT) ← (d-cache tag, way A)
if ((CDBCR[CIS] = 1) ∧ (CDBCR[CWS] = 1)) then (RT) ← (d-cache tag, way B)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

This instruction is a debugging tool for reading the data cache entries for the congruence class specified by $EA_{m:27}$ (where m is the number of bits in the cache array tag field). The cache information is read into register RT.

If (CDBCR[CIS] = 0), the information is an Lword of data cache data from the addressed congruence class. The Lword is specified by $EA_{28:29}$; $EA_{0:m-1}$ are ignored. If $EA_{30:31}$ are not 00, an alignment exception occurs. If (CDBCR[CWS] = 0), the data is from the A-way, otherwise the data are from the B-way.

If (CDBCR[CIS] = 1), the information is a cache tag from the addressed congruence class; $EA_{0:m-1}$ and $EA_{28:31}$ are ignored. If (CDBCR[CWS] = 0), the tag is from the A-way, otherwise the tag is from the B-way. Data cache tag information is placed into register RT as follows:

Table 28-19. Data Cache Array Tag Information

0:m–1	TAG	Cache Tag	Tag size is determined by CDBCR[DSD]. See Section 25.7, “ICU and DCU Performance Modeling,” on page 25-14, for more information.
m:24		Reserved	The size of this field depends on the size of the TAG field.
25	LK	Cache Line Lock 0 Unlocked 1 Locked	
26	D	Cache Line Dirty 0 Not dirty 1 Dirty	
27	V	Cache Line Valid 0 Not valid 1 Valid	
28:30		Reserved	
31	LRU	Least Recently Used (LRU) 0 A-way LRU 1 B-way LRU	

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

Exceptions

If EA is not Lword-aligned, an alignment exception occurs.

This instruction is considered a “load” with respect to data storage exceptions. See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

The execution of a **dcread** instruction can cause a data TLB miss exception, at the specified EA, regardless of the non-specific intent of that effective address.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions. See Section 26.6.3.1, “Data Address Compare (DAC) Applied to Cache Instructions,” on page 26-7, for more Information.

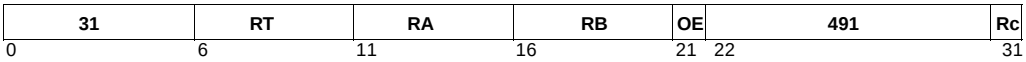
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

divw

Divide Lword

divw	**RT, RA, RB**	OE=0, Rc=0
divw.	**RT, RA, RB**	OE=0, Rc=1
divwo	**RT, RA, RB**	OE=1, Rc=0
divwo.	**RT, RA, RB**	OE=1, Rc=1



(RT) ← (RA) ÷ (RB)

The contents of register RA are divided by the contents of register RB. The quotient is placed into register RT.

Both the dividend and the divisor are interpreted as signed integers. The quotient is the unique signed integer that satisfies:

dividend = (quotient × divisor) + remainder

where the remainder has the same sign as the dividend and its magnitude is less than that of the divisor.

If an attempt is made to perform (0x8000 0000 ÷ –1) or (n ÷ 0), the contents of register RT are undefined; if the Rc field also contains 1, the contents of $CR[CR0]_{LT, GT, EQ}$ are undefined. Either invalid division operation sets XER[OV, SO] to 1 if the OE field contains 1.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[OV, SO] if OE contains 1

Programming Note

The 32-bit remainder can be calculated using the following sequence of instructions:

```
divw    RT,RA,RB        # RT = quotient
mullw   RT,RT,RB        # RT = quotient × divisor
subf    RT,RT,RA        # RT = remainder
```

The sequence does not calculate correct results for the invalid divide operations.

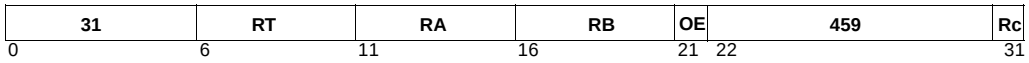
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

divwu

Divide Lword Unsigned

divwu	**RT, RA, RB**	OE=0, Rc=0
divwu.	**RT, RA, RB**	OE=0, Rc=1
divwuo	**RT, RA, RB**	OE=1, Rc=0
divwuo.	**RT, RA, RB**	OE=1, Rc=1



(RT) ← (RA) ÷ (RB)

The contents of register RA are divided by the contents of register RB. The quotient is placed into register RT.

The dividend and the divisor are interpreted as unsigned integers. The quotient is the unique unsigned integer that satisfies

dividend = (quotient × divisor) + remainder

If an attempt is made to perform (n ÷ 0), the contents of register RT are undefined; if the Rc also contains 1, the contents of $CR[CR0]_{LT, GT, EQ}$ are also undefined. The invalid division operation also sets XER[OV, SO] to 1 if the OE field contains 1.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[OV, SO] if OE contains 1

Programming Note

The 32-bit remainder can be calculated using the following sequence of instructions

```
divwu   RT,RA,RB    # RT = quotient
mullw   RT,RT,RB    # RT = quotient × divisor
subf    RT,RT,RA    # RT = remainder
```

This sequence does not calculate the correct result if the divisor is zero.

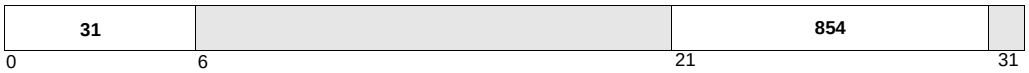
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

eieio

Enforce In Order Execution of I/O

eieio



The **eieio** instruction ensures that all loads and stores preceding an **eieio** instruction complete with respect to main storage before any loads and stores following the **eieio** instruction access main storage.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

Architecturally, **eieio** orders storage access, not instruction completion. Therefore, non-storage operations after **eieio** could complete before storage operations that were before **eieio**. The **sync** instruction guarantees ordering of both instruction completion and storage access. For the IOP 480 CPU, the **eieio** instruction is implemented to behave as a **sync** instruction. To write code which is portable between various PowerPC implementations, programmers should use the mnemonic which corresponds to the desired behavior.

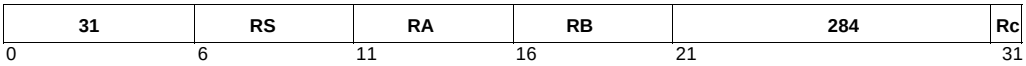
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

eqv

Equivalent

eqv	**RA, RS, RB**	Rc=0
eqv.	**RA, RS, RB**	Rc=1



(RA) ← ¬((RS) ⊕ (RB))

The contents of register RS are XORed with the contents of register RB; the ones complement of the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

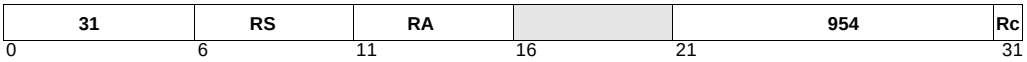
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

extsb

Extend Sign Byte

extsb	**RA, RS**	Rc=0
extsb.	**RA, RS**	Rc=1



(RA) $\leftarrow$ EXTS(RS)$_{24:31}$

The least significant byte of register RS is sign-extended to 32 bits by replicating bit 24 of the register into bits 0 through 23 of the result. The result is placed into register RA.

Registers Altered

- RA
- CR[CR0]$_{LT, GT, EQ, SO}$ if Rc contains 1

Invalid Instruction Forms

- Reserved fields

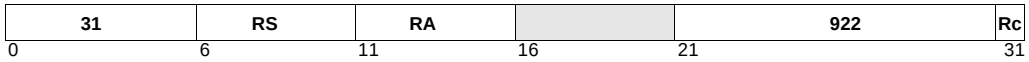
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

extsh

Extend Sign Word

extsh	**RA, RS**	Rc=0
extsh.	**RA, RS**	Rc=1



(RA) ← EXTS(RS)$_{16:31}$

The least significant word of register RS is sign-extended to 32 bits by replicating bit 16 of the register into bits 0 through 15 of the result. The result is placed into register RA.

Registers Altered

- RA
- CR[CR0]$_{LT, GT, EQ, SO}$ if Rc contains 1

Invalid Instruction Forms

- Reserved fields

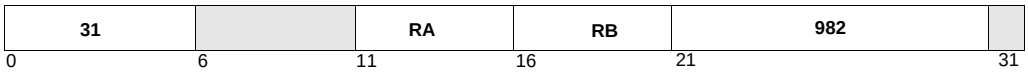
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

icbi

Instruction Cache Block Invalidate

icbi **RA, RB**



EA ← (RA|0) + (RB)
ICBI(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the instruction block at the EA is in the instruction cache, the cache block is marked invalid.

If the instruction block at the EA is not in the instruction cache, no additional operation is performed.

The operation specified by this instruction is performed whether or not the EA is marked as cacheable in the ICCR.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

Instruction cache operations use MSR[DR], not MSR[IR], to determine translation of their operands.

When data translation is disabled, cacheability for the EA of the operand of instruction cache operations is determined by the ICCR, not the DCCR.

Exceptions

Instruction storage exceptions and instruction-side TLB miss exceptions are associated with instruction *fetching*, not with instruction execution. Exceptions that occur during the *execution* of instruction cache operations cause data-side exceptions (data storage exceptions and data TLB miss exceptions).

This instruction is considered a “load” with respect to data storage exceptions (for the IOP 480 CPU, these are cache line locking exceptions). See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions, but does not cause DAC debug events.

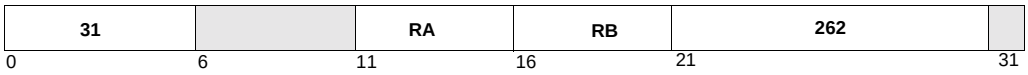
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

icbt

Instruction Cache Block Touch

icbt RA, RB



EA ← (RA|0) + (RB)
ICBT(EA)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

If the instruction block at the EA is not in the instruction cache, and is marked as cacheable, the instruction block is loaded into the instruction cache.

If the instruction block at the EA is in the instruction cache, or if the EA is marked as non-cacheable, no operation is performed.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

Execution of this instruction is privileged.

This instruction allows a program to begin a cache block fetch from main storage before the program needs the instruction. The program can later branch to the instruction address and fetch the instruction from the cache without incurring the latency of a cache miss.

Instruction cache operations use MSR[DR], not MSR[IR], to determine translation of their operands.

When data translation is disabled, cacheability for the effective address of the operand of instruction cache operations is determined by the ICCR, not the DCCR.

Exceptions

Instruction storage exceptions and instruction-side TLB miss exceptions are associated with instruction *fetching*, not with instruction execution. Exceptions that occur during the *execution* of instruction cache operations cause data-side exceptions (data storage exceptions and data TLB miss exceptions).

If the execution of an **icbt** instruction would cause a data TLB miss exception, no operation is performed and no exception occurs.

This instruction is considered a “load” with respect to protection exceptions, but cannot cause a data storage exception.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions, but does not cause DAC debug events.

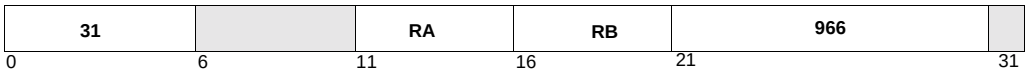
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

iccci

Instruction Cache Congruence Class Invalidate

iccci RA, RB



EA ← (RA|0) + (RB)
ICCCI(EA)

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

Both cache lines in the congruence class specified by $EA_{m-1:27}$ are invalidated, whether or not they match the effective address.

The operation specified by this instruction is performed whether or not the effective address is marked cacheable.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Notes

Execution of this instruction is privileged.

This instruction is intended for use in the power-on reset routine to invalidate the entire cache tag array before enabling the cache. A series of **iccci** instructions should be executed, one for each congruence class. Cacheability can then be enabled.

Instruction cache operations use MSR[DR], not MSR[IR], to determine translation of their operands.

When data translation is disabled, cacheability for the effective address of the operand of instruction cache operations is determined by the ICCR, not the DCCR.

Exceptions

Instruction storage exceptions and instruction-side TLB miss exceptions are associated with instruction *fetching*, not with instruction execution. Exceptions that occur during the *execution* of instruction cache operations cause data-side exceptions (data storage exceptions and data TLB miss exceptions).

The execution of an **iccci** instruction can cause a data TLB miss exception, at the specified effective address, regardless of the non-specific intent of that effective address.

This instruction is considered a “load” with respect to protection exceptions, but cannot cause a data storage exception.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions, but does not cause DAC debug events.

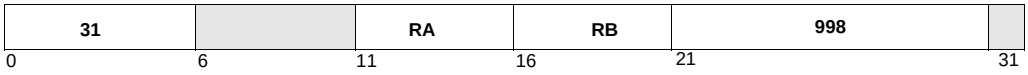
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

icread

Instruction Cache Read

icread RA, RB



EA ← (RA|0) + (RB)
if ((CDBCR[CIS] = 0) ∧ (CDBCR[CWS] = 0)) then (ICDBDR) ← (i-cache data, way A)
if ((CDBCR[CIS] = 0) ∧ (CDBCR[CWS] = 1)) then (ICDBDR) ← (i-cache data, way B)
if ((CDBCR[CIS] = 1) ∧ (CDBCR[CWS] = 0)) then (ICDBDR) ← (i-cache tag, way A)
if ((CDBCR[CIS] = 1) ∧ (CDBCR[CWS] = 1)) then (ICDBDR) ← (i-cache tag, way B)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

This instruction is a debugging tool for reading the instruction cache entries for the congruence class specified by $EA_{m-1:27}$ (where m is the number of bits in the cache array tag field). The cache information is read into the Instruction Cache Debug Data Register (ICDBDR), from where it can be read into a GPR using the extended mnemonic **mficdbdr**.

If CDBCR[CIS] = 0, the information is an Lword of instruction cache data from the addressed line. The Lword is specified by $EA_{28:29}$ ($EA_{0:m}$ and $EA_{30:31}$ are ignored). If CDBCR[CWS] = 0, the data is from the A-way, otherwise from the B-way.

If (CDBCR[CIS] = 1), the information is a cache tag from the addressed congruence class ($EA_{0:m}$ and $EA_{28:31}$ are ignored). If (CDBCR[CWS] = 0), the tag is from the A-way, otherwise from the B-way. Instruction cache tag information is placed in the ICDBDR as follows.

Table 28-20. Instruction Cache Array Tag Information

0:*m*–1	TAG	Cache Tag	See Table 25-1, “Cache Array Size by Core,” on page 25-1 for information on the size of this variable-length field.
m:24		Reserved	The size of this field depends on the size of the tag field.
25	LK	Cache Line Lock 0 Unlocked 1 Locked	
26		Reserved	
27	V	Cache Line Valid 0 Not valid 1 Valid	
28:30		Reserved	
31	LRU	Least Recently Used (LRU) 0 A-way LRU 1 B-way LRU	

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- ICDBDR

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

The instruction pipeline does not automatically wait for data from **icread** to arrive at the ICDBDR before attempting to use the contents of the ICDBDR. Therefore, insert an **isync** instruction between **icread** and **mficdbdr**.

```
icread r5,r6# read cache information
isync    # ensure completion of icread
mficdbdr r7# move information to GPR
```

Instruction cache operations use MSR[DR], not MSR[IR], to determine translation of their operands. When data translation is disabled, cacheability for the EA of the operand of instruction cache operations is determined by the ICCR, not the DCCR.

Exceptions

Instruction storage exceptions and instruction-side TLB miss exceptions are associated with instruction *fetching*, not with instruction execution. Exceptions that occur during the *execution* of instruction cache operations cause data-side exceptions (data storage exceptions and data TLB miss exceptions).

The execution of an **icread** instruction can cause a data TLB miss exception, at the specified EA, regardless of the non-specific intent of that EA.

This instruction is considered a “load” with respect to protection exceptions, but cannot cause a data storage exception.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions, but does not cause DAC debug events.

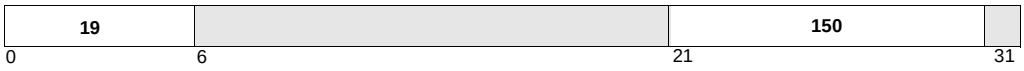
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

isync

Instruction Synchronize

isync



The **isync** instruction is a context synchronizing instruction.

The **isync** instruction provides an ordering function for the effects of all instructions executed by the processor. Executing **isync** insures that all instructions preceding the **isync** instruction have completed before the **isync** instruction completes, except that storage accesses caused by those instructions need not have completed. No subsequent instructions are initiated by the processor until after the **isync** instruction completes. Finally, execution of **isync** causes the processor to discard any prefetched instructions, with the effect that subsequent instructions are fetched and executed in the context established by the instructions preceding the **isync** instruction.

The **isync** instruction has no effect on caches.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

See the discussion of context synchronizing instructions in Section 24.9.1, “Context Synchronization,” on page 24-29.

The following code example illustrates the necessary steps for self-modifying code. This example assumes that addr1 is both data and instruction cacheable.

```
stw     regN, addr1     # the data in regN is to become an instruction at addr1
dcbst   addr1           # forces data from the data cache to memory
sync                    # wait until the data actually reaches the memory
icbi    addr1           # the previous value at addr1 might already be in
                          the instruction cache; invalidate in the cache
isync                   # the previous value at addr1 might already have been
                          pre-fetched into the queue; invalidate the queue
                          so that the instruction must be re-fetched
```

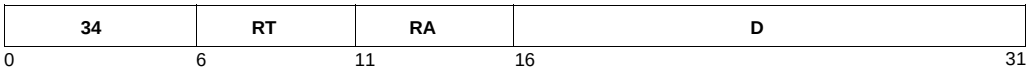
Architecture Note

This instruction is part of the PowerPC Virtual Environment Architecture.

lbz

Load Byte and Zero

lbz **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RT) ← $^{24}0$ || MS(EA,1)

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The byte at the EA is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

Registers Altered

- RT

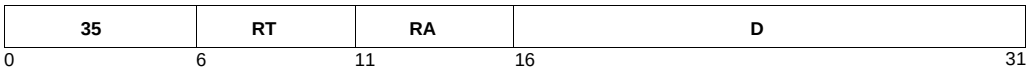
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzu

Load Byte and Zero with Update

lbzu **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RA) ← EA
(RT) ← $^{24}0$ || MS(EA,1)

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The byte at the EA is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA=RT
- RA=0

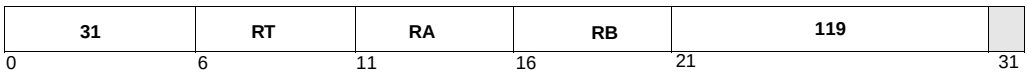
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzux

Load Byte and Zero with Update Indexed

lbzux RT, RA, RB



EA ← (RA|0) + (RB)
(RA) ← EA
(RT) ← $^{24}0$ || MS(EA,1)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The byte at the EA is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA=RT
- RA=0

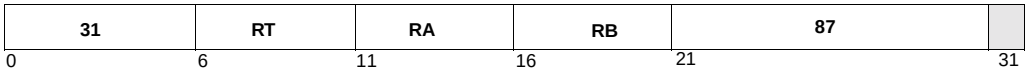
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzx

Load Byte and Zero Indexed

lbzx RT,RA, RB



EA ← (RA|0) + (RB)
(RT) ← $^{24}0$ || MS(EA,1)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The byte at the EA is extended to 32 bits by concatenating 24 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

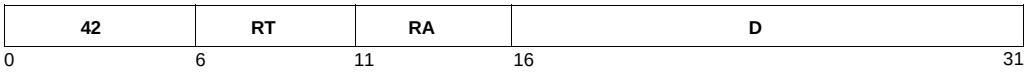
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lha

Load Word Algebraic

lha **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RT) ← EXTS(MS(EA,2))

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The word at the EA is sign-extended to 32 bits and placed into register RT.

Registers Altered

- RT

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

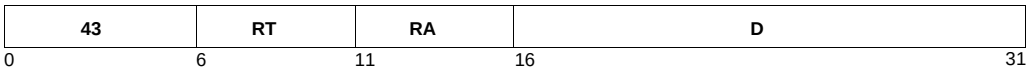
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhau

Load Word Algebraic with Update

lhau **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RA) ← EA
(RT) ← EXTS(MS(EA,2))

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The word at the EA is sign-extended to 32 bits and placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA = RT
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

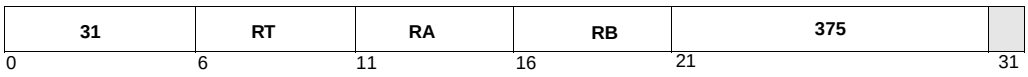
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhaux

Load Word Algebraic with Update Indexed

lhaux **RT, RA, RB**



EA ← (RA|0) + (RB)
(RA) ← EA
(RT) ← EXTS(MS(EA,2))

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The word at the EA is sign-extended to 32 bits and placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA = RT
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

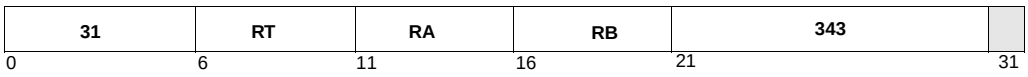
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhax

Load Word Algebraic Indexed

lhax RT, RA, RB



EA ← (RA|0) + (RB)
(RT) ← EXTS(MS(EA,2))

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The word at the EA is sign-extended to 32 bits and placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

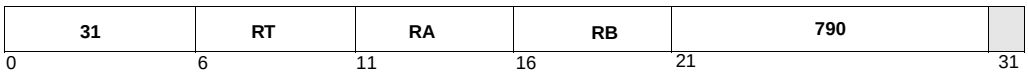
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhbrx

Load Word Byte-Reverse Indexed

lhbrx **RT, RA, RB**



EA ← (RA|0) + (RB)
(RT) ← $^{16}0$ || MS(EA +1,1) || MS(EA,1)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The word at the EA is byte-reversed. The resulting word is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

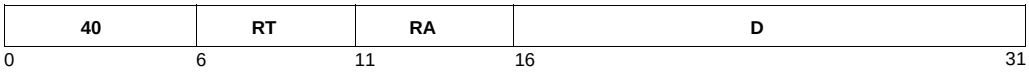
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhz

Load Word and Zero

lhz **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RT) ← $^{16}0$ || MS(EA,2)

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The word at the EA is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

Registers Altered

- RT

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

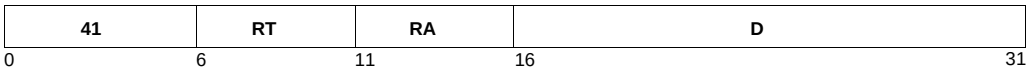
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzu

Load Word and Zero with Update

lhzu **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RA) ← EA
(RT) ← $^{16}0$ || MS(EA,2)

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The word at the EA is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA = RT
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

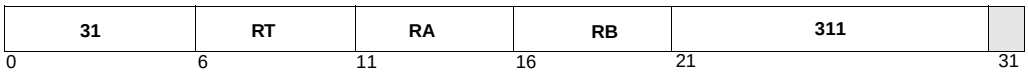
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzux

Load Word and Zero with Update Indexed

lhzux RT, RA, RB



EA ← (RA|0) + (RB)
(RA) ← EA
(RT) ← $^{16}0$ || MS(EA,2)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The word at the EA is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA = RT
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

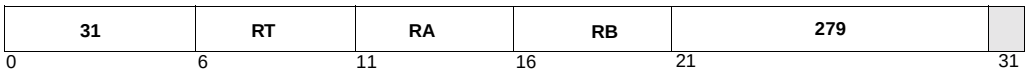
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzx

Load Word and Zero Indexed

lhzx **RT, RA, RB**



EA ← (RA|0) + (RB)
(RT) ← $^{16}0$ || MS(EA,2)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The word at the EA is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

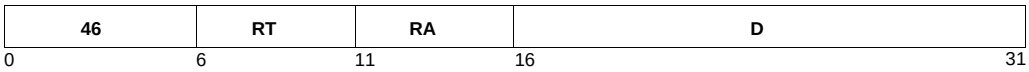
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lmw

Load Multiple Lword

lmw RT, D(RA)




```
EA ← (RA|0) + EXTS(D)
r ← RT
do while r ≤ 31
   if ((r ≠ RA) ∨ (r = 31)) then
      (GPR(r)) ← MS(EA,4)
   r ← r + 1
   EA ← EA + 4
```

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field in the instruction to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

A series of consecutive words starting at the EA are loaded into a set of consecutive GPRs, starting with register RT and continuing to and including GPR(31). Register RA is not altered by this instruction (unless RA is GPR(31), which is an invalid form of this instruction). The Lword which would have been placed into register RA is discarded.

Registers Altered

- RT through GPR(31).

Invalid Instruction Forms

- RA is in the range of registers to be loaded, including the case RA = RT = 0.

Exceptions

If MSR[LE] = 1, an alignment exception occurs.

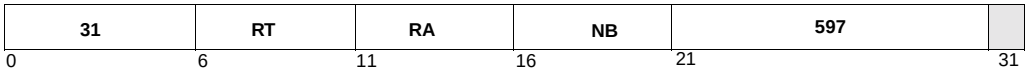
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lswi

Load String Lword Immediate

lswi **RT, RA, NB**




```
EA ← (RA|0)
if NB = 0 then
   CNT ← 32
else
   CNT ← NB
n ← CNT
R_FINAL ← ((RT + CEIL(CNT/4) – 1) % 32)
r ← RT – 1
i ← 0
do while n > 0
   if i = 0 then
      r ← r + 1
      if r = 32 then
         r ← 0
      if ((r ≠ RA) ∨ (r = R_FINAL)) then
         (GPR(r)) ← 0
   if ((r ≠ RA) ∨ (r = R_FINAL)) then
      (GPR(r)_i:i+7) ← MS(EA,1)
   i ← i + 8
   if i = 32 then
      i ← 0
   EA ← EA + 1
   n ← n – 1
```

An effective address (EA) is determined by the RA field. If the RA field contains 0, the EA is 0. Otherwise, the EA is the contents of register RA.

The NB field specifies the byte count CNT. If the NB field contains 0, the byte count is CNT = 32. Otherwise, the byte count is CNT = NB.

A series of CNT consecutive bytes in main storage, starting at the EA, are loaded into CEIL(CNT/4) consecutive GPRs, four bytes per GPR, until the byte count is exhausted. Bytes are loaded into GPRs; the byte at the lowest address is loaded into the most significant byte. Bits to the right of the last byte loaded into the last GPR are set to 0.

The set of loaded GPRs starts at register RT, continues consecutively through GPR(31), wraps to register 0, loading until the byte count is exhausted, which occurs in register R_{FINAL}. Register RA is not altered (unless RA = R_{FINAL}, an invalid form of this instruction). Bytes which would have been loaded into register RA are discarded.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT and subsequent GPRs as described above.

Invalid Instruction Forms

- Reserved fields
- RA is in the range of registers to be loaded
- RA = RT = 0

Exceptions

If MSR[LE] = 1, an alignment exception occurs.

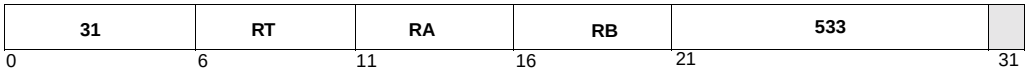
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lswx

Load String Lword Indexed

lswx **RT, RA, RB**




```
EA ← (RA|0) + (RB)
CNT ← XER[TBC]
n ← CNT
R_FINAL ← ((RT + CEIL(CNT/4) – 1) % 32)
r ← RT – 1
i ← 0
do while n > 0
   if i = 0 then
      r ← r + 1
      if r = 32 then
         r ← 0
      if (((r ≠ RA) ∧ (r ≠ RB)) ∨ (r = R_FINAL)) then
         (GPR(r)) ← 0
   if (((r ≠ RA) ∧ (r ≠ RB)) ∨ (r = R_FINAL)) then
      (GPR(r)_i:i+7) ← MS(EA,1)
   i ← i + 8
   if i = 32 then
      i ← 0
   EA ← EA + 1
   n ← n – 1
```

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

A byte count CNT is obtained from XER[TBC].

A series of CNT consecutive bytes in main storage, starting at the EA, are loaded into CEIL(CNT/4) consecutive GPRs, four bytes per GPR, until the byte count is exhausted. Bytes are loaded into GPRs; the byte having the lowest address is loaded into the most significant byte. Bits to the right of the last byte loaded in the last GPR used are set to 0.

The set of consecutive GPRs loaded starts at register RT, continues through GPR(31), and wraps to register 0, loading until the byte count is exhausted, which occurs in register R_{FINAL}. Register RA is not altered (unless RA = R_{FINAL}, which is an invalid form of this instruction). Register RB is not altered (unless RB = R_{FINAL}, which is an invalid form of this instruction). Bytes which would have been loaded into registers RA or RB are discarded.

If XER[TBC] is 0, the byte count is 0 and the contents of register RT are undefined.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT and subsequent GPRs as described above.

Invalid Instruction Forms

- Reserved fields
- RA or RB is in the range of registers to be loaded.
- RA = RT = 0

Programming Note

If XER[TBC] = 0, the contents of register RT are unchanged and **lswx** is treated as a no-op.

The PowerPC Architecture states that, if XER[TBC] = 0 and if the EA is such that a precise data exception would normally occur (if not for the zero length), **lswx** is treated as a no-op and the precise exception does not occur. Data storage exceptions and alignment exceptions are examples of precise data exceptions.

However, the PowerPC Architecture makes no statement regarding imprecise exceptions related to **lswx** with XER[TBC] = 0. The IOP 480 CPU generates an imprecise exception (machine check) on this instruction when all of the following conditions are true:

- The instruction passes all protection bounds checking
- The address is cacheable
- The address is passed to the data cache
- The address misses in the data cache (resulting in a line fill request)
- The address encounters some form of bus error

Exceptions

If MSR[LE] = 1, an alignment exception occurs.

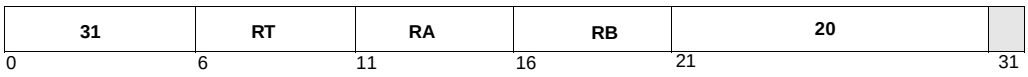
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwarx

Load Lword and Reserve Indexed

lwarx **RT, RA, RB**



EA ← (RA|0) + (RB)
RESERVE ← 1
(RT) ← MS(EA,4)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The Lword at the EA is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Execution of the **lwarx** instruction sets the reservation bit.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Programming Note

lwarx and the **stwcx.** instruction should paired in a loop, as shown in the following example, to create the effect of an atomic operation to a memory area used as a semaphore between asynchronous processes. Only **lwarx** can set the reservation bit to 1. **stwcx.** sets the reservation bit to 0 upon its completion, whether or not **stwcx.** sent (RS) to memory. $CR[CR0]_{EQ}$ must be examined to determine whether (RS) was sent to memory.

```
loop: lwarx                    # read the semaphore from memory; set reservation
“alter”   # change the semaphore bits in register as required
stwcx.   # attempt to store semaphore; reset reservation
bne loop# an asynchronous process has intervened; try again
```

If the asynchronous process in the code example had paired **lwarx** with a store other than **stwcx.**, the reservation bit would not have been cleared in the asynchronous process, and the code example would have overwritten the semaphore.

Exceptions

An alignment exception occurs if the EA is not Lword-aligned.

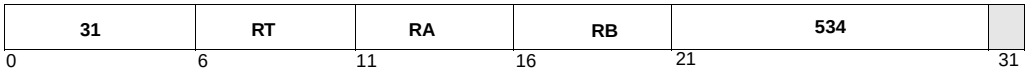
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwbrx

Load Lword Byte-Reverse Indexed

lwbrx **RT, RA, RB**



EA ← (RA|0) + (RB)
(RT) ← MS(EA+3,1) || MS(EA+2,1) || MS(EA+1,1) || MS(EA,1)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The Lword at the EA is byte-reversed: the least significant byte becomes the most significant byte, the next least significant byte becomes the next most significant byte, and so on. The resulting Lword is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

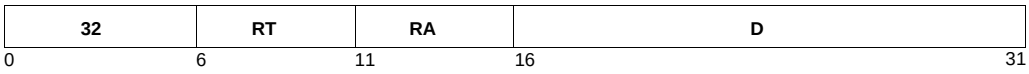
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwz

Load Lword and Zero

lwz **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RT) ← MS(EA,4)

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The Lword at the EA is placed into register RT.

Registers Altered

- RT

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

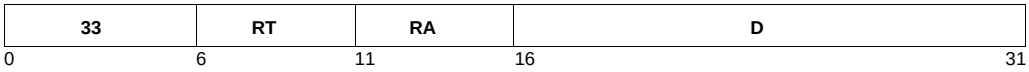
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzu

Load Lword and Zero with Update

lwzu **RT, D(RA)**



EA ← (RA|0) + EXTS(D)
(RA) ← EA
(RT) ← MS(EA,4)

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The Lword at the EA is placed into register RT.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- RA = RT
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

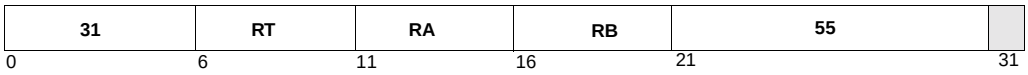
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzux

Load Lword and Zero with Update Indexed

lwzux **RT, RA, RB**



EA ← (RA|0) + (RB)
(RA) ← EA
(RT) ← MS(EA,4)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise. The EA is placed into register RA.

The Lword at the EA is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA
- RT

Invalid Instruction Forms

- Reserved fields
- RA = RT
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

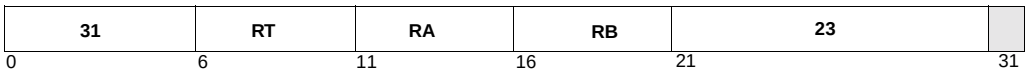
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzx

Load Lword and Zero Indexed

lwzx **RT, RA, RB**



EA ← (RA|0) + (RB)
(RT) ← MS(EA,4)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The Lword at the EA is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

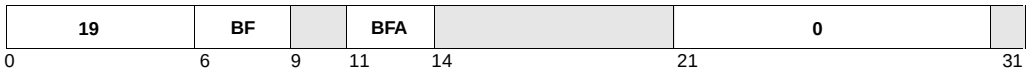
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mcrf

Move Condition Register Field

mcrf **BF, BFA**



m ← BFA
n ← BF
(CR[CRn]) ← (CR[CRm])

The contents of the CR field specified by the BFA field are placed into the CR field specified by the BF field.

Registers Altered

- CR[CRn] where n is specified by the BF field.

Invalid Instruction Forms

- Reserved fields

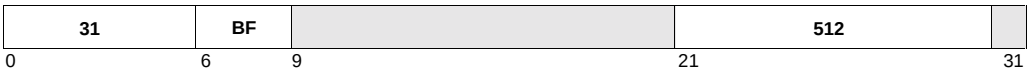
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mcrxr

Move to Condition Register from XER

mcrxr **BF**



n ← BF
(CR[CRn]) ← $XER_{0:3}$
$XER_{0:3}$ ← $^{4}0$

The contents of $XER_{0:3}$ are placed into the CR field specified by the BF field. $XER_{0:3}$ are then set to 0.

This transfer is positional, by bit number, so the mnemonics associated with each bit are changed. See the following table for clarification.

Bit	XER Usage	CR Usage
0	SO	LT
1	OV	GT
2	CA	EQ
3	Reserved	SO

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- CR[CRn] where n is specified by the BF field.
- XER[SO, OV, CA]

Invalid Instruction Forms

- Reserved fields

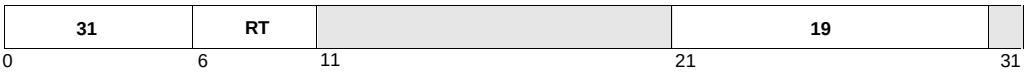
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mfcr

Move From Condition Register

mfcr **RT**



(RT) ← (CR)

The contents of the CR are placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

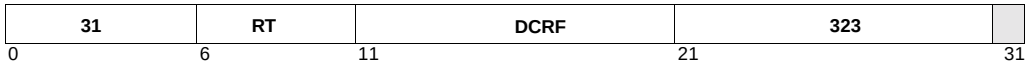
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mfdcr

Move from Device Control Register

mfdcr **RT, DCRN**



DCRN ← $DCRF_{5:9}$ || $DCRF_{0:4}$
(RT) ← (DCR(DCRN))

The contents of the DCR specified by the DCRF field are placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields
- Invalid DCRF values

Programming Note

Execution of this instruction is privileged.

The DCR number (DCRN) specified in the assembler language coding of the **mfdcr** instruction refers to a DCR number. The assembler handles the unusual register number encoding to generate the DCRF field.

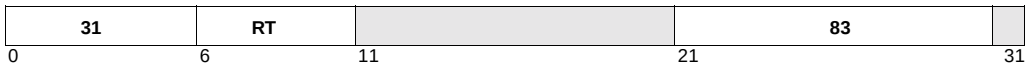
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

mfmsr

Move From Machine State Register

mfmsr **RT**



(RT) ← (MSR)

The contents of the MSR are placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

mfspr

Move From Special Purpose Register

mfspr RT, SPRN

$SPRN \leftarrow SPRF_{5:9} \parallel SPRF_{0:4}$
$(RT) \leftarrow (SPR(SPRN))$

The contents of the SPR specified by the SPRF field are placed into register RT. See Table 29-2, “Special Purpose Registers,” on page 29-2 for a listing of SPR mnemonics and corresponding SPRN and SPRF values.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields
- Invalid SPRF values

Programming Note

Execution of this instruction is privileged if instruction bit 11 contains 1. See Section 24.8.3, “Privileged SPRs,” on page 24-28, for more information.

The SPR number (SPRN) specified in the assembler language coding of the **mfspr** instruction refers to an SPR number (see Table 29-2, “Special Purpose Registers,” on page 29-2 for a list of SPRN values). The assembler handles the unusual register number encoding to generate the SPRF field. Also, see Section 24.8.3, “Privileged SPRs,” on page 24-28 for information about privileged SPRs.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

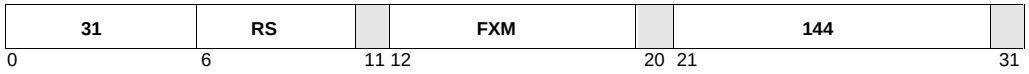
Table 28-21. Extended Mnemonics for mfspr

Mnemonic	Operands	Function	Other Registers Changed
mfcdbcr **mfctr** **mfdac1** **mfdbcr** **mfdbsr** **mfdbsrs** **mfdccr** **mfdcwr** **mfdear** **mfesr** **mfevpr** **mfiac1** **mficcr** **mficdbdr** **mflr** **mfpit** **mfpvr** **mfsgr** **mfsler** **mfsprg0** **mfsprg1** **mfsprg2** **mfsprg3** **mfsrr0** **mfsrr1** **mfsrr2** **mfsrr3** **mftbhi** **mftbhu** **mftblo** **mftblu** **mftcr** **mftsr** **mfxer**	RT	Move from special purpose register SPRN. Extended mnemonic for mfspr RT,SPRN See Table 29-2, “Special Purpose Registers,” on page 29-2 for a list of valid SPRN values.	

mtcrf

Move To Condition Register Fields

mtcrf FXM, RS



mask $\leftarrow$ $^4(FXM_0)$ || $^4(FXM_1)$ || ... || $^4(FXM_6)$ || $^4(FXM_7)$
(CR) $\leftarrow$ ((RS) $\wedge$ mask) $\vee$ ((CR) $\wedge$ $\neg$mask)

Some or all of the contents of register RS are placed into the CR as specified by the FXM field.

Each bit in the FXM field controls the copying of 4 bits in register RS into the corresponding bits in the CR. The correspondence between the bits in the FXM field and the bit copying operation is shown in the following table:

FXM Bit Number	Bits Controlled
0	0:3
1	4:7
2	8:11
3	12:15
4	16:19
5	20:23
6	24:27
7	28:31

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- CR

Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

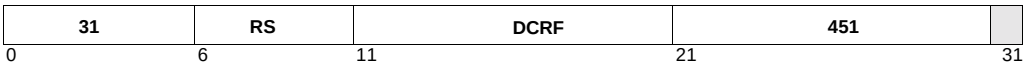
Table 28-22. Extended Mnemonics for mtcrf

Mnemonic	Operands	Function	Other Registers Changed
mtcr	RS	Move to Condition Register. Extended mnemonic for mtcrf 0xFF,RS	

mtdcr

Move To Device Control Register

mtdcr **DCRN, RS**



DCRN ← $DCRF_{5:9}$ || $DCRF_{0:4}$
(DCR(DCRN)) ← (RS)

The contents of register RS are placed into the DCR specified by the DCRF field.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- DCR(DCRN)

Invalid Instruction Forms

- Reserved fields
- Invalid DCRF values

Programming Note

Execution of this instruction is privileged.

The DCR number (DCRN) specified in the assembler language coding of the **mtdcr** instruction refers to a DCR number. The assembler handles the unusual register number encoding to generate the DCRF field.

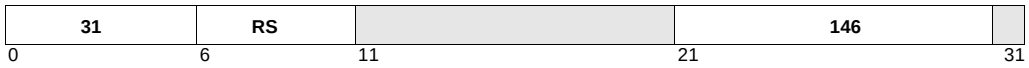
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

mtmsr

Move To Machine State Register

mtmsr **RS**



(MSR) ← (RS)

The contents of register RS are placed into the MSR.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- MSR

Invalid Instruction Forms

- Reserved fields

Programming Note

The **mtmsr** instruction is privileged and execution synchronizing.

Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

mtspr

Move To Special Purpose Register

mtspr SPRN, RS

$SPRN \leftarrow SPRF_{5:9} \parallel SPRF_{0:4}$
$(SPR(SPRN)) \leftarrow (RS)$

The contents of register RS are placed into register RT. See Table 29-2, “Special Purpose Registers,” on page 29-2 for a listing of SPR mnemonics and corresponding SPRN and SPRF values.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- SPR(SPRN)

Invalid Instruction Forms

- Reserved fields
- Invalid SPRF values

Programming Note

Execution of this instruction is privileged if instruction bit 11 is set to 1. See Section 24.8.3, “Privileged SPRs,” on page 24-28, for more information.

The SPR number (SPRN) specified in the assembler language coding of the **mtspr** instruction refers to an SPR number (see Table 29-2, “Special Purpose Registers,” on page 29-2 for a list of SPRN values). The assembler handles the unusual register number encoding to generate the SPRF field. Also, see Section 24.8.3, “Privileged SPRs,” on page 24-28 for information about privileged SPRs.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

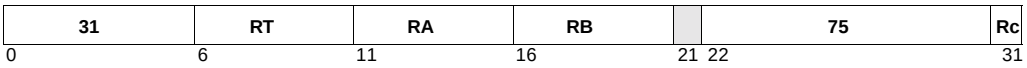
Table 28-23. Extended Mnemonics for mtspr

Mnemonic	Operands	Function	Other Registers Changed
mtcdbcr **mtctr** **mtdac1** **mtdbcr** **mtdbsr** **mtdccr** **mtdcwr** **mtdear** **mtesr** **mtevpr** **mtiac1** **mticcr** **mticdbdr** **mtlr** **mtpit** **mtpvr** **mtsgr** **mtsler** **mttsprg0** **mtsprg1** **mtsprg2** **mtsprg3** **mtsrr0** **mtsrr1** **mtsrr2** **mtsrr3** **mttbhi** **mttblo** **mttcr** **mttsr** **mtxer**	RS	Move to special purpose register SPRN. Extended mnemonic for mtspr SPRN,RS See Table 29-2, “Special Purpose Registers,” on page 29-2 for a list of valid SPRN values.	

mulhw

Multiply High Lword

mulhw	**RT, RA, RB**	Rc=0
mulhw.	**RT, RA, RB**	Rc=1



$prod_{0:63} \leftarrow (RA) \times (RB)$ (signed)
$(RT) \leftarrow prod_{0:31}$

The 64-bit signed product of registers RA and RB is formed. The most significant 32 bits of the result is placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Programming Note

The most significant 32 bits of the product, unlike the least significant 32 bits, may differ depending on whether the registers RA and RB are interpreted as signed or unsigned quantities. The **mulhw** instruction generates the correct result when these operands are interpreted as signed quantities. The **mulhwu** instruction generates the correct result when these operands are interpreted as unsigned quantities.

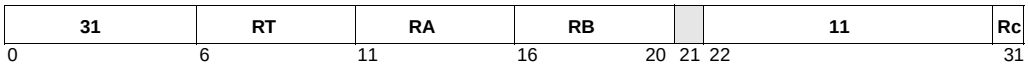
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mulhwu

Multiply High Lword Unsigned

mulhwu	**RT, RA, RB**	Rc=0
mulhwu.	**RT, RA, RB**	Rc=1



$prod_{0:63} \leftarrow (RA) \times (RB)$ (unsigned)
$(RT) \leftarrow prod_{0:31}$

The 64-bit unsigned product of registers RA and RB is formed. The most significant 32 bits of the result are placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Programming Note

The most significant 32 bits of the product, unlike the least significant 32 bits, may differ depending on whether the registers RA and RB are interpreted as signed or unsigned quantities. The **mulhw** instruction generates the correct result when these operands are interpreted as signed quantities. The **mulhwu** instruction generates the correct result when these operands are interpreted as unsigned quantities.

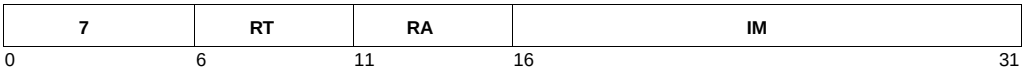
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mulli

Multiply Low Immediate

mulli **RT, RA, IM**



$prod_{0:47} \leftarrow (RA) \times EXTS(IM)$ (signed)
$(RT) \leftarrow prod_{16:47}$

The 48-bit product of register RA and the sign-extended IM field is formed. Both register RA and the IM field are interpreted as signed quantities. The least significant 32 bits of the product are placed into register RT.

Registers Altered

- RT

Programming Note

The least significant 32 bits of the product are correct, regardless of whether register RA and field IM are interpreted as signed or unsigned numbers.

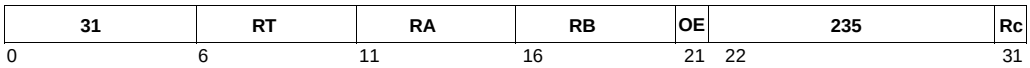
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mullw

Multiply Low Lword

mullw	**RT, RA, RB**	OE=0, Rc=0
mullw.	**RT, RA, RB**	OE=0, Rc=1
mullwo	**RT, RA, RB**	OE=1, Rc=0
mullwo.	**RT, RA, RB**	OE=1, Rc=1



$prod_{0:63} \leftarrow$ (RA) $\times$ (RB) (signed)
(RT) $\leftarrow$ $prod_{32:63}$

The 64-bit signed product of register RA and register RB is formed. The least significant 32 bits of the result is placed into register RT.

If the signed product cannot be represented in 32 bits and OE=1, XER[SO, OV] are set to 1.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE=1

Programming Note

The least significant 32 bits of the product are correct, regardless of whether register RA and register RB are interpreted as signed or unsigned numbers. The overflow indication is correct only if the operands are regarded as signed numbers.

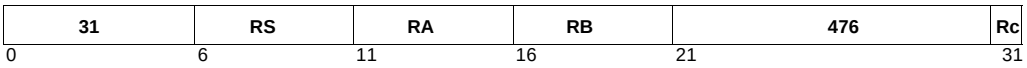
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

nand

NAND

nand	**RA, RS, RB**	Rc=0
nand.	**RA, RS, RB**	Rc=1



$(RA) \leftarrow \neg((RS) \wedge (RB))$

The contents of register RS is ANDed with the contents of register RB; the ones complement of the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

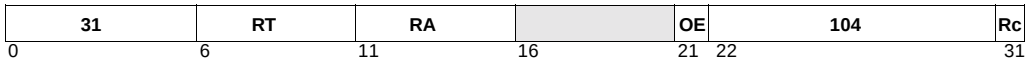
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

neg

Negate

neg	**RT, RA**	OE=0, Rc=0
neg.	**RT, RA**	OE=0, Rc=1
nego	**RT, RA**	OE=1, Rc=0
nego.	**RT, RA**	OE=1, Rc=1



(RT) ← ¬(RA) + 1

The *twos* complement of the contents of register RA are placed into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[CA, SO, OV] if OE=1

Invalid Instruction Forms

- Reserved fields

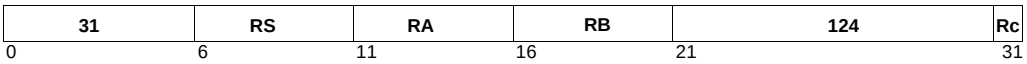
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

nor

NOR

nor	**RA, RS, RB**	Rc=0
nor.	**RA, RS, RB**	Rc=1



(RA) ← ¬((RS) ∨ (RB))

The contents of register RS is ORed with the contents of register RB; the ones complement of the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

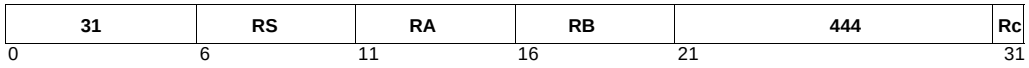
Table 28-24. Extended Mnemonics for nor, nor.

Mnemonic	Operands	Function	Other Registers Changed
not	RA, RS	Complement register. (RA) ← ¬(RS) Extended mnemonic for nor RA,RS,RS	
not.		Extended mnemonic for nor. RA,RS,RS	CR[CR0]

or

OR

or	**RA, RS, RB**	Rc=0
or.	**RA, RS, RB**	Rc=1



(RA) ← (RS) ∨ (RB)

The contents of register RS is ORed with the contents of register RB; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

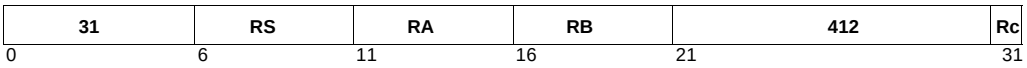
Table 28-25. Extended Mnemonics for or, or.

Mnemonic	Operands	Function	Other Registers Changed
mr	RT, RS	Move register. (RT) ← (RS) Extended mnemonic for or RT,RS,RS	
mr.		Extended mnemonic for or. RT,RS,RS	CR[CR0]

orc

OR with Complement

orc	**RA, RS, RB**	Rc=0
orc.	**RA, RS, RB**	Rc=1



(RA) $\leftarrow$ (RS) $\vee$ $\neg$(RB)

The contents of register RS is ORed with the ones complement of the contents of register RB; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

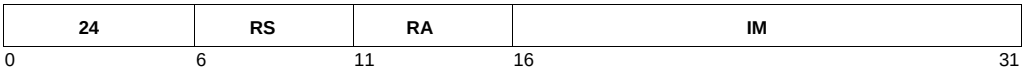
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

ori

OR Immediate

ori **RA, RS, IM**



(RA) $\leftarrow$ (RS) $\vee$ ($^{16}0$ || IM)

The IM field is extended to 32 bits by concatenating 16 0-bits on the left. Register RS is ORed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

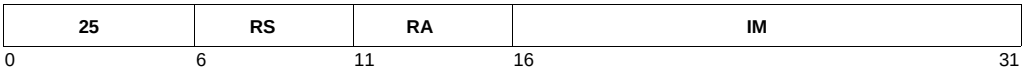
Table 28-26. Extended Mnemonics for ori

Mnemonic	Operands	Function	Other Registers Changed
nop		Preferred no-op; triggers optimizations based on no-ops. Extended mnemonic for ori 0,0,0	

oris

OR Immediate Shifted

oris **RA, RS, IM**



(RA) $\leftarrow$ (RS) $\vee$ (IM $\|$ $^{16}0$)

The IM Field is extended to 32 bits by concatenating 16 0-bits on the right. Register RS is ORed with the extended IM field and the result is placed into register RA.

Registers Altered

- RA

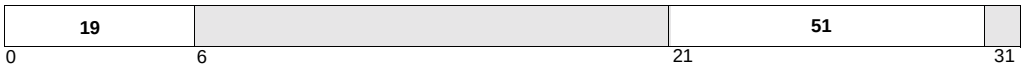
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

rfci

Return From Critical Interrupt

rfci



(PC) ← (SRR2)
(MSR) ← (SRR3)

The program counter (PC) is restored with the contents of SRR2 and the MSR is restored with the contents of SRR3.

Instruction execution returns to the address contained in the PC.

Registers Altered

- MSR

Programming Note

Execution of this instruction is privileged and context-synchronizing.

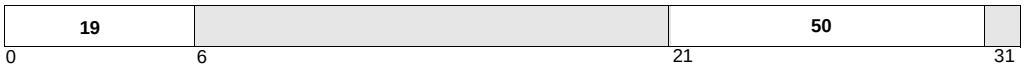
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

rfi

Return From Interrupt

rfi



(PC) ← (SRR0)
(MSR) ← (SRR1)

The program counter (PC) is restored with the contents of SRR0 and the MSR is restored with the contents of SRR1.

Instruction execution returns to the address contained in the PC.

Registers Altered

- MSR

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged and context-synchronizing.

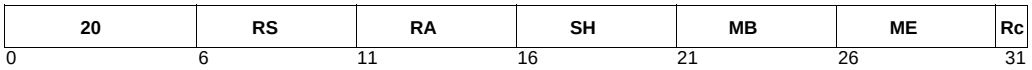
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

rlwimi

Rotate Left Lword Immediate then Mask Insert

rlwimi **RA, RS, SH, MB, ME** Rc=0
rlwimi. **RA, RS, SH, MB, ME** Rc=1



r ← ROTL((RS), SH)
m ← MASK(MB, ME)
(RA) ← (r ∧ m) ∨ ((RA) ∧ ¬m)

The contents of register RS are rotated left by the number of bit positions specified in the SH field. A mask is generated, having 1-bits starting at the bit position specified in the MB field and ending in the bit position specified by the ME field, with 0-bits elsewhere.

If the starting point of the mask is at a higher bit position than the ending point, the 1-bits portion of the mask wraps from the highest bit position back around to the lowest. The rotated data is inserted into register RA, in positions corresponding to the bit positions in the mask that contain a 1-bit.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

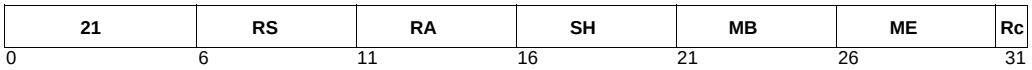
Table 28-27. Extended Mnemonics for rlwimi, rlwimi.

Mnemonic	Operands	Function	Other Registers Changed
inslwi	RA, RS, n, b	Insert from left immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{0:n-1}$ Extended mnemonic for rlwimi RA,RS,32–b,b,b+n–1	
inslwi.		Extended mnemonic for rlwimi. RA,RS,32–b,b,b+n–1	CR[CR0]
insrwi	RA, RS, n, b	Insert from right immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{32-n:31}$ Extended mnemonic for rlwimi RA,RS,32–b–n,b,b+n–1	
insrwi.		Extended mnemonic for rlwimi. RA,RS,32–b–n,b,b+n–1	CR[CR0]

rlwinm

Rotate Left Lword Immediate then AND with Mask

rlwinm **RA, RS, SH, MB, ME** Rc=0
rlwinm. **RA, RS, SH, MB, ME** Rc=1



r ← ROTL((RS), SH)
m ← MASK(MB, ME)
(RA) ← r ∧ m

The contents of register RS are rotated left by the number of bit positions specified in the SH field. A mask is generated, having 1-bits starting at the bit position specified in the MB field and ending in the bit position specified by the ME field with 0-bits elsewhere.

If the starting point of the mask is at a higher bit position than the ending point, the 1-bits portion of the mask wraps from the highest bit position back around to the lowest. The rotated data is ANDed with the generated mask; the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 28-28. Extended Mnemonics for rlwinm, rlwinm.

Mnemonic	Operands	Function	Other Registers Changed
clrlwi	RA, RS, n	Clear left immediate. (n < 32) $(RA)_{0:n-1} \leftarrow {}^{n}0$ Extended mnemonic for rlwinm RA,RS,0,n,31	
clrlwi.		Extended mnemonic for rlwinm. RA,RS,0,n,31	CR[CR0]
clrlslwi	RA, RS, b, n	Clear left and shift left immediate. (n ≤ b < 32) $(RA)_{b-n:31-n} \leftarrow (RS)_{b:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ $(RA)_{0:b-n-1} \leftarrow {}^{b-n}0$ Extended mnemonic for rlwinm RA,RS,n,b–n,31–n	
clrlslwi.		Extended mnemonic for rlwinm. RA,RS,n,b–n,31–n	CR[CR0]
clrrwi	RA, RS, n	Clear right immediate. (n < 32) $(RA)_{32-n:31} \leftarrow {}^{n}0$ *Extended mnemonic* for rlwinm RA,RS,0,0,31–n	
clrrwi.		Extended mnemonic for rlwinm. RA,RS,0,0,31–n	CR[CR0]

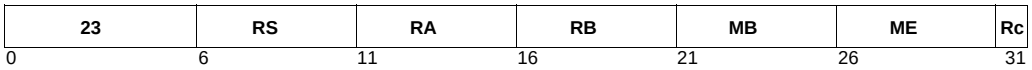
Table 28-28. Extended Mnemonics for rlwinm, rlwinm.

Mnemonic	Operands	Function	Other Registers Changed
extlwi	RA, RS, n, b	Extract and left justify immediate. (n > 0) $(RA)_{0:n-1} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{n:31} \leftarrow {}^{32-n}0$ Extended mnemonic for rlwinm RA,RS,b,0,n–1	
extlwi.		Extended mnemonic for rlwinm. RA,RS,b,0,n–1	CR[CR0]
extrwi	RA, RS, n, b	Extract and right justify immediate. (n > 0) $(RA)_{32-n:31} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{0:31-n} \leftarrow {}^{32-n}0$ Extended mnemonic for rlwinm RA,RS,b+n,32–n,31	
extrwi.		Extended mnemonic for rlwinm. RA,RS,b+n,32–n,31	CR[CR0]
rotlwi	RA, RS, n	Rotate left immediate. (RA) $\leftarrow$ ROTL((RS), n) Extended mnemonic for rlwinm RA,RS,n,0,31	
rotlwi.		Extended mnemonic for rlwinm. RA,RS,n,0,31	CR[CR0]
rotrwi	RA, RS, n	Rotate right immediate. (RA) $\leftarrow$ ROTL((RS), 32–n) Extended mnemonic for rlwinm RA,RS,32–n,0,31	
rotrwi.		Extended mnemonic for rlwinm. RA,RS,32–n,0,31	CR[CR0]
slwi	RA, RS, n	Shift left immediate. (n < 32) $(RA)_{0:31-n} \leftarrow (RS)_{n:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ Extended mnemonic for rlwinm RA,RS,n,0,31–n	
slwi.		Extended mnemonic for rlwinm. RA,RS,n,0,31–n	CR[CR0]
srwi	RA, RS, n	Shift right immediate. (n < 32) $(RA)_{n:31} \leftarrow (RS)_{0:31-n}$ $(RA)_{0:n-1} \leftarrow {}^{n}0$ Extended mnemonic for rlwinm RA,RS,32–n,n,31	
srwi.		Extended mnemonic for rlwinm. RA,RS,32–n,n,31	CR[CR0]

rlwnm

Rotate Left Lword then AND with Mask

rlwnm	**RA, RS, RB, MB, ME**	Rc=0
rlwnm.	**RA, RS, RB, MB, ME**	Rc=1



r ← ROTL((RS), $(RB)_{27:31}$)
m ← MASK(MB, ME)
(RA) ← r ∧ m

The contents of register RS are rotated left by the number of bit positions specified by the contents of register $RB_{27:31}$. A mask is generated, having 1-bits starting at the bit position specified in the MB field and ending in the bit position specified by the ME field with 0-bits elsewhere.

If the starting point of the mask is at a higher bit position than the ending point, the ones portion of the mask wraps from the highest bit position back to the lowest. The rotated data is ANDed with the generated mask and the result is placed into register RA.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

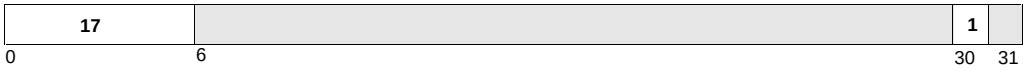
Table 28-29. Extended Mnemonics for rlwnm, rlwnm.

Mnemonic	Operands	Function	Other Registers Changed
rotlw	RA, RS, RB	Rotate left. (RA) ← ROTL((RS), $(RB)_{27:31}$) Extended mnemonic for rlwnm RA,RS,RB,0,31	
rotlw.		Extended mnemonic for rlwnm. RA,RS,RB,0,31	CR[CR0]

sc

System Call

sc



(SRR1) ← (MSR)
(SRR0) ← (PC)
PC ← $EVPR_{0:15}$ || 0x0C00
(MSR[WE, EE, PR, DR, IR]) ← 0
(MSR[LE]) ← (MSR[ILE])

A system call exception is generated. The contents of the MSR are copied into SRR1 and (4 + address of **sc** instruction) is placed into SRR0.

The program counter (PC) is then loaded with the exception vector address. The exception vector address is calculated by concatenating the high word of the Exception Vector Prefix Register (EVPR) to the left of 0x0C00.

The MSR[WE, EE, PR, DR, IR] bits are set to 0, and MSR[ILE] is copied to MSR[LE].

Program execution continues at the new address in the PC.

The **sc** instruction is context synchronizing.

Registers Altered

- SRR0
- SRR1
- MSR[WE, EE, PR, DR, IR, LE]

Invalid Instruction Forms

- Reserved fields

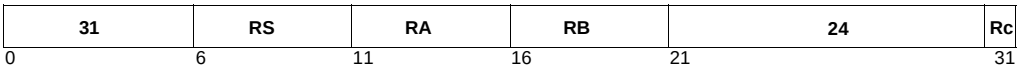
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

slw

Shift Left Lword

slw	**RA, RS, RB**	Rc=0
slw.	**RA, RS, RB**	Rc=1



$n \leftarrow (RB)_{27:31}$
$r \leftarrow ROTL((RS), n)$
if $(RB)_{26} = 0$ then
    $m \leftarrow MASK(0, 31 - n)$
else
    $m \leftarrow {}^{32}0$
$(RA) \leftarrow r \wedge m$

The contents of register RS are shifted left by the number of bits specified by the contents of register $RB_{27:31}$. Bits shifted left out of the most significant bit are lost, and 0-bits fill vacated bit positions on the right. The result is placed into register RA.

If bit 26 of register RB contains a one, register RA is set to zero.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

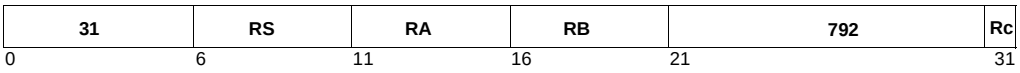
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sraw

Shift Right Algebraic Lword

sraw	**RA, RS, RB**	Rc=0
sraw.	**RA, RS, RB**	Rc=1



$n \leftarrow (RB)_{27:31}$
$r \leftarrow ROTL((RS), 32 - n)$
if $(RB)_{26} = 0$ then
  $m \leftarrow MASK(n, 31)$
else
  $m \leftarrow {}^{32}0$
$s \leftarrow (RS)_0$
$(RA) \leftarrow (r \wedge m) \vee ({}^{32}s \wedge \neg m)$
$XER[CA] \leftarrow s \wedge ((r \wedge \neg m) \neq 0)$

The contents of register RS are shifted right by the number of bits specified the contents of register $RB_{27:31}$. Bits shifted out of the least significant bit are lost. Register RS_0 is replicated to fill the vacated positions on the left. The result is placed into register RA.

If register RS contains a negative number and any 1-bits were shifted out of the least significant bit position, XER[CA] is set to 1; otherwise, it is set to 0.

If bit 26 of register RB contains 1, register RA and XER[CA] are set to bit 0 of register RS.

Registers Altered

- RA
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

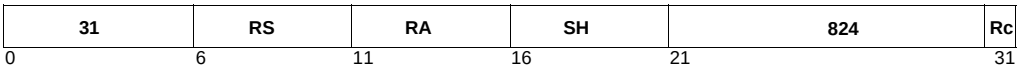
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

srawi

Shift Right Algebraic Lword Immediate

srawi	**RA, RS, SH**	Rc=0
srawi.	**RA, RS, SH**	Rc=1



$n \leftarrow SH$
$r \leftarrow ROTL((RS), 32 - n)$
$m \leftarrow MASK(n, 31)$
$s \leftarrow (RS)_0$
$(RA) \leftarrow (r \wedge m) \vee ({}^{32}s \wedge \neg m)$
$XER[CA] \leftarrow s \wedge ((r \wedge \neg m) \neq 0)$

The contents of register RS are shifted right by the number of bits specified in the SH field. Bits shifted out of the least significant bit are lost. Bit RS_0 is replicated to fill the vacated positions on the left. The result is placed into register RA.

If register RS contains a negative number and any 1-bits were shifted out of the least significant bit position, XER[CA] is set to 1; otherwise, it is set to 0.

Registers Altered

- RA
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

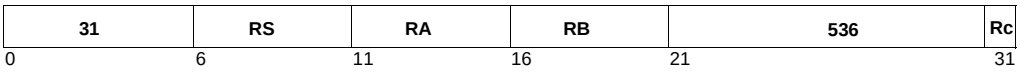
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

srw

Shift Right Lword

srw	**RA, RS, RB**	Rc=0
srw.	**RA, RS, RB**	Rc=1



$n \leftarrow (RB)_{27:31}$
$r \leftarrow ROTL((RS), 32 - n)$
if $(RB)_{26} = 0$ then
  $m \leftarrow MASK(n, 31)$
else
  $m \leftarrow {}^{32}0$
$(RA) \leftarrow r \wedge m$

The contents of register RS are shifted right by the number of bits specified the contents of register $RB_{27:31}$. Bits shifted right out of the least significant bit are lost, and 0-bits fill the vacated bit positions on the left. The result is placed into register RA.

If bit 26 of register RB contains a one, register RA is set to 0.

Registers Altered

- RA
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

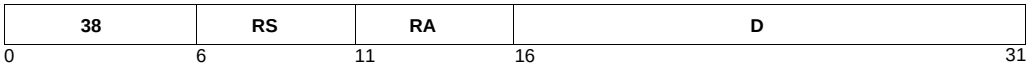
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stb

Store Byte

stb **RS, D(RA)**



EA ← (RA|0) + EXTS(D)
MS(EA, 1) ← $(RS)_{24:31}$

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the EA.

Registers Altered

- None

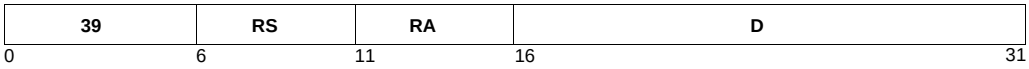
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbu

Store Byte with Update

stbu RS, D(RA)



EA ← (RA|0) + EXTS(D)
MS(EA, 1) ← $(RS)_{24:31}$
(RA) ← EA

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the EA.

The EA is placed into register RA.

Registers Altered

- RA

Invalid Instruction Forms

RA = 0

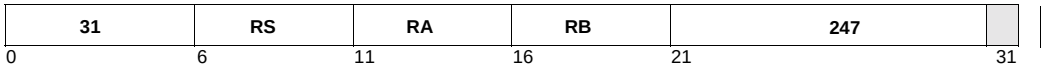
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbux

Store Byte with Update Indexed

stbux **RS, RA, RB**



EA ← (RA|0) + (RB)
MS(EA, 1) ← $(RS)_{24:31}$
(RA) ← EA

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the EA.

The EA is placed into register RA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA

Invalid Instruction Forms

- Reserved fields
- RA = 0

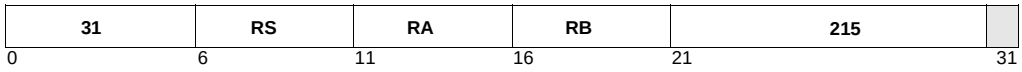
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbx

Store Byte Indexed

stbx **RS, RA, RB**



EA $\leftarrow$ (RA|0) + (RB)
MS(EA, 1) $\leftarrow$ $(RS)_{24:31}$

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant byte of register RS is stored into the byte at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

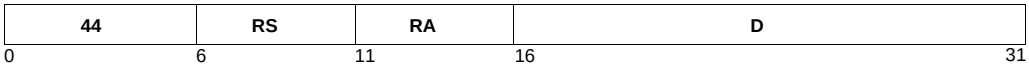
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sth

Store Word

sth **RS, D(RA)**



EA ← (RA|0) + EXTS(D)
MS(EA, 2) ← $(RS)_{16:31}$

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0 and is the contents of register RA otherwise.

The least significant word of register RS is stored into the word at the EA in main storage.

Registers Altered

- None

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

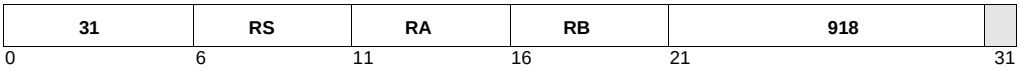
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthbrx

Store Word Byte-Reverse Indexed

sthbrx RS, RA, RB



EA $\leftarrow$ (RA|0) + (RB)
MS(EA, 2) $\leftarrow$ $(RS)_{24:31}$ || $(RS)_{16:23}$

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant word of register RS is byte-reversed. The result is stored into the word at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

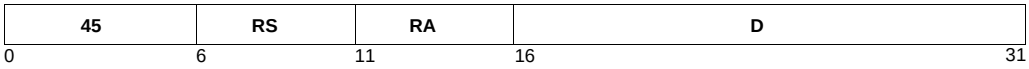
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthu

Store Word with Update

sthu **RS, D(RA)**



EA ← (RA|0) + EXTS(D)
MS(EA, 2) ← $(RS)_{16:31}$
(RA) ← EA

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant word of register RS is stored into the word at the EA.

The EA is placed into register RA.

Registers Altered

- RA

Invalid Instruction Forms

- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

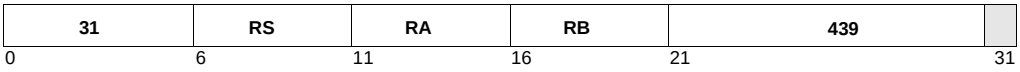
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthux

Store Word with Update Indexed

sthux **RS, RA, RB**



EA ← (RA|0) + (RB)
MS(EA, 2) ← $(RS)_{16:31}$
(RA) ← EA

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant word of register RS is stored into the word at the EA.

The EA is placed into register RA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA

Invalid Instruction Forms

- Reserved fields
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

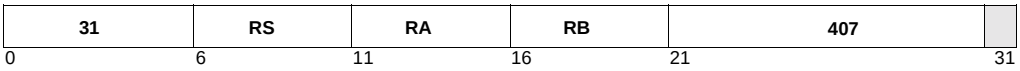
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthx

Store Word Indexed

sthx **RS, RA, RB**



EA $\leftarrow$ (RA|0) + (RB)
MS(EA, 2) $\leftarrow$ $(RS)_{16:31}$

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The least significant word of register RS is stored into the word at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not word-aligned.

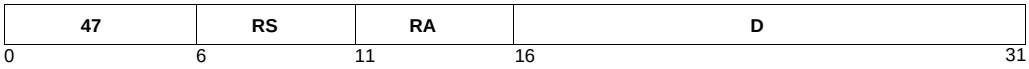
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stmw

Store Multiple Lword

stmw RS, D(RA)




```
EA ← (RA|0) + EXTS(D)
r ← RS
do while r ≤ 31
   MS(EA, 4) ← (GPR(r))
   r ← r + 1
   EA ← EA + 4
```

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The contents of a series of consecutive registers, starting with register RS and continuing through GPR(31), are stored into consecutive words starting at the EA.

Registers Altered

- None

Exceptions

An alignment exception occurs if MSR[LE] = 1.

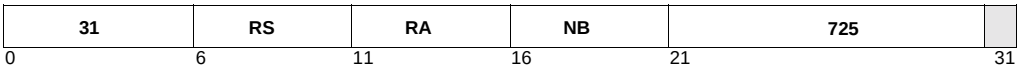
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stswi

Store String Lword Immediate

stswi **RS, RA, NB**




```
EA ← (RA|0)
if NB = 0 then
   n ← 32
else
   n ← NB
r ← RS – 1
i ← 0
do while n > 0
   if i = 0 then
      r ← r + 1
   if r = 32 then
      r ← 0
   MS(EA,1) ← (GPR(r)i:i+7)
   i ← i + 8
   if i = 32 then
      i ← 0
   EA ← EA + 1
   n ← n – 1
```

An effective address (EA) is determined by the RA field. If the RA field contains 0, the EA is 0; otherwise, the EA is the contents of register RA.

A byte count is determined by the NB field. If the NB field contains 0, the byte count is 32; otherwise, the byte count is the contents of the NB field.

The contents of a series of consecutive GPRs (starting with register RS, continuing through GPR(31), wrapping to GPR(0), and continuing to the final byte count) are stored, starting at the EA. The bytes in each GPR are accessed starting with the most significant byte. The byte count determines the number of transferred bytes.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Exceptions

An alignment exception occurs if MSR[LE] = 1.

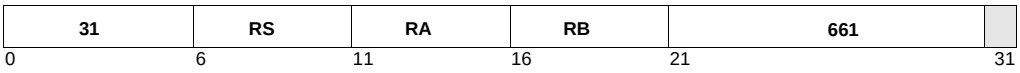
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stswx

Store String Lword Indexed

stswx RS, RA, RB




```
EA ← (RA|0) + (RB)
n ← XER[TBC]
r ← RS – 1
i ← 0
do while n > 0
    if i = 0 then
        r ← r + 1
    if r = 32 then
        r ← 0
    MS(EA, 1) ← (GPR(r)i:i+7)
    i ← i + 8
    if i = 32 then
        i ← 0
    EA ← EA + 1
    n ← n – 1
```

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

A byte count is contained in XER[TBC].

The contents of a series of consecutive GPRs (starting with register RS, continuing through GPR(31), wrapping to GPR(0), and continuing to the final byte count) are stored, starting at the EA. The bytes in each GPR are accessed starting with the most significant byte. The byte count determines the number of transferred bytes.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

If XER[TBC] = 0, **stswx** is treated as a no-op.

The PowerPC Architecture states that, if XER[TBC] = 0 and if the EA is such that a precise data exception would normally occur (if not for the zero length), **stswx** is treated as a no-op and the precise exception does not occur. Data storage exceptions and alignment exceptions are examples of precise data exceptions.

However, the architecture makes no statement regarding imprecise exceptions related to **stswx** with XER[TBC] = 0. The IOP 480 CPU generates an imprecise exception (machine check) on this instruction when all of the following conditions are true:

- The instruction passes all protection bounds checking
- The address is cacheable
- The address is passed to the data cache
- The address misses in the data cache (resulting in a line fill request)
- The address encounters some form of bus error (non-configured, for example).

Exceptions

An alignment exception occurs if MSR[LE] = 1.

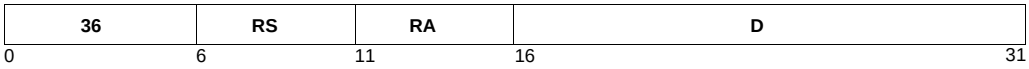
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stw

Store Lword

stw **RS, D(RA)**



EA ← (RA|0) + EXTS(D)
MS(EA, 4) ← (RS)

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The contents of register RS are stored at the EA.

Registers Altered

- None

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

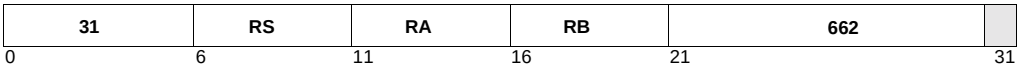
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwbrx

Store Lword Byte-Reverse Indexed

stwbrx **RS, RA, RB**



EA $\leftarrow$ (RA|0) + (RB)
MS(EA, 4) $\leftarrow$ $(RS)_{24:31} \parallel (RS)_{16:23} \parallel (RS)_{8:15} \parallel (RS)_{0:7}$

An EA is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The contents of register RS are byte-reversed: the least significant byte becomes the most significant byte, the next least significant byte becomes the next most significant byte, and so on. The result is stored into the Lword at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

Invalid Instruction Forms

- Reserved fields

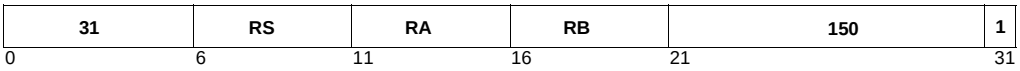
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwcx.

Store Lword Conditional Indexed

stwcx. RS, RA, RB



EA ← (RA|0) + (RB)
if RESERVE = 1 then
  MS(EA, 4) ← (RS)
  RESERVE ← 0
  (CR[CR0]) ← $^{2}0$ || 1 || XER_{so}
else
  (CR[CR0]) ← $^{2}0$ || 0 || XER_{so}

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

If the reservation bit contains 1 when the instruction is executed, the contents of register RS are stored into the Lword at the EA and the reservation bit is cleared. If the reservation bit contains 0 when the instruction is executed, no store operation is performed.

CR[CR0] is set as follows:

- $CR[CR0]_{LT, GT}$ are cleared
- $CR[CR0]_{EQ}$ is set to the state of the reservation bit at the start of the instruction
- $CR[CR0]_{SO}$ is set to the contents of the XER[SO] bit

Registers Altered

- $CR[CR0]_{LT, GT, EQ, SO}$

Programming Note

lwarx and the **stwcx.** instruction should paired in a loop, as shown in the following example, to create the effect of an atomic operation to a memory area used as a semaphore between asynchronous processes. Only **lwarx** can set the reservation bit to 1. **stwcx.** sets the reservation bit to 0 upon its completion, whether or not **stwcx.** sent (RS) to memory. $CR[CR0]_{EQ}$ must be examined to determine whether (RS) was sent to memory.

```
loop: lwarx                    # read the semaphore from memory; set reservation
“alter”   # change the semaphore bits in register as required
stwcx.   # attempt to store semaphore; reset reservation
bne loop# an asynchronous process has intervened; try again
```

If the asynchronous process in the code example had paired **lwarx** with a store other than **stwcx.**, the reservation bit would not have been cleared in the asynchronous process, and the code example would have overwritten the semaphore.

Exceptions

An alignment exception occurs if the EA is not Lword-aligned.

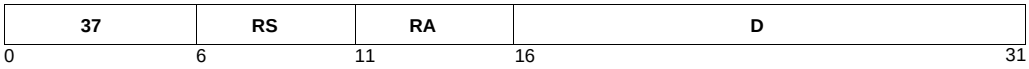
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwu

Store Lword with Update

stwu **RS, D(RA)**



EA ← (RA|0) + EXTS(D)
MS(EA, 4) ← (RS)
(RA) ← EA

An effective address (EA) is formed by adding a displacement to a base address. The displacement is obtained by sign-extending the 16-bit D field to 32 bits. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The contents of register RS are stored into the Lword at the EA.

The EA is placed into register RA.

Registers Altered

- RA

Invalid Instruction Forms

- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

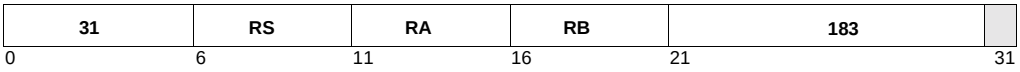
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwux

Store Lword with Update Indexed

stwux **RS, RA, RB**



EA ← (RA|0) + (RB)
MS(EA, 4) ← (RS)
(RA) ← EA

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The contents of register RS are stored into the Lword at the EA.

The EA is placed into register RA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RA

Invalid Instruction Forms

- Reserved fields
- RA = 0

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

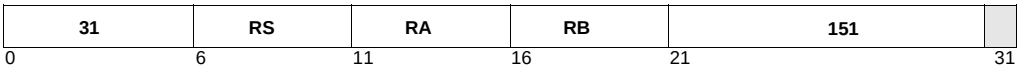
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwx

Store Lword Indexed

stwx **RS, RA, RB**



EA ← (RA|0) + (RB)
MS(EA,4) ← (RS)

An effective address (EA) is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 when the RA field is 0, and is the contents of register RA otherwise.

The contents of register RS are stored into the Lword at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Exceptions

An alignment exception occurs if MSR[LE] = 1 and the EA is not Lword-aligned.

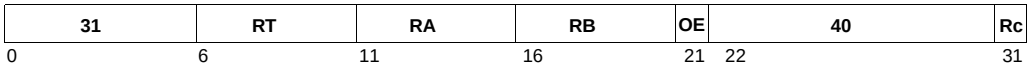
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subf

Subtract From

subf	**RT, RA, RB**	OE=0, Rc=0
subf.	**RT, RA, RB**	OE=0, Rc=1
subfo	**RT, RA, RB**	OE=1, Rc=0
subfo.	**RT, RA, RB**	OE=1, Rc=1



(RT) ← ¬(RA) + (RB) + 1

The sum of the ones complement of register RA, register RB, and 1 is stored into register RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

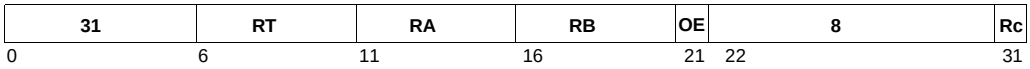
Table 28-30. Extended Mnemonics for subf, subf., subfo, subfo.

Mnemonic	Operands	Function	Other Registers Changed
sub	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. Extended mnemonic for subf RT,RB,RA	
sub.		Extended mnemonic for subf. RT,RB,RA	CR[CR0]
subo		Extended mnemonic for subfo RT,RB,RA	XER[SO, OV]
subo.		Extended mnemonic for subfo. RT,RB,RA	CR[CR0] XER[SO, OV]

subfc

Subtract From Carrying

subfc	**RT, RA, RB**	OE=0, Rc=0
subfc.	**RT, RA, RB**	OE=0, Rc=1
subfco	**RT, RA, RB**	OE=1, Rc=0
subfco.	**RT, RA, RB**	OE=1, Rc=1



(RT) ← ¬(RA) + (RB) + 1
if ¬(RA) + (RB) + 1 $\overset{U}{>}$ $2^{32} - 1$ then
    XER[CA] ← 1
else
    XER[CA] ← 0

The sum of the ones complement of register RA, register RB, and 1 is stored into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

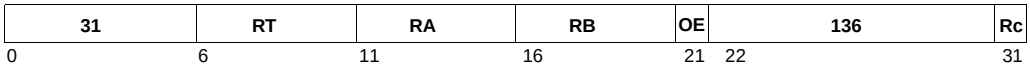
Table 28-31. Extended Mnemonics for subfc, subfc., subfco, subfco.

Mnemonic	Operands	Function	Other Registers Changed
subc	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. Place carry-out in XER[CA]. Extended mnemonic for subfc RT,RB,RA	
subc.		Extended mnemonic for subfc. RT,RB,RA	CR[CR0]
subco		Extended mnemonic for subfco RT,RB,RA	XER[SO, OV]
subco.		Extended mnemonic for subfco. RT,RB,RA	CR[CR0] XER[SO, OV]

subfe

Subtract From Extended

subfe	**RT, RA, RB**	OE=0, Rc=0
subfe.	**RT, RA, RB**	OE=0, Rc=1
subfeo	**RT, RA, RB**	OE=1, Rc=0
subfeo.	**RT, RA, RB**	OE=1, Rc=1



(RT) $\leftarrow$ $\neg$(RA) + (RB) + XER[CA]
if $\neg$(RA) + (RB) + XER[CA] $\overset{u}{\geq}$ $2^{32} - 1$ then
    XER[CA] $\leftarrow$ 1
else
    XER[CA] $\leftarrow$ 0

The sum of the ones complement of register RA, register RB, and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

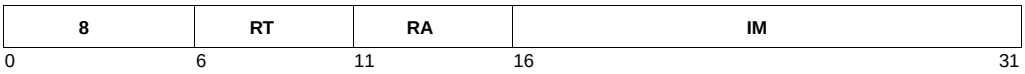
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subfic

Subtract From Immediate Carrying

subfic RT, RA, IM



(RT) $\leftarrow$ $\neg$(RA) + EXTS(IM) + 1
if $\neg$(RA) + EXTS(IM) + 1 $\overset{u}{>}$ $2^{32} - 1$ then
    XER[CA] $\leftarrow$ 1
else
    XER[CA] $\leftarrow$ 0

The sum of the ones complement of RA, the IM field sign-extended to 32 bits, and 1 is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]

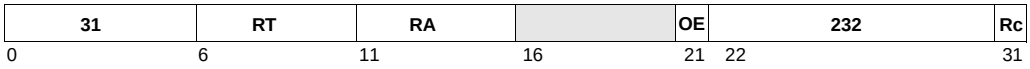
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subfme

Subtract From Minus One Extended

subfme	**RT, RA**	OE=0, Rc=0
subfme.	**RT, RA**	OE=0, Rc=1
subfmeo	**RT, RA**	OE=1, Rc=0
subfmeo.	**RT, RA**	OE=1, Rc=1



(RT) $\leftarrow$ $\neg$(RA) – 1 + XER[CA]
if $\neg$(RA) + 0xFFFF FFFF + XER[CA] $\overset{u}{\geq}$ 2^{32} – 1 then
    XER[CA] $\leftarrow$ 1
else
    XER[CA] $\leftarrow$ 0

The sum of the ones complement of register RA, –1, and XER[CA] is placed into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1
- XER[CA]

Invalid Instruction Forms

- Reserved fields

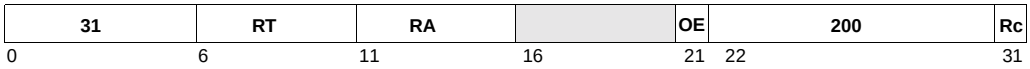
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subfze

Subtract From Zero Extended

subfze	**RT, RA**	OE=0, Rc=0
subfze.	**RT, RA**	OE=0, Rc=1
subfzeo	**RT, RA**	OE=1, Rc=0
subfzeo.	**RT, RA**	OE=1, Rc=1



(RT) $\leftarrow$ $\neg$(RA) + XER[CA]
if $\neg$(RA) + XER[CA] $\overset{u}{\geq}$ $2^{32} - 1$ then
    XER[CA] $\leftarrow$ 1
else
    XER[CA] $\leftarrow$ 0

The sum of the ones complement of register RA and XER[CA] is stored into register RT.

XER[CA] is set to a value determined by the unsigned magnitude of the result of the subtract operation.

Registers Altered

- RT
- XER[CA]
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

Invalid Instruction Forms

- Reserved fields

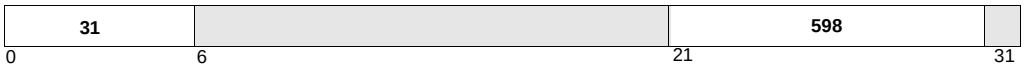
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sync

Synchronize

sync



Synchronize System

The **sync** instruction guarantees that all instructions initiated by the processor preceding the **sync** instruction complete before the **sync** instruction completes, and that no subsequent instructions are initiated by the processor until after **sync** completes. When **sync** completes, all storage accesses initiated by the processor prior to **sync** are completed with respect to all mechanisms that access storage.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None.

Invalid Instruction Forms

- Reserved fields

Programming Note

Architecturally, the **eieio** instruction orders storage access, not instruction completion. Therefore, non-storage operations that follow **eieio** could complete before storage operations that precede **eieio**. The **sync** instruction guarantees ordering of instruction completion and storage access. For the IOP 480 CPU, the **eieio** instruction is implemented to behave as a **sync** instruction. To write code that is portable between various PowerPC implementations, programmers should use the mnemonic which corresponds to the desired behavior.

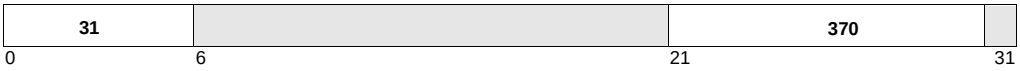
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

tlbia

TLB Invalidate All

tlbia



All of the entries in the TLB are invalidated and become unavailable for translation by clearing the valid (V) bit in the TLBHI portion of each TLB entry. The rest of the fields in the TLB entries are unmodified.

Registers Altered

- None.

Invalid Instruction Forms

- None.

Programming Note

This instruction is privileged. Translation is not required to be active during the execution of this instruction. The effects of the invalidation are not guaranteed to be visible to the programming model until the completion of a context synchronizing operation.

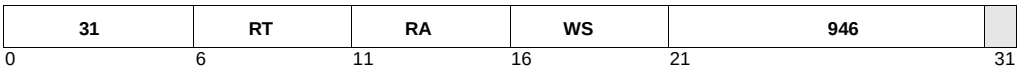
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

tlbre

TLB Read Entry

tlbre RT, RA, WS



if WS_4 = 1
(RT) ← TLBLO[($RA_{26:31}$)]
else
(RT) ← TLBHI[($RA_{26:31}$)]
(PID) ← TID from TLB[($RA_{26:31}$)]

The contents of the selected TLB entry is placed into register RT (and possibly into PID).

Bits 26:31 of the contents of RA is used as an index into the TLB. If this index specifies a TLB entry that does not exist, the results are undefined.

The WS field specifies which portion (TLBHI or TLBLO) of the entry is loaded into RT. If TLBHI is being accessed, the PID SPR is set to the value of the TID field in the TLB entry.

If the WS field is not 0 or 1, the instruction form is invalid and the result is undefined.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT
- PID (if WS = 0)

Invalid Instruction Forms

- Reserved fields
- Invalid WS value

Programming Notes

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

The contents of RT after the execution of this instruction are interpreted as follows:

If WS = 0 (TLBHI):

RT[0:21] ← EPN[0:21]
RT[22:24] ← SIZE[0:2]
RT[25] ← V

RT[26] ← E

RT[27] ← K

RT[28:31] ← 0
PID[24:31] ← TID[0:7]; (note that the TID is copied to the PID, not to RT)

If WS = 1 (TLBLO):

RT[0:21] ← RPN[0:21]
RT[22:23] ← EX,WR
RT[24:27] ← ZSEL[0:3]
RT[28:31] ← WIMG

Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

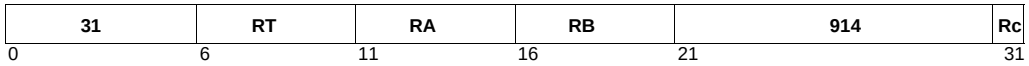
Table 28-32. Extended Mnemonics for tlbre

Mnemonic	Operands	Function	Other Registers Changed
tlbrehi	RT, RA	Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← $TLB[(RA)]_{TID}$ Extended mnemonic for tlbre RT,RA,0	
tlbrelo	RT, RA	Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)] Extended mnemonic for tlbre RT,RA,1	

tlbsx

TLB Search Indexed

tlbsx	**RT, RA, RB**	Rc=0
tlbsx.	**RT, RA, RB**	Rc=1




```
EA ← (RA|0) + (RB)
if Rc = 1
    CR[CR0]LT ← 0
    CR[CR0]GT ← 0
    CR[CR0]SO ← XER[SO}
if  Valid TLB entry matching EA and PID is in the TLB then
    (RT) ← Index of matching TLB Entry
    if Rc = 1
        CR[CR0]EQ ← 1
else
    (RT) Undefined
    if Rc = 1
        CR[CR0]EQ ← 0
```

An effective address is formed by adding an index to a base address. The index is the contents of register RB. The base address is 0 if the RA field is 0 and is the contents of register RA otherwise.

The TLB is searched for a valid entry which translates EA and PID. See Section 27.3.3, “Page Identification Fields,” on page 27-3, for details. The record bit (Rc) specifies whether the results of the search affect CR[CR0] as shown above. The intention is that $CR[CR0]_{EQ}$ can be tested after a **tlbsx.** instruction if there is a possibility that the search may fail.

Registers Altered

- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Invalid Instruction Forms

- None.

Programming Note

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

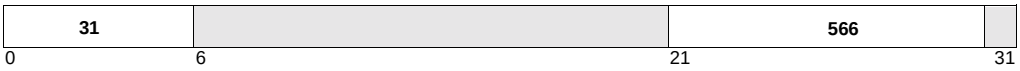
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

tlbsync

TLB Synchronize

tlbsync



The **tlbsync** instruction is provided in the PowerPC architecture to support synchronization of TLB operations among the processors of a multi-processor system. This instruction performs no operation on the IOP 480 CPU, and is provided to facilitate code portability.

Registers Altered

- None.

Invalid Instruction Forms

- None.

Programming Notes

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

Because the IOP 480 CPU does not support tightly-coupled multiprocessor systems, **tlbsync** performs no operation.

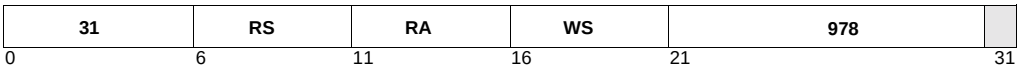
Architecture Note

This instruction is part of the PowerPC Operating Environment Architecture.

tlbwe

TLB Write Entry

tlbwe RS, RA, WS



if $WS_4 = 1$
  TLBLO[$(RA_{26:31})$] ← (RS)
else
  TLBHI[$(RA_{26:31})$] ← (RS)
  TID of TLB[$(RA_{26:31})$] ← $(PID_{24:31})$

The contents of the selected TLB entry is replaced with the contents of register RS (and possibly PID).

Bits 26:31 of the contents of RA are used as an index into the TLB. If this index specifies a TLB entry that does not exist, the results are undefined.

The WS field specifies which portion (TLBHI or TLBLO) of the entry is replaced from RS. For instructions that specify TLBHI, the TID field in the TLB entry is supplied from $PID_{24:31.}$

If the WS field is not 0 or 1, the instruction form is invalid and the result is undefined.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None.

Invalid Instruction Forms

- Reserved fields
- Invalid WS value

Programming Notes

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

The effects of this update are not guaranteed to be visible to the programming model until the completion of a context synchronizing operation. For example, updating a zone selection field within the TLB while in supervisor code should be followed by an **isync** instruction (or other context synchronizing operation) to guarantee that the desired translation and protection domains are used.

tlbwe writes the TLB fields from RS and the PID as follows:

If WS = 0 (TLBHI):

EPN[0:21] ← RS[0:21]
SIZE[0:2] ← RS[22:24]
V ← RS[25]
E ← RS[26]
K ← RS[27]
TID[0:7] ← PID[24:31]; (note that the TID is written from the PID, not RS)

If WS = 1 (TLBLO):

RPN[0:21] ← RS[0:21]
EX,WR ← RS[22:23]
ZSEL[0:3] ← RS[24:27]
WIMG ← RS[28:31]

Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

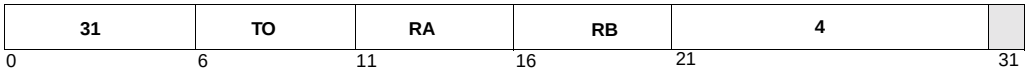
Table 28-33. Extended Mnemonics for tlbwe

Mnemonic	Operands	Function	Other Registers Changed
tlbwehi	RS, RA	Write TLBHI portion of the selected TLB entry from RS. Write the TID register of the selected TLB entry from the PID register. TLBHI[(RA)] ← (RS) $TLB[(RA)]_{TID}$ ← $(PID_{24:31})$ Extended mnemonic for tlbwe RS,RA,0	
tlbwelo	RS, RA	Write TLBLO portion of the selected TLB entry from RS. TLBLO[(RA)] ← (RS) Extended mnemonic for tlbwe RS,RA,1	

tw

Trap Lword

tw **TO, RA, RB**



if (((RA) $<$ (RB) $\wedge$ TO_0 = 1) $\vee$
((RA) $>$ (RB) $\wedge$ TO_1 = 1) $\vee$
((RA) $=$ (RB) $\wedge$ TO_2 = 1) $\vee$
((RA) $\leq$ (RB) $\wedge$ TO_3 = 1) $\vee$
((RA) $\geq$ (RB) $\wedge$ TO_4 = 1)) then TRAP (see details below)

Register RA is compared with register RB. If any comparison condition selected by the TO field is true, a TRAP occurs. The behavior of a TRAP depends upon the debug mode of the processor, as described below:

- If TRAP is not enabled as a debug event (DBCR[TDE] = 0 or DBCR[EDM,IDM] = 0,0):

  TRAP causes a program interrupt. See Section 11.7.6, “Data Storage Exceptions,” on page 11-21, for more information.

  (SRR0) ← address of **tw** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (MSR[WE, EE, PR, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || 0x0700

- If TRAP is enabled as an external debug event (DBCR[TDE] = 1 and DBCR[EDM] = 1):

  TRAP goes to the debug stop state, to be handled by an external debugger with hardware control.

  (DBSR[TIE]) ← 1

  In addition, if TRAP is also enabled as an internal debug event (DBCR[IDM] = 1) and debug exceptions are disabled (MSR[DE] = 0), then report an imprecise event:
  (DBSR[IDE]) ← 1

  PC ← address of **tw** instruction

- If TRAP is enabled as an internal debug event and *not* an external debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and debug exceptions are enabled (MSR[DE] = 1):

  TRAP causes a debug interrupt. See Section 11.7.17, “Debug Exception Handling,” on page 11-28, for more information.

  (SRR2) ← address of **tw** instruction
  (SRR3) ← (MSR)
  (DBSR[TIE]) ← 1
  (MSR[WE, EE, PR, CE, DE, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || 0x2000

- If TRAP is enabled as an internal debug event and *not* an external debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and Debug Exceptions are disabled (MSR[DE] = 0):

  TRAP reports the debug event as an *imprecise* event and causes a program interrupt. See Section 11.7.10, “Program Exceptions,” on page 11-24, for more information.

  (SRR0) ← address of **tw** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (DBSR[TIE,IDE]) ← 1,1
  (MSR[WE, EE, PR, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || 0x0700

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

Programming Note

This instruction is inserted into the execution stream by a debugger to implement breakpoints, and is not typically used by application code.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

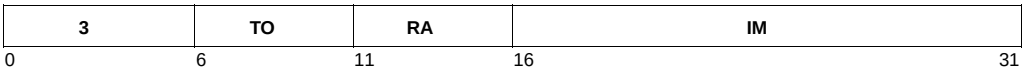
Table 28-34. Extended Mnemonics for tw

Mnemonic	Operands	Function	Other Registers Changed
trap		Trap unconditionally. Extended mnemonic for tw 31,0,0	
tweq	RA, RB	Trap if (RA) equal to (RB). Extended mnemonic for tw 4,RA,RB	
twge	RA, RB	Trap if (RA) greater than or equal to (RB). Extended mnemonic for tw 12,RA,RB	
twgt	RA, RB	Trap if (RA) greater than (RB). Extended mnemonic for tw 8,RA,RB	
twle	RA, RB	Trap if (RA) less than or equal to (RB). Extended mnemonic for tw 20,RA,RB	
twlge	RA, RB	Trap if (RA) logically greater than or equal to (RB). Extended mnemonic for tw 5,RA,RB	
twlgt	RA, RB	Trap if (RA) logically greater than (RB). Extended mnemonic for tw 1,RA,RB	
twlle	RA, RB	Trap if (RA) logically less than or equal to (RB). Extended mnemonic for tw 6,RA,RB	
twllt	RA, RB	Trap if (RA) logically less than (RB). Extended mnemonic for tw 2,RA,RB	
twlng	RA, RB	Trap if (RA) logically not greater than (RB). Extended mnemonic for tw 6,RA,RB	
twlnl	RA, RB	Trap if (RA) logically not less than (RB). Extended mnemonic for tw 5,RA,RB	
twlt	RA, RB	Trap if (RA) less than (RB). Extended mnemonic for tw 16,RA,RB	
twne	RA, RB	Trap if (RA) not equal to (RB). Extended mnemonic for tw 24,RA,RB	
twng	RA, RB	Trap if (RA) not greater than (RB). Extended mnemonic for tw 20,RA,RB	
twnl	RA, RB	Trap if (RA) not less than (RB). Extended mnemonic for tw 12,RA,RB	

twi

Trap Lword Immediate

twi **TO, RA, IM**



if (((RA) < EXTS(IM) ∧ TO_0 = 1) ∨
((RA) > EXTS(IM) ∧ TO_1 = 1) ∨
((RA) = EXTS(IM) ∧ TO_2 = 1) ∨
((RA) ≠ EXTS(IM) ∧ TO_3 = 1) ∨
((RA) ≥ EXTS(IM) ∧ TO_4 = 1)) then TRAP (see details below)

Register RA is compared with the IM field, which has been sign-extended to 32 bits. If any comparison condition selected by the TO field is true, a TRAP occurs. The behavior of a TRAP depends upon the Debug Mode of the processor, as described below:

- If TRAP is not enabled as a debug event (DBCR[TDE] = 0 or DBCR[EDM,IDM] = 0,0):

  TRAP causes a Program interrupt. See Section 11.7.10, “Program Exceptions,” on page 11-24, for more information.

  (SRR0) ← address of **twi** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (MSR[WE, EE, PR, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || 0x0700

- If TRAP is enabled as an External debug event (DBCR[TDE] = 1 and DBCR[EDM] = 1):

  TRAP goes to the Debug Stop state, to be handled by an external debugger with hardware control over the IOP 480 CPU.

  (DBSR[TIE]) ← 1
  In addition, if TRAP is also enabled as an Internal debug event (DBCR[IDM] = 1)
  and Debug Exceptions are disabled (MSR[DE] = 0), then report an imprecise event:
  (DBSR[IDE]) ← 1
  PC ← address of **twi** instruction

- If TRAP is enabled as an Internal debug event and *not* an External debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and Debug Exceptions are enabled (MSR[DE] = 1):

  TRAP causes a Debug interrupt. See Section 11.7.17, “Debug Exception Handling,” on page 11-28, for further information.

  (SRR2) ← address of **twi** instruction
  (SRR3) ← (MSR)
  (DBSR[TIE]) ← 1
  (MSR[WE, EE, PR, CE, DE, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || 0x2000

- If TRAP is enabled as an Internal debug event and *not* an External debug event (DBCR[TDE] = 1 and DBCR[EDM,IDM] = 0,1) and Debug Exceptions are disabled (MSR[DE] = 0):

  TRAP reports the debug event as an *imprecise* event and causes a Program interrupt. See Section 11.7.10, “Program Exceptions,” on page 11-24, for more information.

  (SRR0) ← address of **twi** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (DBSR[TIE,IDE]) ← 1,1
  (MSR[WE, EE, PR, DR, IR]) ← 0
  (MSR[LE]) ← (MSR[ILE])
  PC ← $EVPR_{0:15}$ || 0x0700

Registers Altered

- None

Programming Note

This instruction is inserted into the execution stream by a debugger to implement breakpoints, and is not typically used by application code.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

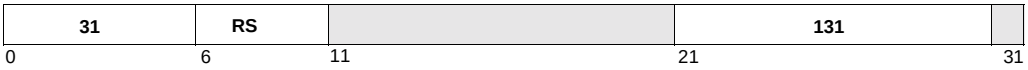
Table 28-35. Extended Mnemonics for twi

Mnemonic	Operands	Function	Other Registers Changed
tweqi	RA, IM	Trap if (RA) equal to EXTS(IM). *Extended mnemonic* for **twi 4,RA,IM**	
twgei	RA, IM	Trap if (RA) greater than or equal to EXTS(IM). Extended mnemonic for twi 12,RA,IM	
twgti	RA, IM	Trap if (RA) greater than EXTS(IM). Extended mnemonic for twi 8,RA,IM	
twlei	RA, IM	Trap if (RA) less than or equal to EXTS(IM). Extended mnemonic for twi 20,RA,IM	
twlgei	RA, IM	Trap if (RA) logically greater than or equal to EXTS(IM). Extended mnemonic for twi 5,RA,IM	
twlgti	RA, IM	Trap if (RA) logically greater than EXTS(IM). Extended mnemonic for twi 1,RA,IM	
twllei	RA, IM	Trap if (RA) logically less than or equal to EXTS(IM). Extended mnemonic for twi 6,RA,IM	
twllti	RA, IM	Trap if (RA) logically less than EXTS(IM). Extended mnemonic for twi 2,RA,IM	
twlngi	RA, IM	Trap if (RA) logically not greater than EXTS(IM). Extended mnemonic for twi 6,RA,IM	
twlnli	RA, IM	Trap if (RA) logically not less than EXTS(IM). Extended mnemonic for twi 5,RA,IM	
twlti	RA, IM	Trap if (RA) less than EXTS(IM). Extended mnemonic for twi 16,RA,IM	
twnei	RA, IM	Trap if (RA) not equal to EXTS(IM). Extended mnemonic for twi 24,RA,IM	
twngi	RA, IM	Trap if (RA) not greater than EXTS(IM). Extended mnemonic for twi 20,RA,IM	
twnli	RA, IM	Trap if (RA) not less than EXTS(IM). Extended mnemonic for twi 12,RA,IM	

wrtee

Write External Enable

wrtee **RS**



MSR[EE] ← $(RS)_{16}$

The MSR[EE] is set to the value specified by bit 16 of register RS.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- MSR[EE]

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

This instruction is used to provide atomic update of MSR[EE]. Typical usage is:

```
mfmsr    Rn          #Save EE in Rn[16]
wrteei   0           #Turn off EE
 • • • • • • • • • • • #Code with EE disabled
wrtee    Rn          #Restore EE without affecting other MSR changes
                       that may have occurred during the disabled code
```

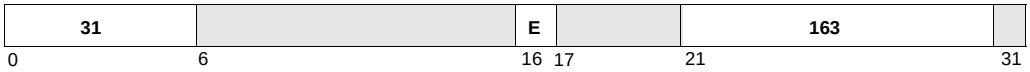
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

wrteei

Write External Enable Immediate

wrteei **E**



MSR[EE] ← E

The MSR[EE] is set to the value specified by the E field.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- MSR[EE]

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

This instruction is used to provide atomic update of MSR[EE]. Typical usage is:

```
mfmsr     Rn          #Save EE in Rn[16]
wrteei    0           #Turn off EE
 • • • • • • • • • • •  #Code with EE disabled
wrtee     Rn          #Restore EE without affecting other MSR changes
                        that may have occurred during the disabled code
```

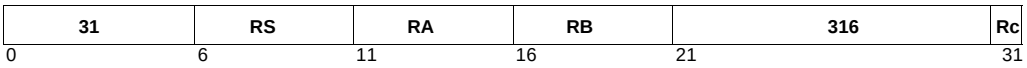
Architecture Note

Programs using this instruction are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

xor

XOR

xor	**RA, RS, RB**	Rc=0
xor.	**RA, RS, RB**	Rc=1



(RA) ← (RS) ⊕ (RB)

The contents of register RS are XORed with the contents of register RB; the result is placed into register RA.

Registers Altered

- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- RA

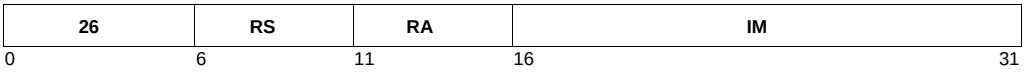
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

xori

XOR Immediate

xori **RA, RS, IM**



(RA) $\leftarrow$ (RS) $\oplus$ (^{16}0 || IM)

The IM field is extended to 32 bits by concatenating 16 0-bits on the left. The contents of register RS are XORed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA

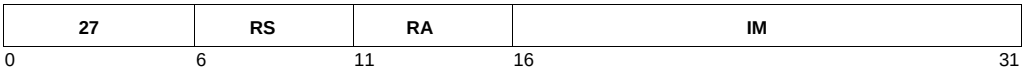
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

xoris

XOR Immediate Shifted

xoris **RA, RS, IM**



(RA) $\leftarrow$ (RS) $\oplus$ (IM || $^{16}0$)

The IM field is extended to 32 bits by concatenating 16 0-bits on the right. The contents of register RS are XORed with the extended IM field; the result is placed into register RA.

Registers Altered

- RA

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

29 IOP 480 CPU REGISTER SUMMARY

All registers contained in the IOP 480 CPU are architected as 32 bits. Register 29-1 and Register 29-2 define required addressing to access the registers. The pages following these tables define the bit usage within each register.

29.1 RESERVED REGISTERS

Any register numbers not listed in these tables are *reserved,* and should be neither read nor written. These reserved register numbers may be used for additional functions on future PowerPC Embedded Controllers.

29.2 RESERVED FIELDS

For all registers with fields marked as ***reserved***, the reserved fields should be written as *zero* and read as *undefined*. That is, when writing to a reserved field, write a zero to that field. When reading from a reserved field, ignore that field.

A good coding practice is to perform the initial write to a register with reserved fields as described in the preceding paragraph, and to perform all subsequent writes to the register using a read-modify-write strategy: read the register, alter desired fields with logical instructions, and then write the register.

29.3 GENERAL PURPOSE REGISTERS

The IOP 480 CPU contains 32 General Purpose Registers (GPRs). The contents of these registers can be loaded from memory using load instructions and stored to memory using store instructions. GPRs are also addressed by all integer instructions.

Register 29-1 lists the IOP 480 CPU General Purpose Registers.

29.4 MACHINE STATE REGISTER AND CONDITION REGISTER

Because these registers are accessed using special instructions, they do not require addressing.

29.5 SPECIAL PURPOSE REGISTERS

Special Purpose Registers (SPRs), which are part of the PowerPC Embedded Architecture, are accessed using the **mtspr** and **mfspr** instructions. SPRs control the use of the debug facilities, timers, interrupts, storage control attributes, and other architected processor resources.

Register 29-2 shows the mnemonics, names, and numbers of the SPRs. The columns under “SPRN” list the register numbers used as operands in assembler language coding of the **mfspr** and **mtspr** instructions. The column labeled “SPRF” lists the corresponding fields contained in the *machine code* of **mfspr** and **mtspr**. The SPRN field contains two five-bit subfields of the SPRF field; the subfields are *reversed* in the machine code for the **mfspr** and **mtspr** instructions ($SPRN \leftarrow SPRF_{5:9} \parallel SPRF_{0:4}$) for compatibility with the POWER Architecture. Note that the assembler handles the special coding transparently.

The only SPRs that are not privileged are the Count Register (CTR), the Link Register (LR), the Time Base High User-mode (TBHU), the Time Base Low User-mode (TBLU), and the Fixed-Point Exception Register (XER). See Section 24.8.3, “Privileged SPRs,” on page 24-28.

Register 29-2 lists the SPRs, their mnemonics and names, numbers (SPRN) and corresponding SPRF numbers, and access. All SPR numbers not listed are reserved, and should be neither read nor written.

Register 29-1. IOP 480 CPU General Purpose Registers

Mnemonic	Register Name	GPR Number		Access
		Decimal	Hex	
R0–R31	General Purpose Register 0–31	0–31	0x0–0x1F	Read/Write

Register 29-2. Special Purpose Registers

Mnemonic	Register Name	SPRN		SPRF	Access
		Decimal	Hex		
CDBCR	Cache Debug Control Register	983	0x3D7	0x2FE	Read/Write
CTR	Count Register	9	0x009	0x120	Read/Write
DAC	Data Address Compare	1014	0x3F6	0x2DF	Read/Write
DBCR	Debug Control Register	1010	0x3F2	0x25F	Read/Write
DBSR	Debug Status Register	1008	0x3F0	0x21F	Read/Clear
DCCR	Data Cache Cacheability Register	1018	0x3FA	0x35F	Read/Write
DCWR	Data Cache Write-through Register	954	0x3BA	0x35D	Read/Write
DEAR	Data Error Address Register	981	0x3D5	0x2BE	Read/Write
ESR	Exception Syndrome Register	980	0x3D4	0x29E	Read/Write
EVPR	Exception Vector Prefix Register	982	0x3D6	0x2DE	Read/Write
IAC	Instruction Address Compare	1012	0x3F4	0x29F	Read/Write
ICCR	Instruction Cache Cacheability Register	1019	0x3FB	0x37F	Read/Write
ICDBDR	Instruction Cache Debug Data Register	979	0x3D3	0x27E	Read-only
LR	Link Register	8	0x008	0x100	Read/Write
PID	Process ID	945	0x3B1	0x23D	Read/Write
PIT	Programmable Interval Timer	987	0x3DB	0x37E	Read/Write
PVR	Processor Version Register	287	0x11F	0x3E8	Read-only
SGR	Storage Guarded Register	953	0x3B9	0x33D	Read/Write
SKR	Storage Compression Register	956	0x3BC	0x39D	Read/Write
SLER	Storage Little Endian Register	955	0x3BB	0x37D	Read/Write
SPRG0	SPR General Purpose Register 0	272	0x110	0x208	Read/Write
SPRG1	SPR General Purpose Register 1	273	0x111	0x228	Read/Write
SPRG2	SPR General Purpose Register 2	274	0x112	0x248	Read/Write
SPRG3	SPR General Purpose Register 3	275	0x113	0x268	Read/Write
SRR0	Save/Restore Register 0	26	0x01A	0x340	Read/Write
SRR1	Save/Restore Register 1	27	0x01B	0x360	Read/Write
SRR2	Save/Restore Register 2	990	0x3DE	0x3DE	Read/Write
SRR3	Save/Restore Register 3	991	0x3DF	0x3FE	Read/Write
TBHI	Time Base High	988	0x3DC	0x39E	Read/Write
TBHU	Time Base High User-mode	972	0x3CC	0x19E	Read Only
TBLO	Time Base Low	989	0x3DD	0x3BE	Read/Write
TBLU	Time Base Low User-mode	973	0x3CD	0x1BE	Read Only
TCR	Timer Control Register	986	0x3DA	0x35E	Read/Write
TSR	Timer Status Register	984	0x3D8	0x31E	Read/Clear
XER	Fixed Point Exception Register	1	0x001	0x020	Read/Write
ZPR	Zone Protection Register	944	0x3B0	0x21D	Privileged

29.6 DEVICE CONTROL REGISTERS

Device Control Registers (DCRs), which are architecturally outside of the processor core, are accessed using the **mtdcr** and **mfdcr** instructions. DCRs are used to control, configure, and hold status for various functional units that are not part of the RISC processor core. Although the IOP 480 CPU does not contain DCRs, the **mtdcr** and **mfdcr** instructions are provided.

The **mtdcr** and **mfdcr** instructions are privileged, for all DCR numbers. Therefore, all accesses to DCRs are privileged. See Section 24.8, “Privileged Mode Operation,” on page 24-27.

All DCR numbers reserved, and should not be read nor written, unless they are part of a Core+ASIC implementation.

29.7 ALPHABETICAL REGISTER LISTING

The following pages list the registers available in the IOP 480 CPU. For each register, the following information is supplied:

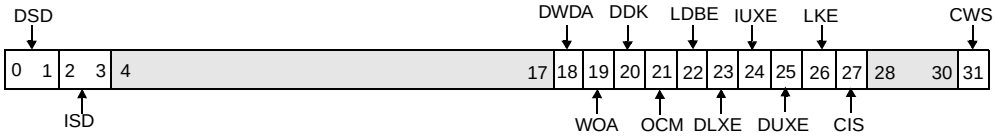
- Register name and mnemonic
- Register type (SPR)
- Register number (address)
- A diagram illustrating the register fields (all register fields have mnemonics, unless there is only one field)
- A table describing the register fields, giving field mnemonic, bit location, name, and function associated with various field values

CDBCR

SPR 0x3D7

See also Section 25.4, “Cache Control and Debugging Features,” on page 25-7.

Register 29-3. Cache Debug Control Register (CDBCR)



0:1	DSD	DCU Size Disable 00 The connected cache size is the real cache size. 01 The connected cache size is the real cache size/2. 10 The connected cache size is the real cache size/4. 11 The connected cache size is the real cache size/8.	See Table 25-5, “CDBCR[DSD], CDBCR[ISD], and Effective Cache Size,” on page 25-15, for more information on bit settings for various real and effective cache sizes.
2:3	ISD	ICU Size Disable 00 The connected cache size is the real cache size. 01 The connected cache size is the real cache size/2. 10 The connected cache size is the real cache size/4. 11 The connected cache size is the real cache size/8.	See Table 25-5, “CDBCR[DSD], CDBCR[ISD], and Effective Cache Size,” on page 25-15, for more information on bit settings for various real and effective cache sizes.
4:17		***Reserved***	
18	DWDA	Delayed Write Data Acknowledge 0 Normal processor write data acknowledge. 1 Write data acknowledge is delayed one processor clock cycle.	
19	WOA	Write-on-Allocate 0 All store misses result in a line fill. 1 Store misses do not cause a line fill, but result in a non-cacheable store.	
20	DDK	Disable Data-side Compression 0 Use K storage attribute to specify data compression 1 Disable data-side compression, regardless of K attribute	
21	OCM	Instruction-side OCM (IOCM) Mode 0 IOCM is presented only with cacheable fetches 1 IOCM is presented with cacheable and non-cacheable fetches	
22	LDBE	Load Debug Enable 0 Load data is invisible on data-side OCM 1 Load data is visible on data-side OCM	
23	DLXE	DCU Lock-out Exception Enable 0 DCU lock-out exception is disabled. 1 DCU lock-out exception is enabled.	
24	IUXE	ICU Unlock Exception Enable 0 ICU unlock exception is disabled. 1 ICU unlock exception is enabled.	
25	DUXE	DCU Unlock Exception Enable 0 DCU unlock exception is disabled. 1 DCU unlock exception is enabled.	

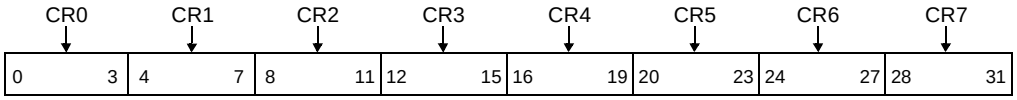
Register 29-3. Cache Debug Control Register (CDBCR) (Continued)

26	LKE	Lock Enable 0 Line locking is disabled. 1 Line locking is enabled.
27	CIS	Cache Information Select 0 Information is cache data. 1 Information is cache tag.
28:30		***Reserved***
31	CWS	Cache Way Select 0 Cache way is A. 1 Cache way is B.

CR

See also Section 24.2.3, “Condition Register (CR),” on page 24-7.

Register 29-4. Condition Register (CR)



0:3	CR0	Condition Register Field 0	CR[CRn]$_{0:3}$ indicate less than, greater than, equal to, and summary overflow, respectively.
4:7	CR1	Condition Register Field 1	See the description of CR[CR0].
8:11	CR2	Condition Register Field 2	See the description of CR[CR0].
12:15	CR3	Condition Register Field 3	See the description of CR[CR0].
16:19	CR4	Condition Register Field 4	See the description of CR[CR0].
20:23	CR5	Condition Register Field 5	See the description of CR[CR0].
24:27	CR6	Condition Register Field 6	See the description of CR[CR0].
28:31	CR7	Condition Register Field 7	See the description of CR[CR0].

CTR

SPR 0x009

See also Section 24.2.2.1, “Count Register (CTR),” on page 24-4.

Register 29-5. Count Register (CTR)

0:31		Count	Used as count for branch conditional with decrement instructions, or as address for branch-to-counter instructions.

DAC1

SPR 0x3F6

See also Section 26.6.3, “Data Address Compare Register (DAC1),” on page 26-7.

Register 29-6. Data Address Compare Register (DAC1)

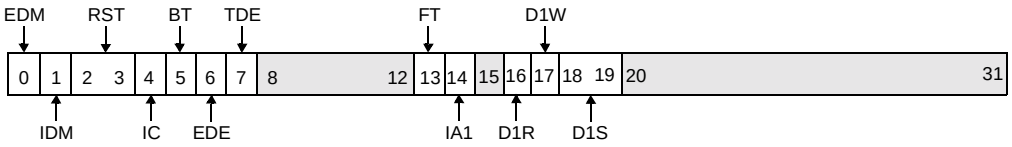
0:31		Data Address Compare (DAC) byte address	DBCR[D1S] determines byte, word, or Lword usage.

DBCR

SPR 0x3F2

See also Section 26.6.1, “Debug Control Register (DBCR),” on page 26-3.

Register 29-7. Debug Control Register (DBCR)



0	EDM	External Debug Mode 0 Disable 1 Enable	
1	IDM	Internal Debug Mode 0 Disable 1 Enable	
2:3	RST	Reset 00 No action 01 Core reset 10 Chip reset 11 System reset ***Note:*** Writing 01, 10, or 11 to this field causes a processor reset request.	
4	IC	Instruction Completion Debug Event 0 Disable 1 Disable	Instruction completion does not cause a debug event if MSR[DE] = 0 in internal debug mode.
5	BT	Branch Taken Debug Event 0 Disable 1 Enable	Branch taken does not cause a debug event if MSR[DE] = 0 in internal debug mode.
6	EDE	Exception Debug Event 0 Disable 1 Enable	Critical exceptions do not cause debug events if MSR[DE] = 0 in internal debug mode.
7	TDE	TRAP Debug Event 0 Disable 1 Enable	
8:12		***Reserved***	
13	FT	Freeze timers on debug event 0 Free-run timers 1 Freeze timers	
14	IA1	Instruction Address Compare 1 Enable 0 Disable 1 Enable	
15		***Reserved***	
16	D1R	DAC Read Enable 0 Disable 1 Enable	
17	D1W	DAC Write Enable 0 Disable 1 Enable	

Register 29-7. Debug Control Register (DBCR) (Continued)

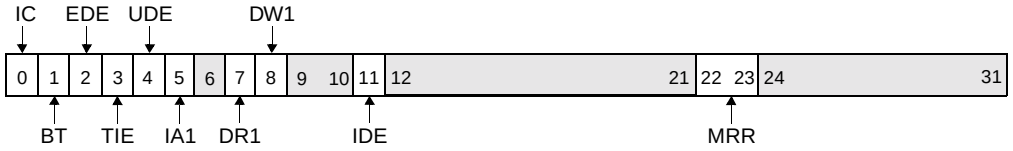
18:19	D1S	DAC Size	
		00 Compare all bits	Exact address compare
		01 Ignore the least significant bit (lsb)	Byte within word address compare
		10 Ignore the two lsbs	Byte within Lword address compare
		11 Ignore the four lsbs	Qword address compare
20:31		***Reserved***	

DBSR

SPR 0x3F0 Read/Clear

See also Section 26.6.2, “Debug Status Register (DBSR),” on page 26-5.

Register 29-8. Debug Status Register (DBSR)



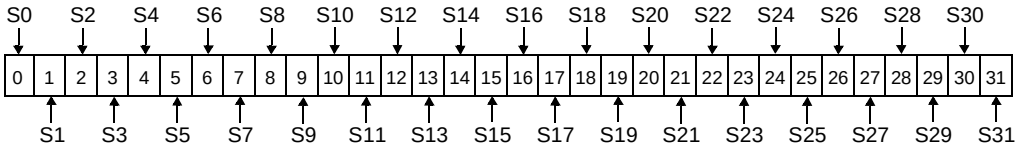
0	IC	Instruction Completion Debug Event 0 Event didn’t occur 1 Event occurred	
1	BT	Branch Taken Debug Event 0 Event didn’t occur 1 Event occurred	
2	EDE	Exception Debug Event 0 Event didn’t occur 1 Event occurred	
3	TIE	TRAP Instruction Debug Event 0 Event didn’t occur 1 Event occurred	
4	UDE	Unconditional Debug Event 0 Event didn’t occur 1 Event occurred	
5	IA1	IAC1 Debug Event 0 Event didn’t occur 1 Event occurred	
6		***Reserved***	
7	DR1	DAC Read Debug Event 0 Event didn’t occur 1 Event occurred	
8	DW1	DAC Write Debug Event 0 Event didn’t occur 1 Event occurred	
9:10		***Reserved***	
11	IDE	Imprecise Debug Event 0 Event didn’t occur 1 Event occurred	
12:21		***Reserved***	
22:23	MRR	Most Recent Reset 00 No reset has occurred since last cleared by software. 01 Core reset 10 Chip reset 11 System reset	This field is set to a value, indicating the type of reset, when a reset occurs.
24:31		***Reserved***	

DCCR

SPR 0x3FA

See Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13.

Register 29-9. Data Cache Cacheability Register (DCCR)



0	S0	0 Noncacheable 1 Cacheable	0x0000 0000 – 0x07FF FFFF
1	S1	0 Noncacheable 1 Cacheable	0x0800 0000 – 0x0FFF FFFF
2	S2	0 Noncacheable 1 Cacheable	0x1000 0000 – 0x17FF FFFF
3	S3	0 Noncacheable 1 Cacheable	0x1800 0000 – 0x1FFF FFFF
4	S4	0 Noncacheable 1 Cacheable	0x2000 0000 – 0x27FF FFFF
5	S5	0 Noncacheable 1 Cacheable	0x2800 0000 – 0x2FFF FFFF
6	S6	0 Noncacheable 1 Cacheable	0x3000 0000 – 0x37FF FFFF
7	S7	0 Noncacheable 1 Cacheable	0x3800 0000 – 0x3FFF FFFF
8	S8	0 Noncacheable 1 Cacheable	0x4000 0000 – 0x47FF FFFF
9	S9	0 Noncacheable 1 Cacheable	0x4800 0000 – 0x4FFF FFFF
10	S10	0 Noncacheable 1 Cacheable	0x5000 0000 – 0x57FF FFFF
11	S11	0 Noncacheable 1 Cacheable	0x5800 0000 – 0x5FFF FFFF
12	S12	0 Noncacheable 1 Cacheable	0x6000 0000 – 0x67FF FFFF
13	S13	0 Noncacheable 1 Cacheable	0x6800 0000 – 0x6FFF FFFF
14	S14	0 Noncacheable 1 Cacheable	0x7000 0000 – 0x77FF FFFF
15	S15	0 Noncacheable 1 Cacheable	0x7800 0000 – 0x7FFF FFFF
16	S16	0 Noncacheable 1 Cacheable	0x8000 0000 – 0x87FF FFFF
17	S17	0 Noncacheable 1 Cacheable	0x8800 0000 – 0x8FFF FFFF

Register 29-9. Data Cache Cacheability Register (DCCR) (Continued)

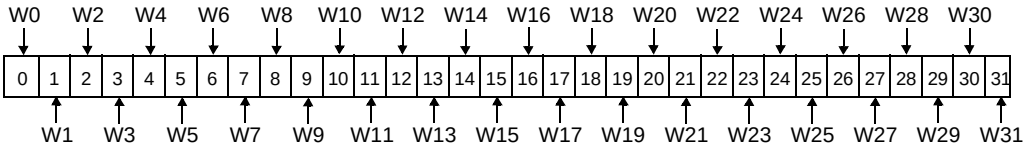
18	S18	0 Noncacheable 1 Cacheable	0x9000 0000 – 0x97FF FFFF
19	S19	0 Noncacheable 1 Cacheable	0x9800 0000 – 0x9FFF FFFF
20	S20	0 Noncacheable 1 Cacheable	0xA000 0000 – 0xA7FF FFFF
21	S21	0 Noncacheable 1 Cacheable	0xA800 0000 – 0xAFFF FFFF
22	S22	0 Noncacheable 1 Cacheable	0xB000 0000 – 0xB7FF FFFF
23	S23	0 Noncacheable 1 Cacheable	0xB800 0000 – 0xBFFF FFFF
24	S24	0 Noncacheable 1 Cacheable	0xC000 0000 – 0xC7FF FFFF
25	S25	0 Noncacheable 1 Cacheable	0xC800 0000 – 0xCFFF FFFF
26	S26	0 Noncacheable 1 Cacheable	0xD000 0000 – 0xD7FF FFFF
27	S27	0 Noncacheable 1 Cacheable	0xD800 0000 – 0xDFFF FFFF
28	S28	0 Noncacheable 1 Cacheable	0xE000 0000 – 0xE7FF FFFF
29	S29	0 Noncacheable 1 Cacheable	0xE800 0000 – 0xEFFF FFFF
30	S30	0 Noncacheable 1 Cacheable	0xF000 0000 – 0xF7FF FFFF
31	S31	0 Noncacheable 1 Cacheable	0xF800 0000 – 0xFFFF FFFF

DCWR

SPR 0x3BA

See Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13.

Register 29-10. Data Cache Write-thru Register (DCWR)



0	W0	0 Write-back 1 Write-through	0x0000 0000 – 0x07FF FFFF
1	W1	0 Write-back 1 Write-through	0x0800 0000 – 0x0FFF FFFF
2	W2	0 Write-back 1 Write-through	0x1000 0000 – 0x17FF FFFF
3	W3	0 Write-back 1 Write-through	0x1800 0000 – 0x1FFF FFFF
4	W4	0 Write-back 1 Write-through	0x2000 0000 – 0x27FF FFFF
5	W5	0 Write-back 1 Write-through	0x2800 0000 – 0x2FFF FFFF
6	W6	0 Write-back 1 Write-through	0x3000 0000 – 0x37FF FFFF
7	W7	0 Write-back 1 Write-through	0x3800 0000 – 0x3FFF FFFF
8	W8	0 Write-back 1 Write-through	0x4000 0000 – 0x47FF FFFF
9	W9	0 Write-back 1 Write-through	0x4800 0000 – 0x4FFF FFFF
10	W10	0 Write-back 1 Write-through	0x5000 0000 – 0x57FF FFFF
11	W11	0 Write-back 1 Write-through	0x5800 0000 – 0x5FFF FFFF
12	W12	0 Write-back 1 Write-through	0x6000 0000 – 0x67FF FFFF
13	W13	0 Write-back 1 Write-through	0x6800 0000 – 0x6FFF FFFF
14	W14	0 Write-back 1 Write-through	0x7000 0000 – 0x77FF FFFF
15	W15	0 Write-back 1 Write-through	0x7800 0000 – 0x7FFF FFFF
16	W16	0 Write-back 1 Write-through	0x8000 0000 – 0x87FF FFFF
17	W17	0 Write-back 1 Write-through	0x8800 0000 – 0x8FFF FFFF

Register 29-10. Data Cache Write-thru Register (DCWR) (Continued)

18	W18	0 Write-back 1 Write-through	0x9000 0000 – 0x97FF FFFF
19	W19	0 Write-back 1 Write-through	0x9800 0000 – 0x9FFF FFFF
20	W20	0 Write-back 1 Write-through	0xA000 0000 – 0xA7FF FFFF
21	W21	0 Write-back 1 Write-through	0xA800 0000 – 0xAFFF FFFF
22	W22	0 Write-back 1 Write-through	0xB000 0000 – 0xB7FF FFFF
23	W23	0 Write-back 1 Write-through	0xB800 0000 – 0xBFFF FFFF
24	W24	0 Write-back 1 Write-through	0xC000 0000 – 0xC7FF FFFF
25	W25	0 Write-back 1 Write-through	0xC800 0000 – 0xCFFF FFFF
26	W26	0 Write-back 1 Write-through	0xD000 0000 – 0xD7FF FFFF
27	W27	0 Write-back 1 Write-through	0xD800 0000 – 0xDFFF FFFF
28	W28	0 Write-back 1 Write-through	0xE000 0000 – 0xE7FF FFFF
29	W29	0 Write-back 1 Write-through	0xE800 0000 – 0xEFFF FFFF
30	W30	0 Write-back 1 Write-through	0xF000 0000 – 0xF7FF FFFF
31	W31	0 Write-back 1 Write-through	0xF800 0000 – 0xFFFF FFFF

DEAR

SPR 0x3D5

See also Section 11.7.3.6, “Data Exception Address Register (DEAR),” on page 11-19.

Register 29-11. Data Exception Address Register (DEAR)

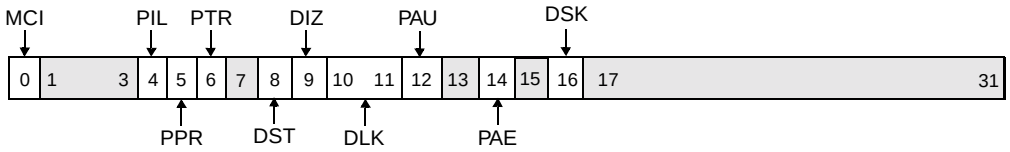
0:31		Address of Data Error (synchronous)

ESR

SPR 0x3D4

See also Section 11.7.3.5, “Exception Syndrome Register (ESR),” on page 11-17.

Register 29-12. Exception Syndrome Register (ESR)



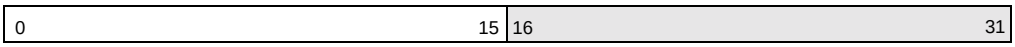
0	MCI	Machine check—instruction 0 Instruction machine check did not occur. 1 Instruction machine check occurred.
1:3		***Reserved***
4	PIL	Program exception—illegal 0 Illegal Instruction error did not occur. 1 Illegal Instruction error occurred.
5	PPR	Program exception—privileged 0 Privileged instruction error did not occur. 1 Privileged instruction error occurred.
6	PTR	Program exception—trap 0 Trap with successful compare did not occur. 1 Trap with successful compare occurred.
7		***Reserved***
8	DST	Data storage exception—store fault 0 Excepting instruction was not a store. 1 Excepting instruction was a store (includes **dcbi**, **dcbz**, and **dccci**).
9	DIZ	Data/instruction storage exception—zone fault 0 Excepting condition was not a zone fault. 1 Excepting condition was a zone fault.
10:11	DLK	Data Storage exception— lock fault 00 No lock exception 01 **dcbf** unlock exception 10 **icbi** unlock exception 11 **dcbz** lock-out exception
12	PAU	Program exception—auxiliary processor unavailable 0 Auxiliary processor unavailable exception did not occur. 1 Auxiliary processor unavailable exception occurred.
13		***Reserved***
14	PAE	Program exception—auxiliary processor enabled 0 Auxiliary processor enabled exception did not occur. 1 Auxiliary processor enabled exception occurred.
15		***Reserved***
16	DSK	Data storage exception—compressed 0 Excepting instruction did not access compressed storage. 1 Excepting instruction accessed compressed storage.
17:31		***Reserved***

EVPR

SPR 0x3D6

See also Section 11.7.3.4, “Exception Vector Prefix Register (EVPR),” on page 11-17.

Register 29-13. Exception Vector Prefix Register (EVPR)



0:15		Exception Vector Prefix
16:31		***Reserved***

R0-R31

See also Section 24.2.1, “General Purpose Registers,” on page 24-2.

Register 29-14. General Purpose Register (R0-R31)

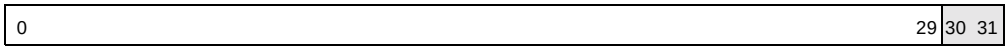
0:31		General Purpose Register data

IAC1

SPR 0x3F4

See also Section 26.6.4, “Instruction Address Compare Register (IAC1),” on page 26-8.

Register 29-15. Instruction Address Compare Register (IAC1)



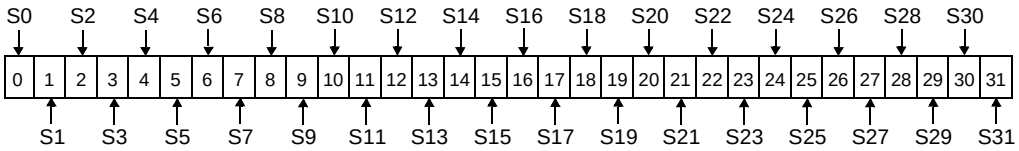
0:29		Instruction Address Compare Lword address	Omit two low-order bits of complete address.
30:31		***Reserved***	

ICCR

SPR 0x3FB

See Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13.

Register 29-16. Instruction Cache Cacheability Register (ICCR)



0	S0	0 Noncacheable 1 Cacheable	0x0000 0000 – 0x07FF FFFF
1	S1	0 Noncacheable 1 Cacheable	0x0800 0000 – 0x0FFF FFFF
2	S2	0 Noncacheable 1 Cacheable	0x1000 0000 – 0x17FF FFFF
3	S3	0 Noncacheable 1 Cacheable	0x1800 0000 – 0x1FFF FFFF
4	S4	0 Noncacheable 1 Cacheable	0x2000 0000 – 0x27FF FFFF
5	S5	0 Noncacheable 1 Cacheable	0x2800 0000 – 0x2FFF FFFF
6	S6	0 Noncacheable 1 Cacheable	0x3000 0000 – 0x37FF FFFF
7	S7	0 Noncacheable 1 Cacheable	0x3800 0000 – 0x3FFF FFFF
8	S8	0 Noncacheable 1 Cacheable	0x4000 0000 – 0x47FF FFFF
9	S9	0 Noncacheable 1 Cacheable	0x4800 0000 – 0x4FFF FFFF
10	S10	0 Noncacheable 1 Cacheable	0x5000 0000 – 0x57FF FFFF
11	S11	0 Noncacheable 1 Cacheable	0x5800 0000 – 0x5FFF FFFF
12	S12	0 Noncacheable 1 Cacheable	0x6000 0000 – 0x67FF FFFF
13	S13	0 Noncacheable 1 Cacheable	0x6800 0000 – 0x6FFF FFFF
14	S14	0 Noncacheable 1 Cacheable	0x7000 0000 – 0x77FF FFFF
15	S15	0 Noncacheable 1 Cacheable	0x7800 0000 – 0x7FFF FFFF
16	S16	0 Noncacheable 1 Cacheable	0x8000 0000 – 0x87FF FFFF
17	S17	0 Noncacheable 1 Cacheable	0x8800 0000 – 0x8FFF FFFF

Register 29-16. Instruction Cache Cacheability Register (ICCR) (Continued)

18	S18	0 Noncacheable 1 Cacheable	0x9000 0000 – 0x97FF FFFF
19	S19	0 Noncacheable 1 Cacheable	0x9800 0000 – 0x9FFF FFFF
20	S20	0 Noncacheable 1 Cacheable	0xA000 0000 – 0xA7FF FFFF
21	S21	0 Noncacheable 1 Cacheable	0xA800 0000 – 0xAFFF FFFF
22	S22	0 Noncacheable 1 Cacheable	0xB000 0000 – 0xB7FF FFFF
23	S23	0 Noncacheable 1 Cacheable	0xB800 0000 – 0xBFFF FFFF
24	S24	0 Noncacheable 1 Cacheable	0xC000 0000 – 0xC7FF FFFF
25	S25	0 Noncacheable 1 Cacheable	0xC800 0000 – 0xCFFF FFFF
26	S26	0 Noncacheable 1 Cacheable	0xD000 0000 – 0xD7FF FFFF
27	S27	0 Noncacheable 1 Cacheable	0xD800 0000 – 0xDFFF FFFF
28	S28	0 Noncacheable 1 Cacheable	0xE000 0000 – 0xE7FF FFFF
29	S29	0 Noncacheable 1 Cacheable	0xE800 0000 – 0xEFFF FFFF
30	S30	0 Noncacheable 1 Cacheable	0xF000 0000 – 0xF7FF FFFF
31	S31	0 Noncacheable 1 Cacheable	0xF800 0000 – 0xFFFF FFFF

ICDBDR

SPR 0x3D3 Read-Only

See also Section 25.4, “Cache Control and Debugging Features,” on page 25-7.

Register 29-17. Instruction Cache Debug Data Register (ICDBDR)

0:31		Instruction cache information	See **icread**, page 28-69.

Register 29-18. ICU Tag Information

0:*m*–1	TAG	Cache Tag	See Table 25-1, “Cache Array Size by Core,” on page 25-1 for information on the size of this variable-length field.
m:24		***Reserved***	The size of this field depends on the size of the tag field.
25	LK	Cache Line Lock 0 Unlocked 1 Locked	
26		***Reserved***	
27	V	Cache Line Valid 0 Not valid 1 Valid	
28:30		***Reserved***	
31	LRU	Least Recently Used (LRU) 0 A-way LRU 1 B-way LRU	

LR

SPR 0x008

See also Section 24.2.2.2, “Link Register (LR),” on page 24-4.

Register 29-19. Link Register (LR)

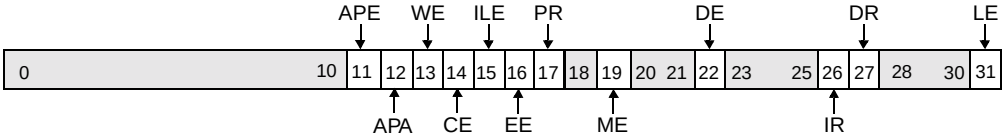
0:31		Link Registers contents	If (LR) represents an instruction address, $LR_{0:31}$ should be zero.

MSR

SPR 0x3B1

See also Section 11.7.3.1, “Machine State Register (MSR),” on page 11-13.

Register 29-20. Machine State Register (MSR)



0:10		***Reserved***	
11	APE	Auxiliary Processor Exception Enable 0 Auxiliary processor exception disabled. 1 Auxiliary processor exception enabled.	
12	APA	Auxiliary Processor Available 0 Auxiliary processor not available. 1 Auxiliary processor available.	
13	WE	Wait State Enable 0 The processor is not in the wait state. 1 The processor is in the wait state.	If MSR[WE] = 1, the processor remains in the wait state until an exception is taken, a reset occurs, or an external debug tool clears WE. This bit places the processor in a sleep mode. Typical power in sleep mode with no other on-going operations is under 150 mW.
14	CE	Critical Interrupt Enable 0 Critical interrupts are disabled. 1 Critical interrupts are enabled.	Controls the critical interrupt input and watchdog timer first timeout interrupts.
15	ILE	Interrupt Little Endian 0 Interrupt handlers execute in Big Endian mode. 1 Interrupt handlers execute in PowerPC Little Endian mode.	Copied to MSR(LE) when an interrupt is taken.
16	EE	External Interrupt Enable 0 Asynchronous exceptions are disabled. 1 Asynchronous exceptions are enabled.	Controls the non-critical external interrupt input, Programmable Interval Timer, and Fixed Interval Timer interrupts.
17	PR	Problem State 0 Supervisor State (all instructions allowed) 1 Problem State (some instructions not allowed)	
18		***Reserved***	
19	ME	Machine Check Enable 0 Machine check exceptions are disabled 1 Machine check exceptions are enabled.	
20:21		***Reserved***	
22	DE	Debug Exception Enable 0 Debug exceptions are disabled. 1 Debug exceptions are enabled.	
23:25		***Reserved***	
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.	If TIE_cpuMmuEn is 0, reading or writing this bit has no effect.

Register 29-20. Machine State Register (MSR) (Continued)

27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.	If TIE_cpuMmuEn is 0, reading or writing this bit has no effect.
28:30		***Reserved***	
31	LE	Little Endian 0 Processor executes in Big Endian mode. 1 Processor executes in PowerPC Little Endian mode.	

PID

See Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13.

Register 29-21. Process ID (PID)

0:23		***Reserved***
24:31		Process ID

PIT

SPR 0x3DB

See also Section 11.7.18.2, “Programmable Interval Timer (PIT),” on page 11-32.

Register 29-22. Programmable Interval Timer (PIT)

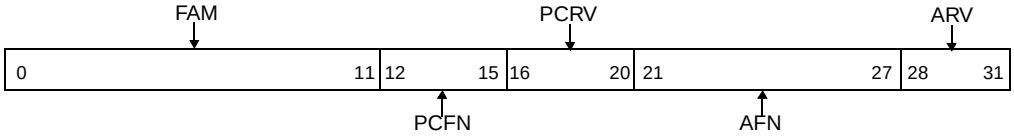
0:31		Programmed interval remaining	Number of clocks remaining until the PIT event

PVR

SPR 0x11F Read-Only

See also Section 24.2.2.5, “Processor Version Register (PVR),” on page 24-6.

Register 29-23. Processor Version Register (PVR)



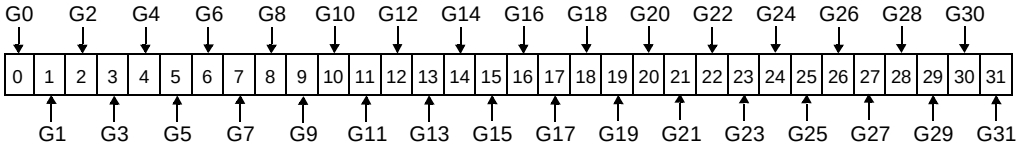
0:11	FAM	Processor Family. Identifies a PowerPC family, such as 4xx or 6xx.	0x002 for the 4xx family.
12:15	PCFN	Processor Core Function. Identifies a specific processor core implementation.	2 for PPC401B2.
16:20	PCRV	Processor Core Revision. Identifies a revision of the processor core defined by the PFN field.	.
21:27	AFN	ASIC Function. An assigned identifier for an ASIC containing a PowerPC 400 Series processor core.	
28:31	ARV	ASIC Revision. An assigned identifier for a revision of the ASIC defined by the AFN field.	

SGR

SPR 0x3B9

See Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13.

Register 29-24. Storage Guarded Register (SGR)



0	G0	0 Normal 1 Guarded	0x0000 0000 – 0x07FF FFFF
1	G1	0 Normal 1 Guarded	0x0800 0000 – 0x0FFF FFFF
2	G2	0 Normal 1 Guarded	0x1000 0000 – 0x17FF FFFF
3	G3	0 Normal 1 Guarded	0x1800 0000 – 0x1FFF FFFF
4	G4	0 Normal 1 Guarded	0x2000 0000 – 0x27FF FFFF
5	G5	0 Normal 1 Guarded	0x2800 0000 – 0x2FFF FFFF
6	G6	0 Normal 1 Guarded	0x3000 0000 – 0x37FF FFFF
7	G7	0 Normal 1 Guarded	0x3800 0000 – 0x3FFF FFFF
8	G8	0 Normal 1 Guarded	0x4000 0000 – 0x47FF FFFF
9	G9	0 Normal 1 Guarded	0x4800 0000 – 0x4FFF FFFF
10	G10	0 Normal 1 Guarded	0x5000 0000 – 0x57FF FFFF
11	G11	0 Normal 1 Guarded	0x5800 0000 – 0x5FFF FFFF
12	G12	0 Normal 1 Guarded	0x6000 0000 – 0x67FF FFFF
13	G13	0 Normal 1 Guarded	0x6800 0000 – 0x6FFF FFFF
14	G14	0 Normal 1 Guarded	0x7000 0000 – 0x77FF FFFF
15	G15	0 Normal 1 Guarded	0x7800 0000 – 0x7FFF FFFF
16	G16	0 Normal 1 Guarded	0x8000 0000 – 0x87FF FFFF
17	G17	0 Normal 1 Guarded	0x8800 0000 – 0x8FFF FFFF

Register 29-24. Storage Guarded Register (SGR) (Continued)

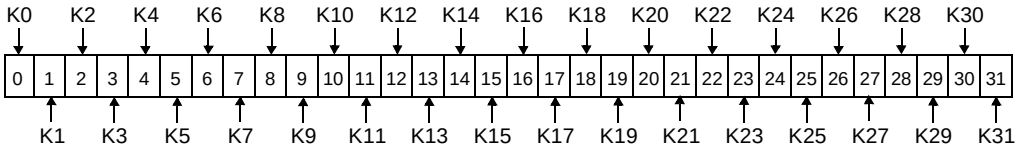
18	G18	0 Normal 1 Guarded	0x9000 0000 – 0x97FF FFFF
19	G19	0 Normal 1 Guarded	0x9800 0000 – 0x9FFF FFFF
20	G20	0 Normal 1 Guarded	0xA000 0000 – 0xA7FF FFFF
21	G21	0 Normal 1 Guarded	0xA800 0000 – 0xAFFF FFFF
22	G22	0 Normal 1 Guarded	0xB000 0000 – 0xB7FF FFFF
23	G23	0 Normal 1 Guarded	0xB800 0000 – 0xBFFF FFFF
24	G24	0 Normal 1 Guarded	0xC000 0000 – 0xC7FF FFFF
25	G25	0 Normal 1 Guarded	0xC800 0000 – 0xCFFF FFFF
26	G26	0 Normal 1 Guarded	0xD000 0000 – 0xD7FF FFFF
27	G27	0 Normal 1 Guarded	0xD800 0000 – 0xDFFF FFFF
28	G28	0 Normal 1 Guarded	0xE000 0000 – 0xE7FF FFFF
29	G29	0 Normal 1 Guarded	0xE800 0000 – 0xEFFF FFFF
30	G30	0 Normal 1 Guarded	0xF000 0000 – 0xF7FF FFFF
31	G31	0 Normal 1 Guarded	0xF800 0000 – 0xFFFF FFFF

SKR

SPR 0x3BC

See Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13.

Register 29-25. Storage Compression Register (SKR)



0	K0	0 Storage compression is off 1 Storage compression is on	0x0000 0000 – 0x07FF FFFF
1	K1	0 Storage compression is off 1 Storage compression is on	0x0800 0000 – 0x0FFF FFFF
2	K2	0 Storage compression is off 1 Storage compression is on	0x1000 0000 – 0x17FF FFFF
3	K3	0 Storage compression is off 1 Storage compression is on	0x1800 0000 – 0x1FFF FFFF
4	K4	0 Storage compression is off 1 Storage compression is on	0x2000 0000 – 0x27FF FFFF
5	K5	0 Storage compression is off 1 Storage compression is on	0x2800 0000 – 0x2FFF FFFF
6	K6	0 Storage compression is off 1 Storage compression is on	0x3000 0000 – 0x37FF FFFF
7	K7	0 Storage compression is off 1 Storage compression is on	0x3800 0000 – 0x3FFF FFFF
8	K8	0 Storage compression is off 1 Storage compression is on	0x4000 0000 – 0x47FF FFFF
9	K9	0 Storage compression is off 1 Storage compression is on	0x4800 0000 – 0x4FFF FFFF
10	K10	0 Storage compression is off 1 Storage compression is on	0x5000 0000 – 0x57FF FFFF
11	K11	0 Storage compression is off 1 Storage compression is on	0x5800 0000 – 0x5FFF FFFF
12	K12	0 Storage compression is off 1 Storage compression is on	0x6000 0000 – 0x67FF FFFF
13	K13	0 Storage compression is off 1 Storage compression is on	0x6800 0000 – 0x6FFF FFFF
14	K14	0 Storage compression is off 1 Storage compression is on	0x7000 0000 – 0x77FF FFFF
15	K15	0 Storage compression is off 1 Storage compression is on	0x7800 0000 – 0x7FFF FFFF
16	K16	0 Storage compression is off 1 Storage compression is on	0x8000 0000 – 0x87FF FFFF
17	K17	0 Storage compression is off 1 Storage compression is on	0x8800 0000 – 0x8FFF FFFF

Register 29-25. Storage Compression Register (SKR) (Continued)

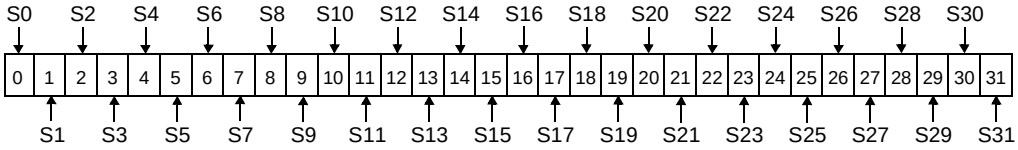
18	K18	0 Storage compression is off 1 Storage compression is on	0x9000 0000 – 0x97FF FFFF
19	K19	0 Storage compression is off 1 Storage compression is on	0x9800 0000 – 0x9FFF FFFF
20	K20	0 Storage compression is off 1 Storage compression is on	0xA000 0000 – 0xA7FF FFFF
21	K21	0 Storage compression is off 1 Storage compression is on	0xA800 0000 – 0xAFFF FFFF
22	K22	0 Storage compression is off 1 Storage compression is on	0xB000 0000 – 0xB7FF FFFF
23	K23	0 Storage compression is off 1 Storage compression is on	0xB800 0000 – 0xBFFF FFFF
24	K24	0 Storage compression is off 1 Storage compression is on	0xC000 0000 – 0xC7FF FFFF
25	K25	0 Storage compression is off 1 Storage compression is on	0xC800 0000 – 0xCFFF FFFF
26	K26	0 Storage compression is off 1 Storage compression is on	0xD000 0000 – 0xD7FF FFFF
27	K27	0 Storage compression is off 1 Storage compression is on	0xD800 0000 – 0xDFFF FFFF
28	K28	0 Storage compression is off 1 Storage compression is on	0xE000 0000 – 0xE7FF FFFF
29	K29	0 Storage compression is off 1 Storage compression is on	0xE800 0000 – 0xEFFF FFFF
30	K30	0 Storage compression is off 1 Storage compression is on	0xF000 0000 – 0xF7FF FFFF
31	K31	0 Storage compression is off 1 Storage compression is on	0xF800 0000 – 0xFFFF FFFF

SLER

SPR 0x3BB

See Section 27.8, “Real-Mode Storage Attribute Control,” on page 27-13.

Register 29-26. Storage Little-Endian Register (SLER)



0	S0	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x0000 0000 – 0x07FF FFFF
1	S1	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x0800 0000 – 0x0FFF FFFF
2	S2	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x1000 0000 – 0x17FF FFFF
3	S3	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x1800 0000 – 0x1FFF FFFF
4	S4	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x2000 0000 – 0x27FF FFFF
5	S5	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x2800 0000 – 0x2FFF FFFF
6	S6	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x3000 0000 – 0x37FF FFFF
7	S7	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x3800 0000 – 0x3FFF FFFF
8	S8	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x4000 0000 – 0x47FF FFFF
9	S9	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x4800 0000 – 0x4FFF FFFF
10	S10	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x5000 0000 – 0x57FF FFFF
11	S11	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x5800 0000 – 0x5FFF FFFF
12	S12	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x6000 0000 – 0x67FF FFFF
13	S13	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x6800 0000 – 0x6FFF FFFF
14	S14	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x7000 0000 – 0x77FF FFFF
15	S15	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x7800 0000 – 0x7FFF FFFF
16	S16	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x8000 0000 – 0x87FF FFFF
17	S17	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x8800 0000 – 0x8FFF FFFF

Register 29-26. Storage Little-Endian Register (SLER) (Continued)

18	S18	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x9000 0000 – 0x97FF FFFF
19	S19	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0x9800 0000 – 0x9FFF FFFF
20	S20	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xA000 0000 – 0xA7FF FFFF
21	S21	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xA800 0000 – 0xAFFF FFFF
22	S22	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xB000 0000 – 0xB7FF FFFF
23	S23	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xB800 0000 – 0xBFFF FFFF
24	S24	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xC000 0000 – 0xC7FF FFFF
25	S25	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xC800 0000 – 0xCFFF FFFF
26	S26	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xD000 0000 – 0xD7FF FFFF
27	S27	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xD800 0000 – 0xDFFF FFFF
28	S28	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xE000 0000 – 0xE7FF FFFF
29	S29	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xE800 0000 – 0xEFFF FFFF
30	S30	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xF000 0000 – 0xF7FF FFFF
31	S31	0 Big Endian or PowerPC Little Endian 1 True Little Endian	0xF800 0000 – 0xFFFF FFFF

SPRG0-SPRG3

SPR 0x110-0x113

See also Section 24.2.2.4, “Special Purpose Register General (SPRG0-SPRG3),” on page 24-6.

Register 29-27. Special Purpose Register General (SPRG0-SPRG3)

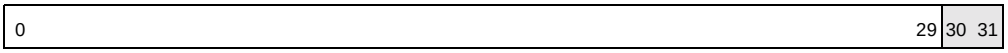
0-31		General data	Privileged user-specified; no hardware usage.

SRR0

SPR 0x01A

See also Section 11.7.3.2, “Save/Restore Registers 0 and 1 (SRR0–SRR1),” on page 11-15.

Register 29-28. Save/Restore Register 0 (SRR0)



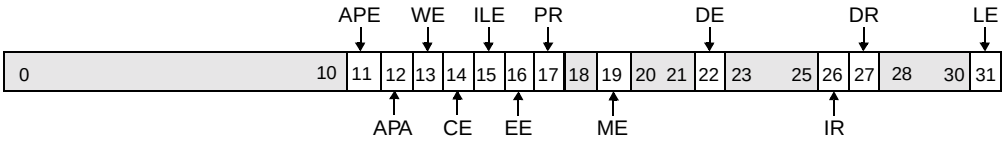
0:29		SRR0 receives an instruction address when a non-critical interrupt is taken; the Program Counter is restored from SRR0 when **rfi** executes.
30:31		***Reserved***

SRR1

SPR 0x01B

See also Section 11.7.3.2, “Save/Restore Registers 0 and 1 (SRR0–SRR1),” on page 11-15.

Register 29-29. Save/Restore Register 1 (SRR1)



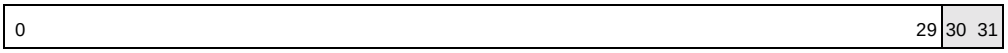
0:31		SRR1 receives a copy of the MSR when a non-critical interrupt is taken; the MSR is restored from SRR1 when **rfi** executes.

SRR2

SPR 0x3DE

See also Section 11.7.3.3, “Save/Restore Registers 2 and 3 (SRR2–SRR3),” on page 11-15.

Register 29-30. Save/Restore Register 2 (SRR2)



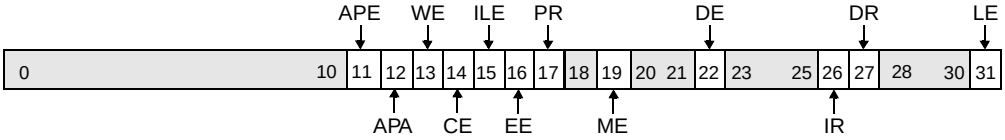
0:29		SRR2 receives an instruction address when a critical interrupt is taken; the Program Counter is restored from SRR2 when **rfci** executes.	
30:31		***Reserved***	

SRR1

SPR 0x3DF

See also Section 11.7.3.3, “Save/Restore Registers 2 and 3 (SRR2–SRR3),” on page 11-15.

Register 29-31. Save/Restore Register 3 (SRR3)



0:31		SRR3 receives a copy of the MSR when a critical interrupt is taken; the MSR is restored from SRR3 when **rfci** executes.

TBHI

SPR 0x3DC

See also Section 11.7.18.1, “Time Base,” on page 11-30.

Register 29-32. Time Base High Register (TBHI)

0:31		Time High	Current count, high-order.

TBHU

SPR 0x3CC Read-Only

See also Section 11.7.18.1, “Time Base,” on page 11-30.

Register 29-33. Time Base High User-mode (TBHU)

0:31		Time High	Current count, high-order. ***Note:*** TBHU is a read-only access vehicle to the time base register TBHI. It is not possible for the contents of TBHU and TBHI to differ.

TBLO

SPR 0x3DD

See also Section 11.7.18.1, “Time Base,” on page 11-30.

Register 29-34. Time Base Low Register (TBLO)

0:31		Time Low	Current count, low-order.

TBLU

SPR 0x3CD Read-Only

See also Section 11.7.18.1, “Time Base,” on page 11-30.

Register 29-35. Time Base Low User-mode (TBLU)

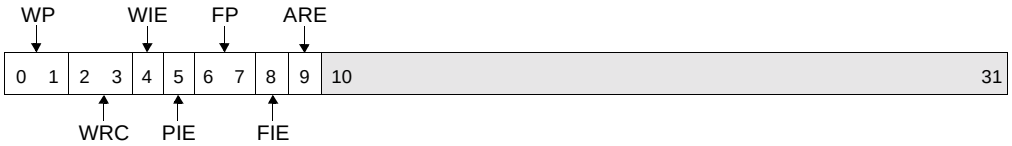
0:31		Time Low	Current count, low-order. ***Note:*** TBLU is a read-only access vehicle to the time base register TBLO. It is not possible for the contents of TBLU and TBLO to differ.

TCR

SPR 0x3DA

See Section 11.7.18.6, “Timer Control Register (TCR),” on page 11-36.

Register 29-36. Timer Control Register (TCR)



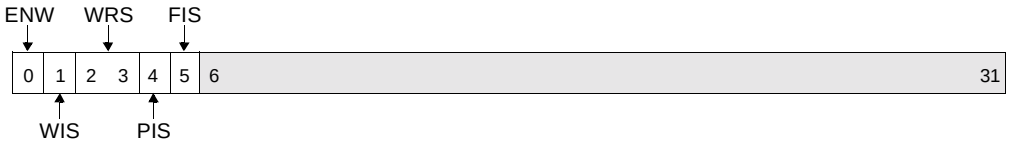
0:1	WP	Watchdog Period 00 2^{17} clocks 01 2^{21} clocks 10 2^{25} clocks 11 2^{29} clocks	
2:3	WRC	Watchdog Reset Control 00 No Watchdog reset occurs. 01 Core reset is forced by the Watchdog. 10 Chip reset is forced by the Watchdog. 11 System reset is forced by the Watchdog.	TCR[WRC] resets to 00. This field can be set by software, but cannot be cleared by software, except by a software-induced reset.
4	WIE	Watchdog Interrupt Enable 0 Disable WDT interrupt. 1 Enable WDT interrupt.	
5	PIE	PIT Interrupt Enable 0 Disable PIT interrupt. 1 Enable PIT interrupt.	
6:7	FP	FIT Period 00 2^{9} clocks 01 2^{13} clocks 10 2^{17} clocks 11 2^{21} clocks	
8	FIE	FIT Interrupt Enable 0 Disable FIT interrupt. 1 Enable FIT interrupt.	
9	ARE	Auto Reload Enable 0 Disable auto reload. 1 Enable auto reload.	Disables on reset.
10:31		***Reserved***	

TSR

SPR 0x3D8 Read/Clear

See Section 11.7.18.5, “Timer Status Register (TSR),” on page 11-35.

Register 29-37. Timer Status Register (TSR)



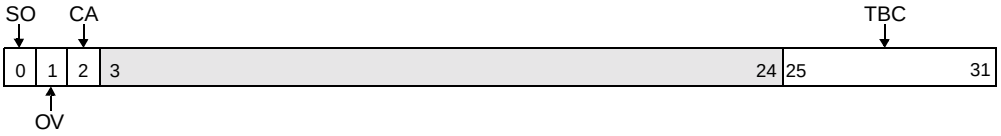
0	ENW	Enable Next Watchdog 0 Action on next Watchdog event is to set TSR[0]. 1 Action on next Watchdog event is governed by TSR[1].	See Section 11.7.18.4, “Watchdog Timer,” on page 11-33.
1	WIS	Watchdog Interrupt Status 0 No Watchdog interrupt is pending. 1 Watchdog interrupt is pending.	
2:3	WRS	Watchdog Reset Status 00 No Watchdog reset has occurred. 01 Core reset was forced by the Watchdog. 10 Chip reset was forced by the Watchdog. 11 System reset was forced by the Watchdog.	
4	PIS	PIT Interrupt Status 0 No PIT interrupt is pending. 1 PIT interrupt is pending.	
5	FIS	FIT Interrupt Status 0 No FIT interrupt is pending. 1 FIT interrupt is pending.	
6:31		***Reserved***	

XER

SPR 0x001

See Section 24.2.2.3, “Fixed Point Exception Register (XER),” on page 24-5.

Register 29-38. Fixed Point Exception Register (XER)



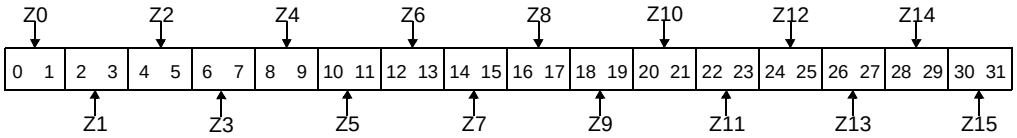
0	SO	Summary Overflow 0 No overflow has occurred. 1 Overflow has occurred.	Can be *set* by **mtspr** or using arithmetic instructions with the “OE” option (see Table 24-2, “XER-Updating Arithmetic Instructions,” on page 24-5); can be *reset* by **mtspr** or by **mcrxr**.
1	OV	Overflow 0 No overflow has occurred. 0 Overflow has occurred.	Can be *set* by **mtspr** or arithmetic instructions with the “OE” option (see Table 24-2, “XER-Updating Arithmetic Instructions,” on page 24-5); can be *reset* by **mtspr**, by **mcrxr**, or by arithmetic instructions with the “OE” option.
2	CA	Carry 0 Carry has not occurred. 1 Carry has occurred.	Can be *set* by **mtspr** or arithmetic instructions that update the CA field (see Table 24-2, “XER-Updating Arithmetic Instructions,” on page 24-5); can be *reset* by **mtspr**, by **mcrxr**, or by arithmetic instructions that update the CA field.
3:24		***Reserved***	
25:31	TBC	Transfer Byte Count	Used by **lswx** and **stswx**; written by **mtspr**

ZPR

SPR 0x3B0

See Section 27.7.1.4, “Zone Protection,” on page 27-10.

Register 29-39. Zone Protection Register (ZPR)



0:1	Z0	TLB page access control for all pages in this zone; TLB_entry[EX, WR] are bits that translate an effective address and PID. In the problem state (MSR[PR] = 1): 00 No access 01 Access controlled by EX and WR 10 Access controlled by EX and WR 11 Accessed as if EX and WR are set In the supervisor state (MSR[PR] = 0): 00 Access controlled by EX and WR 01 Access controlled by EX and WR 10 Access as if EX and WR are set 11 Accessed as if EX and WR are set
2:3	Z1	See the description of Z0.
4:5	Z2	See the description of Z0.
6:7	Z3	See the description of Z0.
8:9	Z4	See the description of Z0.
10:11	Z5	See the description of Z0.
12:13	Z6	See the description of Z0.
14:15	Z7	See the description of Z0.
16:17	Z8	See the description of Z0.
18:19	Z9	See the description of Z0.
20:21	Z10	See the description of Z0.
22:23	Z11	See the description of Z0.
24:25	Z12	See the description of Z0.
26:27	Z13	See the description of Z0.
28:29	Z14	See the description of Z0.
30:31	Z15	See the description of Z0.

A IOP 480 CPU INSTRUCTION SUMMARY

This appendix contains IOP 480 CPU instructions summarized alphabetically and by opcode.

- On page A-1, Section A.1 lists all IOP 480 CPU mnemonics, including extended mnemonics, alphabetically. A short functional description is included for each mnemonic.
- On page A-30, Section A.2 lists all IOP 480 CPU instructions, sorted by primary and secondary opcodes. Extended mnemonics are not included in the opcode list.
- On page A-36, Section A.3 illustrates IOP 480 CPU instruction forms (allowed arrangements of fields within instructions).

A.1 INSTRUCTION SET AND EXTENDED MNEMONICS—ALPHABETICAL

Table A-1 summarizes IOP 480 CPU instruction set, including required extended mnemonics. All mnemonics are listed alphabetically, without regard to whether the mnemonic is realized in hardware or software. When an instruction supports multiple hardware mnemonics (for example, **b**, **ba**, **bl**, **bla** are all forms of **b**), the instruction is alphabetized under the root form. The hardware instructions are described in detail in Section 28, "IOP 480 CPU Instruction Set," which is also alphabetized under the root form. Section 28 also describes the instruction operands and notation.

Note: *The following applies for every Branch Conditional mnemonic:*

Bit 4 of the BO field provides a hint about the most likely outcome of a conditional branch (see Section 24.6.5, "Branch Prediction," on page 24-23 for a full discussion of Branch Prediction). Assemblers should set $BO_4 = 0$ unless a specific reason exists otherwise. In the BO field values specified in the table below, $BO_4 = 0$ has always been assumed. The assembler must allow the programmer to specify Branch Prediction. To do this, the assembler supports a suffix to every conditional branch mnemonic, as follows:

- **+** *Predict branch to be taken.*
- – *Predict branch not to be taken.*

As specific examples, **bc** also could be coded as **bc+** or **bc**–, and **bne** also could be coded **bne+** or **bne**–. These alternate codings set $BO_4 = 1$ only if the requested prediction differs from the Standard Prediction (refer to Section 24.6.5, "Branch Prediction," on page 24-23).

Table A-1. IOP 480 CPU Instruction Syntax Summary

Mnemonic	Operands	Function	Other Registers Changed	Page
add	RT, RA, RB	Add (RA) to (RB). Place result in RT.		28-7
add.			CR[CR0]	
addo			XER[SO, OV]	
addo.			CR[CR0] XER[SO, OV]	
addc	RT, RA, RB	Add (RA) to (RB). Place result in RT. Place carry-out in XER[CA].		28-7
addc.			CR[CR0]	
addco			XER[SO, OV]	
addco.			CR[CR0] XER[SO, OV]	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
adde	RT, RA, RB	Add XER[CA], (RA), (RB). Place result in RT. Place carry-out in XER[CA].		28-9
adde.			CR[CR0]	
addeo			XER[SO, OV]	
addeo.			CR[CR0] XER[SO, OV]	
addi	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT.		28-10
addic	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].		28-11
addic.	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].	CR[CR0]	28-12
addis	RT, RA, IM	Add (IM $\|\|$ $^{16}0$) to (RA|0). Place result in RT.		28-13
addme	RT, RA	Add XER[CA], (RA), (-1). Place result in RT. Place carry-out in XER[CA].		28-14
addme.			CR[CR0]	
addmeo			XER[SO, OV]	
addmeo.			CR[CR0] XER[SO, OV]	
addze	RT, RA	Add XER[CA] to (RA). Place result in RT. Place carry-out in XER[CA].		28-15
addze.			CR[CR0]	
addzeo			XER[SO, OV]	
addzeo.			CR[CR0] XER[SO, OV]	
and	RA, RS, RB	AND (RS) with (RB). Place result in RA.		28-16
and.			CR[CR0]	
andc	RA, RS, RB	AND (RS) with ¬(RB). Place result in RA.		28-17
andc.			CR[CR0]	
andi.	RA, RS, IM	AND (RS) with ($^{16}0$ $\|\|$ IM). Place result in RA.	CR[CR0]	28-18
andis.	RA, RS, IM	AND (RS) with (IM $\|\|$ $^{16}0$). Place result in RA.	CR[CR0]	28-19

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
b	target	Branch unconditional relative. LI ← (target – CIA)$_{6:29}$ NIA ← CIA + EXTS(LI ‖ ^{2}0)		28-20
ba		Branch unconditional absolute. LI ← target$_{6:29}$ NIA ← EXTS(LI ‖ ^{2}0)		
bl		Branch unconditional relative. LI ← (target – CIA)$_{6:29}$ NIA ← CIA + EXTS(LI ‖ ^{2}0)	(LR) ← CIA + 4.	
bla		Branch unconditional absolute. LI ← target$_{6:29}$ NIA ← EXTS(LI ‖ ^{2}0)	(LR) ← CIA + 4.	
bc	BO, BI, target	Branch conditional relative. BD ← (target – CIA)$_{16:29}$ NIA ← CIA + EXTS(BD ‖ ^{2}0)	CTR if BO$_2$ = 0.	28-21
bca		Branch conditional absolute. BD ← target$_{16:29}$ NIA ← EXTS(BD ‖ ^{2}0)	CTR if BO$_2$ = 0.	
bcl		Branch conditional relative. BD ← (target – CIA)$_{16:29}$ NIA ← CIA + EXTS(BD ‖ ^{2}0)	CTR if BO$_2$ = 0. (LR) ← CIA + 4.	
bcla		Branch conditional absolute. BD ← target$_{16:29}$ NIA ← EXTS(BD ‖ ^{2}0)	CTR if BO$_2$ = 0. (LR) ← CIA + 4.	
bcctr	BO, BI	Branch conditional to address in CTR. Using (CTR) at exit from instruction, NIA ← CTR$_{0:29}$ ‖ ^{2}0.	CTR if BO$_2$ = 0.	28-26
bcctrl			CTR if BO$_2$ = 0. (LR) ← CIA + 4.	
bclr	BO, BI	Branch conditional to address in LR. Using (LR) at entry to instruction, NIA ← LR$_{0:29}$ ‖ ^{2}0.	CTR if BO$_2$ = 0.	28-29
bclrl			CTR if BO$_2$ = 0. (LR) ← CIA + 4.	
bctr		Branch unconditionally, to address in CTR. *Extended mnemonic* for **bcctr 20,0**		28-26
bctrl		*Extended mnemonic* for **bcctrl 20,0**	(LR) ← CIA + 4.	
bdnz	target	Decrement CTR. Branch if CTR ≠ 0. *Extended mnemonic* for **bc 16,0,target**		28-21
bdnza		*Extended mnemonic* for **bca 16,0,target**		
bdnzl		*Extended mnemonic* for **bcl 16,0,target**	(LR) ← CIA + 4.	
bdnzla		*Extended mnemonic* for **bcla 16,0,target**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdnzlr		Decrement CTR. Branch if CTR ≠ 0, to address in LR. *Extended mnemonic* for **bclr 16,0**		28-29
bdnzlrl		*Extended mnemonic* for **bclrl 16,0**	(LR) ← CIA + 4.	
bdnzf	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 0,cr_bit,target**		28-21
bdnzfa		*Extended mnemonic* for **bca 0,cr_bit,target**		
bdnzfl		*Extended mnemonic* for **bcl 0,cr_bit,target**	(LR) ← CIA + 4.	
bdnzfla		*Extended mnemonic* for **bcla 0,cr_bit,target**	(LR) ← CIA + 4.	
bdnzflr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0, to address in LR. *Extended mnemonic* for **bclr 0,cr_bit**		28-29
bdnzflrl		*Extended mnemonic* for **bclrl 0,cr_bit**	(LR) ← CIA + 4.	
bdnzt	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 8,cr_bit,target**		28-21
bdnzta		*Extended mnemonic* for **bca 8,cr_bit,target**		
bdnztl		*Extended mnemonic* for **bcl 8,cr_bit,target**	(LR) ← CIA + 4.	
bdnztla		*Extended mnemonic* for **bcla 8,cr_bit,target**	(LR) ← CIA + 4.	
bdnztlr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 8,cr_bit**		28-29
bdnztlrl		*Extended mnemonic* for **bclrl 8,cr_bit**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdz	target	Decrement CTR. Branch if CTR = 0. *Extended mnemonic* for **bc 18,0,target**		28-21
bdza		*Extended mnemonic* for **bca 18,0,target**		
bdzl		*Extended mnemonic* for **bcl 18,0,target**	(LR) ← CIA + 4.	
bdzla		*Extended mnemonic* for **bcla 18,0,target**	(LR) ← CIA + 4.	
bdzlr		Decrement CTR. Branch if CTR = 0, to address in LR. *Extended mnemonic* for **bclr 18,0**		28-29
bdzlrl		*Extended mnemonic* for **bclrl 18,0**	(LR) ← CIA + 4.	
bdzf	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 2,cr_bit,target**		28-21
bdzfa		*Extended mnemonic* for **bca 2,cr_bit,target**		
bdzfl		*Extended mnemonic* for **bcl 2,cr_bit,target**	(LR) ← CIA + 4.	
bdzfla		*Extended mnemonic* for **bcla 2,cr_bit,target**	(LR) ← CIA + 4.	
bdzflr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0 to address in LR. *Extended mnemonic* for **bclr 2,cr_bit**		28-29
bdzflrl		*Extended mnemonic* for **bclrl 2,cr_bit**	(LR) ← CIA + 4.	
bdzt	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 10,cr_bit,target**		28-21
bdzta		*Extended mnemonic* for **bca 10,cr_bit,target**		
bdztl		*Extended mnemonic* for **bcl 10,cr_bit,target**	(LR) ← CIA + 4.	
bdztla		*Extended mnemonic* for **bcla 10,cr_bit,target**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdztlr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 10,cr_bit**		28-29
bdztlrl		*Extended mnemonic* for **bclrl 10,cr_bit**	(LR) ← CIA + 4.	
beq	[cr_field], target	Branch if equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+2,target**		28-21
beqa		*Extended mnemonic* for **bca 12,4∗cr_field+2,target**		
beql		*Extended mnemonic* for **bcl 12,4∗cr_field+2,target**	(LR) ← CIA + 4.	
beqla		*Extended mnemonic* for **bcla 12,4∗cr_field+2,target**	(LR) ← CIA + 4.	
beqctr	[cr_field]	Branch if equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+2**		28-26
beqctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+2**	(LR) ← CIA + 4.	
beqlr	[cr_field]	Branch if equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+2**		28-29
beqlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+2**	(LR) ← CIA + 4.	
bf	cr_bit, target	Branch if CR_{cr_bit} = 0. *Extended mnemonic* for **bc 4,cr_bit,target**		28-21
bfa		*Extended mnemonic* for **bca 4,cr_bit,target**		
bfl		*Extended mnemonic* for **bcl 4,cr_bit,target**	(LR) ← CIA + 4.	
bfla		*Extended mnemonic* for **bcla 4,cr_bit,target**	(LR) ← CIA + 4.	
bfctr	cr_bit	Branch if CR_{cr_bit} = 0, to address in CTR. *Extended mnemonic* for **bcctr 4,cr_bit**		28-26
bfctrl		*Extended mnemonic* for **bcctrl 4,cr_bit**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bflr	cr_bit	Branch if CR_{cr_bit} = 0, to address in LR. *Extended mnemonic* for **bclr 4,cr_bit**		28-29
bflrl		*Extended mnemonic* for **bclrl 4,cr_bit**	(LR) ← CIA + 4.	
bge	[cr_field], target	Branch if greater than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+0,target**		28-21
bgea		*Extended mnemonic* for **bca 4,4∗cr_field+0,target**		
bgel		*Extended mnemonic* for **bcl 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bgela		*Extended mnemonic* for **bcla 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bgectr	[cr_field]	Branch if greater than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+0**		28-26
bgectrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bgelr	[cr_field]	Branch if greater than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+0**		28-29
bgelrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bgt	[cr_field], target	Branch if greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+1,target**		28-21
bgta		*Extended mnemonic* for **bca 12,4∗cr_field+1,target**		
bgtl		*Extended mnemonic* for **bcl 12,4∗cr_field+1,target**	(LR) ← CIA + 4.	
bgtla		*Extended mnemonic* for **bcla 12,4∗cr_field+1,target**	(LR) ← CIA + 4.	
bgtctr	[cr_field]	Branch if greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+1**		28-26
bgtctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+1**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bgtlr	[cr_field]	Branch if greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+1**		28-29
bgtlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+1**	(LR) ← CIA + 4.	
ble	[cr_field], target	Branch if less than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+1,target**		28-21
blea		*Extended mnemonic* for **bca 4,4∗cr_field+1,target**		
blel		*Extended mnemonic* for **bcl 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	
blela		*Extended mnemonic* for **bcla 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	
blectr	[cr_field]	Branch if less than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+1**		28-26
blectrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	
blelr	[cr_field]	Branch if less than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+1**		28-29
blelrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	
blr		Branch unconditionally, to address in LR. *Extended mnemonic* for **bclr 20,0**		28-29
blrl		*Extended mnemonic* for **bclrl 20,0**	(LR) ← CIA + 4.	
blt	[cr_field], target	Branch if less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+0,target**		28-21
blta		*Extended mnemonic* for **bca 12,4∗cr_field+0,target**		
bltl		*Extended mnemonic* for **bcl 12,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bltla		*Extended mnemonic* for **bcla 12,4∗cr_field+0,target**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bltctr	[cr_field]	Branch if less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+0**		28-26
bltctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+0**	(LR) ← CIA + 4.	
bltlr	[cr_field]	Branch if less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+0**		28-29
bltlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+0**	(LR) ← CIA + 4.	
bne	[cr_field], target	Branch if not equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+2,target**		28-21
bnea		*Extended mnemonic* for **bca 4,4∗cr_field+2,target**		
bnel		*Extended mnemonic* for **bcl 4,4∗cr_field+2,target**	(LR) ← CIA + 4.	
bnela		*Extended mnemonic* for **bcla 4,4∗cr_field+2,target**	(LR) ← CIA + 4.	
bnectr	[cr_field]	Branch if not equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+2**		28-26
bnectrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+2**	(LR) ← CIA + 4.	
bnelr	[cr_field]	Branch if not equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+2**		28-29
bnelrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+2**	(LR) ← CIA + 4.	
bng	[cr_field], target	Branch if not greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+1,target**		28-21
bnga		*Extended mnemonic* for **bca 4,4∗cr_field+1,target**		
bngl		*Extended mnemonic* for **bcl 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	
bngla		*Extended mnemonic* for **bcla 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bngctr	[cr_field]	Branch if not greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+1**		28-26
bngctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	
bnglr	[cr_field]	Branch if not greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+1**		28-29
bnglrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	
bnl	[cr_field], target	Branch if not less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+0,target**		28-21
bnla		*Extended mnemonic* for **bca 4,4∗cr_field+0,target**		
bnll		*Extended mnemonic* for **bcl 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bnlla		*Extended mnemonic* for **bcla 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bnlctr	[cr_field]	Branch if not less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+0**		28-26
bnlctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bnllr	[cr_field]	Branch if not less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+0**		28-29
bnllrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bns	[cr_field], target	Branch if not summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+3,target**		28-21
bnsa		*Extended mnemonic* for **bca 4,4∗cr_field+3,target**		
bnsl		*Extended mnemonic* for **bcl 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bnsla		*Extended mnemonic* for **bcla 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bnsctr	[cr_field]	Branch if not summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+3**		28-26
bnsctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	
bnslr	[cr_field]	Branch if not summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+3**		28-29
bnslrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	
bnu	[cr_field], target	Branch if not unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+3,target**		28-21
bnua		*Extended mnemonic* for **bca 4,4∗cr_field+3,target**		
bnul		*Extended mnemonic* for **bcl 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bnula		*Extended mnemonic* for **bcla 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bnuctr	[cr_field]	Branch if not unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+3**		28-26
bnuctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	
bnulr	[cr_field]	Branch if not unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+3**		28-29
bnulrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	
bso	[cr_field], target	Branch if summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+3,target**		28-21
bsoa		*Extended mnemonic* for **bca 12,4∗cr_field+3,target**		
bsol		*Extended mnemonic* for **bcl 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bsola		*Extended mnemonic* for **bcla 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bsoctr	[cr_field]	Branch if summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+3**		28-26
bsoctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	
bsolr	[cr_field]	Branch if summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+3**		28-29
bsolrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	
bt	cr_bit, target	Branch if CR_{cr_bit} = 1. *Extended mnemonic* for **bc 12,cr_bit,target**		28-21
bta		*Extended mnemonic* for **bca 12,cr_bit,target**		
btl		*Extended mnemonic* for **bcl 12,cr_bit,target**	(LR) ← CIA + 4.	
btla		*Extended mnemonic* for **bcla 12,cr_bit,target**	(LR) ← CIA + 4.	
btctr	cr_bit	Branch if CR_{cr_bit} = 1, to address in CTR. *Extended mnemonic* for **bcctr 12,cr_bit**		28-26
btctrl		*Extended mnemonic* for **bcctrl 12,cr_bit**	(LR) ← CIA + 4.	
btlr	cr_bit	Branch if CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 12,cr_bit**		28-29
btlrl		*Extended mnemonic* for **bclrl 12,cr_bit**	(LR) ← CIA + 4.	
bun	[cr_field], target	Branch if unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+3,target**		28-21
buna		*Extended mnemonic* for **bca 12,4∗cr_field+3,target**		
bunl		*Extended mnemonic* for **bcl 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bunla		*Extended mnemonic* for **bcla 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bunctr	[cr_field]	Branch if unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+3**		28-26
bunctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	
bunlr	[cr_field]	Branch if unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+3**		28-29
bunlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	
clrlwi	RA, RS, n	Clear left immediate. (n < 32) $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,n,31**		28-122
clrlwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,n,31**	CR[CR0]	
clrlslwi	RA, RS, b, n	Clear left and shift left immediate. (n ≤ b < 32) $(RA)_{b-n:31-n} \leftarrow (RS)_{b:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ $(RA)_{0:b-n-1} \leftarrow {}^{b-n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,b–n,31–n**		28-122
clrlslwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,b–n,31–n**	CR[CR0]	
clrrwi	RA, RS, n	Clear right immediate. (n < 32) $(RA)_{32-n:31} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,0,31–n**		28-122
clrrwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,0,31–n**	CR[CR0]	
cmp	BF, 0, RA, RB	Compare (RA) to (RB), signed. Results in CR[CRn], where n = BF.		28-33
cmpi	BF, 0, RA, IM	Compare (RA) to EXTS(IM), signed. Results in CR[CRn], where n = BF.		28-34
cmpl	BF, 0, RA, RB	Compare (RA) to (RB), unsigned. Results in CR[CRn], where n = BF.		28-35
cmpli	BF, 0, RA, IM	Compare (RA) to ($^{16}0$ || IM), unsigned. Results in CR[CRn], where n = BF.		28-36
cmplw	[BF,] RA, RB	Compare Logical Lword. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpl BF,0,RA,RB**		28-35

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
cmplwi	[BF,] RA, IM	Compare Logical Lword Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpli BF,0,RA,IM**		28-36
cmpw	[BF,] RA, RB	Compare Lword. Use CR0 if BF is omitted. *Extended mnemonic* for **cmp BF,0,RA,RB**		28-33
cmpwi	[BF,] RA, IM	Compare Lword Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpi BF,0,RA,IM**		28-34
cntlzw	RA, RS	Count leading zeros in RS. Place result in RA.		28-37
cntlzw.			CR[CR0]	
crand	BT, BA, BB	AND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-38
crandc	BT, BA, BB	AND bit (CR_{BA}) with $\neg$(CR_{BB}). Place result in CR_{BT}.		28-39
crclr	bx	Condition register clear. *Extended mnemonic* for **crxor bx,bx,bx**		28-45
creqv	BT, BA, BB	Equivalence of bit CR_{BA} with CR_{BB}. $CR_{BT} \leftarrow \neg(CR_{BA} \oplus CR_{BB})$		28-40
crmove	bx, by	Condition register move. *Extended mnemonic* for **cror bx,by,by**		28-43
crnand	BT, BA, BB	NAND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-38
crnor	BT, BA, BB	NOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-42
crnot	bx, by	Condition register not. *Extended mnemonic* for **crnor bx,by,by**		28-42
cror	BT, BA, BB	OR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-43
crorc	BT, BA, BB	OR bit (CR_{BA}) with $\neg$(CR_{BB}). Place result in CR_{BT}.		28-44
crset	bx	Condition register set. *Extended mnemonic* for **creqv bx,bx,bx**		28-40
crxor	BT, BA, BB	XOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-45
dcba	RA, RB	Speculatively establish the data cache block which contains the effective address (RA|0) + (RB).		28-46
dcbf	RA, RB	Flush (store, then invalidate) the data cache block which contains the effective address (RA|0) + (RB).		28-48

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
dcbi	RA, RB	Invalidate the data cache block which contains the effective address (RA|0) + (RB).		28-49
dcbst	RA, RB	Store the data cache block which contains the effective address (RA|0) + (RB).		28-50
dcbt	RA, RB	Load the data cache block which contains the effective address (RA|0) + (RB).		28-51
dcbtst	RA,RB	Load the data cache block which contains the effective address (RA|0) + (RB).		28-52
dcbz	RA, RB	Zero the data cache block which contains the effective address (RA|0) + (RB).		28-53
dccci	RA, RB	Invalidate the data cache congruence class associated with the effective address (RA|0) + (RB).		28-55
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the effective address (RA|0) + (RB). Place the results in RT.		28-56
divw	RT, RA, RB	Divide (RA) by (RB), signed. Place result in RT.		28-58
divw.			CR[CR0]	
divwo			XER[SO, OV]	
divwo.			CR[CR0] XER[SO, OV]	
divwu	RT, RA, RB	Divide (RA) by (RB), unsigned. Place result in RT.		28-59
divwu.			CR[CR0]	
divwuo			XER[SO, OV]	
divwuo.			CR[CR0] XER[SO, OV]	
eieio		Storage synchronization. All loads and stores that precede the **eieio** instruction complete before any loads and stores that follow the instruction access main storage. Implemented as **sync**, which is more restrictive.		28-60
eqv	RA, RS, RB	Equivalence of (RS) with (RB). (RA) $\leftarrow \neg((RS) \oplus (RB))$		28-61
eqv.			CR[CR0]	
extlwi	RA, RS, n, b	Extract and left justify immediate. (n > 0) $(RA)_{0:n-1} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{n:31} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b,0,n–1**		28-122
extlwi.		*Extended mnemonic* for **rlwinm. RA,RS,b,0,n–1**	CR[CR0]	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
extrwi	RA, RS, n, b	Extract and right justify immediate. (n > 0) $(RA)_{32-n:31} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{0:31-n} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b+n,32–n,31**		28-122
extrwi.		*Extended mnemonic* for **rlwinm. RA,RS,b+n,32–n,31**	CR[CR0]	
extsb	RA, RS	Extend the sign of byte $(RS)_{24:31}$. Place the result in RA.		28-62
extsb.			CR[CR0]	
extsh	RA, RS	Extend the sign of wordword $(RS)_{16:31}$. Place the result in RA.		28-63
extsh.			CR[CR0]	
icbi	RA, RB	Invalidate the instruction cache block which contains the effective address (RA|0) + (RB).		28-64
icbt	RA, RB	Load the instruction cache block which contains the effective address (RA|0) + (RB).		28-65
iccci	RA, RB	Invalidate instruction cache congruence class associated with the effective address (RA|0) + (RB).		28-67
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the effective address (RA|0) + (RB). Place the results in ICDBDR.		28-69
inslwi	RA, RS, n, b	Insert from left immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{0:n-1}$ *Extended mnemonic* for **rlwimi RA,RS,32–b,b,b+n–1**		28-121
inslwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b,b,b+n–1**	CR[CR0]	
insrwi	RA, RS, n, b	Insert from right immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{32-n:31}$ *Extended mnemonic* for **rlwimi RA,RS,32–b–n,b,b+n–1**		28-121
insrwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b–n,b,b+n–1**	CR[CR0]	
isync		Synchronize execution context by flushing the prefetch queue.		28-71
la	RT, D(RA)	Load address. (RA ≠ 0) D is an offset from a base address that is assumed to be (RA). (RT) ← (RA) + EXTS(D) *Extended mnemonic* for **addi RT,RA,D**		28-10
lbz	RT, D(RA)	Load byte from EA = (RA|0) + EXTS(D) and pad left with zeroes, $(RT) \leftarrow {}^{24}0 \;\|\|\; MS(EA,1)$.		28-72

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lbzu	RT, D(RA)	Load byte from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1). Update the base address, (RA) ← EA.		28-73
lbzux	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1). Update the base address, (RA) ← EA.		28-74
lbzx	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		28-75
lha	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)).		28-76
lhau28-80	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		28-77
lhaux	RT, RA, RB	Load word from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		28-78
lhax	RT, RA, RB	Load word from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)).		28-79
lhbrx	RT, RA, RB	Load word from EA = (RA|0) + (RB) then reverse byte order and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA+1,1) || MS(EA,1).		28-80
lhz	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		28-81
lhzu	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		28-82
lhzux	RT, RA, RB	Load word from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		28-83
lhzx	RT, RA, RB	Load word from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		28-84

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
li	RT, IM	Load immediate. (RT) ← EXTS(IM) *Extended mnemonic* for **addi RT,0,value**		28-10
lis	RT, IM	Load immediate shifted. (RT) ← (IM ‖ ^{16}0) *Extended mnemonic* for **addis RT,0,value**		28-13
lmw	RT, D(RA)	Load multiple words starting from EA = (RA|0) + EXTS(D). Place into consecutive registers, RT through GPR(31). RA is not altered unless RA = GPR(31).		28-85
lswi	RT, RA, NB	Load consecutive bytes from EA=(RA|0). Number of bytes n=32 if NB=0, else n=NB. Stack bytes into words in CEIL(n/4) consecutive registers starting with RT, to R_{FINAL} ← ((RT + CEIL(n/4) – 1) % 32). GPR(0) is consecutive to GPR(31). RA is not altered unless RA = R_{FINAL}.		28-86
lswx	RT, RA, RB	Load consecutive bytes from EA=(RA|0)+(RB). Number of bytes n=XER[TBC]. Stack bytes into words in CEIL(n/4) consecutive registers starting with RT, to R_{FINAL} ← ((RT + CEIL(n/4) – 1) % 32). GPR(0) is consecutive to GPR(31). RA is not altered unless RA = R_{FINAL}. RB is not altered unless RB = R_{FINAL}. If n=0, content of RT is undefined.		28-88
lwarx	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Set the Reservation bit.		28-90
lwbrx	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) then reverse byte order, (RT) ← MS(EA+3,1) ‖ MS(EA+2,1) ‖ MS(EA+1,1) ‖ MS(EA,1).		28-91
lwz	RT, D(RA)	Load Lword from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4).		28-92
lwzu	RT, D(RA)	Load Lword from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		28-93
lwzux	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		28-94

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lwzx	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4).		28-95
mcrf	BF, BFA	Move CR field, (CR[CRn]) ← (CR[CRm]) where m ← BFA and n ← BF.		28-96
mcrxr	BF	Move XER[0:3] into field CRn, where n←BF. CR[CRn] ← (XER[SO, OV, CA]). (XER[SO, OV, CA]) ← $^{3}0$.		28-97
mfcr	RT	Move from CR to RT, (RT) ← (CR).		28-98
mfdcr	RT, DCRN	Move from DCR to RT, (RT) ← (DCR(DCRN)).		28-99
mfmsr	RT	Move from MSR to RT, (RT) ← (MSR).		28-100
mfcdbcr mfctr mfdac1 mfdbsr mfdccr mfdcwr mfdear mfesr mfevpr mfiac1 mficcr mficdbdr mflr mfpit mfpvr mfsgr mfsprg0 mfsprg1 mfsprg2 mfsprg3 mfsrr0 mfsrr1 mfsrr2 mfsrr3 mftbhi mftbhu mftblo mftblu mftcr mftsr mfxer	RT	Move from special purpose register (SPR) SPRN. *Extended mnemonic* for **mfspr RT,SPRN** See Table 29-2 on page 29-2 for listing of valid SPRN values.		28-101
mfspr	RT, SPRN	Move from SPR to RT, (RT) ← (SPR(SPRN)).		28-101
mr	RT, RS	Move register. (RT) ← (RS) *Extended mnemonic* for **or RT,RS,RS**		28-115
mr.		*Extended mnemonic* for **or. RT,RS,RS**	CR[CR0]	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mtcr	RS	Move to Condition Register. *Extended mnemonic* for **mtcrf 0xFF,RS**		28-103
mtcrf	FXM, RS	Move some or all of the contents of RS into CR as specified by FXM field, mask $\leftarrow$ $^4(FXM_0) \parallel ^4(FXM_1) \parallel ... \parallel ^4(FXM_6) \parallel ^4(FXM_7)$. (CR)$\leftarrow$((RS) $\wedge$ mask) $\vee$ (CR) $\wedge$ $\neg$mask).		28-103
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) $\leftarrow$ (RS).		28-104
mtmsr	RS	Move to MSR from RS, (MSR) $\leftarrow$ (RS).		28-105
mtcdbcr mtctr mtdac1 mtdbcr mtdbsr mtdccr mtdcwr mtesr mtevpr mtiac1 mticcr mticdbdr mtlr mtpit mtpvr mtsgr mtsprg0 mtsprg1 mtsprg2 mtsprg3 mtsrr0 mtsrr1 mtsrr2 mtsrr3 mttbhi mttblo mttcr mttsr mtxer	RS	Move to SPR SPRN. *Extended mnemonic* for **mtspr SPRN,RS** See Table 29-2 on page 29-2 for listing of valid SPRN values.		28-106
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) $\leftarrow$ (RS).		28-106
mulhw	RT, RA, RB	Multiply (RA) and (RB), signed. Place high-order result in RT. $prod_{0:63}$ $\leftarrow$ (RA) $\times$ (RB) (signed). (RT) $\leftarrow$ $prod_{0:31.}$		28-108
mulhw.			CR[CR0]	
mulhwu	RT, RA, RB	Multiply (RA) and (RB), unsigned. Place high-order result in RT. $prod_{0:63}$ $\leftarrow$ (RA) $\times$ (RB) (unsigned). (RT) $\leftarrow$ $prod_{0:31.}$		28-108
mulhwu.			CR[CR0]	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mulli	RT, RA, IM	Multiply (RA) and IM, signed. Place low-order result in RT. $prod_{0:47} \leftarrow$ (RA) $\times$ IM (signed) (RT) $\leftarrow$ $prod_{16:47}$		28-110
mullw	RT, RA, RB	Multiply (RA) and (RB), signed. Place low-order result in RT. $prod_{0:63} \leftarrow$ (RA) $\times$ (RB) (signed). (RT) $\leftarrow$ $prod_{32:63.}$		28-111
mullw.			CR[CR0]	
mullwo			XER[SO, OV]	
mullwo.			CR[CR0] XER[SO, OV]	
nand	RA, RS, RB	NAND (RS) with (RB). Place result in RA.		28-112
nand.			CR[CR0]	
neg	RT, RA	Negative (*twos* complement) of RA. (RT) $\leftarrow$ $\neg$(RA) + 1		28-113
neg.			CR[CR0]	
nego			XER[SO, OV]	
nego.			CR[CR0] XER[SO, OV]	
nop		Preferred no-op, triggers optimizations based on no-ops. *Extended mnemonic* for **ori 0,0,0**		28-117
nor	RA, RS, RB	NOR (RS) with (RB). Place result in RA.		28-114
nor.			CR[CR0]	
not	RA, RS	Complement register. (RA) $\leftarrow$ $\neg$(RS) *Extended mnemonic* for **nor RA,RS,RS**		28-114
not.		*Extended mnemonic* for **nor. RA,RS,RS**	CR[CR0]	
or	RA, RS, RB	OR (RS) with (RB). Place result in RA.		28-115
or.			CR[CR0]	
orc	RA, RS, RB	OR (RS) with $\neg$(RB). Place result in RA.		28-116
orc.			CR[CR0]	
ori	RA, RS, IM	OR (RS) with ($^{16}0$ $\|\|$ IM). Place result in RA.		28-117
oris	RA, RS, IM	OR (RS) with (IM $\|\|$ $^{16}0$). Place result in RA.		28-118
rfci		Return from critical interrupt (PC) $\leftarrow$ (SRR2). (MSR) $\leftarrow$ (SRR3).		28-119
rfi		Return from interrupt. (PC) $\leftarrow$ (SRR0). (MSR) $\leftarrow$ (SRR1).		28-120

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
rlwimi	RA, RS, SH, MB, ME	Rotate left Lword immediate, then insert according to mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m) ∨ ((RA) ∧ ¬m)		28-121
rlwimi.			CR[CR0]	
rlwinm	RA, RS, SH, MB, ME	Rotate left Lword immediate, then AND with mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m)		28-122
rlwinm.			CR[CR0]	
rlwnm	RA, RS, RB, MB, ME	Rotate left Lword, then AND with mask. r ← ROTL((RS), $(RB)_{27:31}$) m ← MASK(MB, ME) (RA) ← (r ∧ m)		28-124
rlwnm.			CR[CR0]	
rotlw	RA, RS, RB	Rotate left. (RA) ← ROTL((RS), $(RB)_{27:31}$) *Extended mnemonic* for **rlwnm RA,RS,RB,0,31**		28-124
rotlw.		*Extended mnemonic* for **rlwnm. RA,RS,RB,0,31**	CR[CR0]	
rotlwi	RA, RS, n	Rotate left immediate. (RA) ← ROTL((RS), n) *Extended mnemonic* for **rlwinm RA,RS,n,0,31**		28-122
rotlwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31**	CR[CR0]	
rotrwi	RA, RS, n	Rotate right immediate. (RA) ← ROTL((RS), 32−n) *Extended mnemonic* for **rlwinm RA,RS,32−n,0,31**		28-122
rotrwi.		*Extended mnemonic* for **rlwinm. RA,RS,32−n,0,31**	CR[CR0]	
sc		System call exception is generated. (SRR1) ← (MSR) (SRR0) ← (PC) PC ← $EVPR_{0:15}$ || x'0C00' (MSR[WE, PR, EE, PE, DR, IR]) ← 0 (MSR[LE]) ← (MSR[ILE])		28-125
slw	RA, RS, RB	Shift left (RS) by $(RB)_{27:31}$. n ← $(RB)_{27:31.}$ r ← ROTL((RS), n). if $(RB)_{26}$ = 0 then m ← MASK(0, 31 – n) else m ← $^{32}0$. (RA) ← r ∧ m.		28-126
slw.			CR[CR0]	
slwi	RA, RS, n	Shift left immediate. (n < 32) $(RA)_{0:31-n}$ ← $(RS)_{n:31}$ $(RA)_{32-n:31}$ ← $^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,0,31−n**		28-122
slwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31−n**	CR[CR0]	

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
sraw	RA, RS, RB	Shift right algebraic (RS) by $(RB)_{27:31}$. $n \leftarrow (RB)_{27:31}$. $r \leftarrow ROTL((RS), 32 - n)$. if $(RB)_{26} = 0$ then $m \leftarrow MASK(n, 31)$ else $m \leftarrow {}^{32}0$. $s \leftarrow (RS)_{0.}$ $(RA) \leftarrow (r \wedge m) \vee ({}^{32}s \wedge \neg m)$. $XER[CA] \leftarrow s \wedge ((r \wedge \neg m) \neq 0)$.		28-127
sraw.			CR[CR0]	
srawi	RA, RS, SH	Shift right algebraic (RS) by SH. $n \leftarrow SH$. $r \leftarrow ROTL((RS), 32 - n)$. $m \leftarrow MASK(n, 31)$. $s \leftarrow (RS)_{0.}$ $(RA) \leftarrow (r \wedge m) \vee ({}^{32}s \wedge \neg m)$. $XER[CA] \leftarrow s \wedge ((r \wedge \neg m) \neq 0)$.		28-128
srawi.			CR[CR0]	
srw	RA, RS, RB	Shift right (RS) by $(RB)_{27:31}$. $n \leftarrow (RB)_{27:31}$. $r \leftarrow ROTL((RS), 32 - n)$. if $(RB)_{26} = 0$ then $m \leftarrow MASK(n, 31)$ else $m \leftarrow {}^{32}0$. $(RA) \leftarrow r \wedge m$.		28-129
srw.			CR[CR0]	
srwi	RA, RS, n	Shift right immediate. ($n < 32$) $(RA)_{n:31} \leftarrow (RS)_{0:31-n}$ $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,32−n,n,31**		28-122
srwi.		*Extended mnemonic* for **rlwinm. RA,RS,32−n,n,31**	CR[CR0]	
stb	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D).		28-130
stbu	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, $(RA) \leftarrow EA$.		28-131
stbux	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB). Update the base address, $(RA) \leftarrow EA$.		28-132
stbx	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB).		28-133
sth	RS, D(RA)	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D).		28-134
sthbrx	RS, RA, RB	Store word $(RS)_{16:31}$ byte-reversed in memory at EA = (RA|0) + (RB). $MS(EA, 2) \leftarrow (RS)_{24:31} \parallel (RS)_{16:23}$		28-135
sthu	RS, D(RA)	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, $(RA) \leftarrow EA$.		28-136

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
sthux	RS, RA, RB	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		28-137
sthx	RS, RA, RB	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB).		28-138
stmw	RS, D(RA)	Store consecutive words from RS through GPR(31) in memory starting at EA = (RA|0) + EXTS(D).		28-139
stswi	RS, RA, NB	Store consecutive bytes in memory starting at EA=(RA|0). Number of bytes n=32 if NB=0, else n=NB. Bytes are unstacked from CEIL(n/4) consecutive registers starting with RS. GPR(0) is consecutive to GPR(31).		28-140
stswx	RS, RA, RB	Store consecutive bytes in memory starting at EA=(RA|0) + (RB). Number of bytes n=XER[TBC]. Bytes are unstacked from CEIL(n/4) consecutive registers starting with RS. GPR(0) is consecutive to GPR(31).		28-141
stw	RS, D(RA)	Store Lword (RS) in memory at EA = (RA|0) + EXTS(D).		28-143
stwbrx	RS, RA, RB	Store Lword (RS) byte-reversed in memory at EA = (RA|0) + (RB). MS(EA, 4) ← $(RS)_{24:31}$ ‖ $(RS)_{16:23}$ ‖ $(RS)_{8:15}$ ‖ $(RS)_{0:7}$		28-144

stwcx.	RS, RA, RB	Store Lword (RS) in memory at
		EA = (RA|0) + (RB)
		only if reservation bit is set.
		if RESERVE = 1 then
		MS(EA, 4) ← (RS)
		RESERVE ← 0
		(CR[CR0]) ← $^{2}0$ ‖ 1 ‖ XER_{so}
		else
		(CR[CR0]) ← $^{2}0$ ‖ 0 ‖ $XER_{so.}$
				28-145

stwu	RS, D(RA)	Store Lword (RS) in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		28-146
stwux	RS, RA, RB	Store Lword (RS) in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		28-147
stwx	RS, RA, RB	Store Lword (RS) in memory at EA = (RA|0) + (RB).		28-148

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
sub	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. *Extended mnemonic* for **subf RT,RB,RA**		28-149
sub.		*Extended mnemonic* for **subf. RT,RB,RA**	CR[CR0]	
subo		*Extended mnemonic* for **subfo RT,RB,RA**	XER[SO, OV]	
subo.		*Extended mnemonic* for **subfo. RT,RB,RA**	CR[CR0] XER[SO, OV]	
subc	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. Place carry-out in XER[CA]. *Extended mnemonic* for **subfc RT,RB,RA**		28-150
subc.		*Extended mnemonic* for **subfc. RT,RB,RA**	CR[CR0]	
subco		*Extended mnemonic* for **subfco RT,RB,RA**	XER[SO, OV]	
subco.		*Extended mnemonic* for **subfco. RT,RB,RA**	CR[CR0] XER[SO, OV]	
subf	RT, RA, RB	Subtract (RA) from (RB). (RT) ← ¬(RA) + (RB) + 1.		28-149
subf.			CR[CR0]	
subfo			XER[SO, OV]	
subfo.			CR[CR0] XER[SO, OV]	
subfc	RT, RA, RB	Subtract (RA) from (RB). (RT) ← ¬(RA) + (RB) + 1. Place carry-out in XER[CA].		28-150
subfc.			CR[CR0]	
subfco			XER[SO, OV]	
subfco.			CR[CR0] XER[SO, OV]	
subfe	RT, RA, RB	Subtract (RA) from (RB) with carry-in. (RT) ← ¬(RA) + (RB) + XER[CA]. Place carry-out in XER[CA].		28-151
subfe.			CR[CR0]	
subfeo			XER[SO, OV]	
subfeo.			CR[CR0] XER[SO, OV]	
subfic	RT, RA, IM	Subtract (RA) from EXTS(IM). (RT) ← ¬(RA) + EXTS(IM) + 1. Place carry-out in XER[CA].		28-152

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
subfme	RT, RA, RB	Subtract (RA) from (–1) with carry-in. (RT) ← ¬(RA) + (–1) + XER[CA]. Place carry-out in XER[CA].		28-153
subfme.			CR[CR0]	
subfmeo			XER[SO, OV]	
subfmeo.			CR[CR0] XER[SO, OV]	
subfze	RT, RA, RB	Subtract (RA) from zero with carry-in. (RT) ← ¬(RA) + XER[CA]. Place carry-out in XER[CA].		28-154
subfze.			CR[CR0]	
subfzeo			XER[SO, OV]	
subfzeo.			CR[CR0] XER[SO, OV]	
subi	RT, RA, IM	Subtract EXTS(IM) from (RA|0). Place result in RT. *Extended mnemonic* for **addi RT,RA,–IM**		28-10
subic	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic RT,RA,–IM**		28-11
subic.	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic. RT,RA,–IM**	CR[CR0]	28-12
subis	RT, RA, IM	Subtract (IM || ^{16}0) from (RA|0). Place result in RT. *Extended mnemonic* for **addis RT,RA,–IM**		28-13
sync		Synchronization. All instructions that precede **sync** complete before any instructions that follow **sync** begin. When **sync** completes, all storage accesses initiated prior to **sync** are completed.		28-155
tlbia		All of the entries in the TLB are invalidated and become unavailable for translation by clearing the valid (V) bit in the TLBHI portion of each TLB entry. The rest of the fields in the TLB entries are unmodified.		28-156
tlbre	RT, RA,WS	If WS = 0: Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← TLB[(RA)]TID If WS = 1: Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)]		28-157

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbrehi	RT, RA	Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← TLB[(RA)]TID *Extended mnemonic* for tlbre RT,RA,0		28-157
tlbrelo	RT, RA	Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)] *Extended mnemonic* for tlbre RT,RA,1		28-157
tlbsx	RT,RA,RB	Search the TLB array for a valid entry which translates the effective address EA = (RA|0) + (RB). If found, (RT) ← Index of TLB entry. If not found, (RT) Undefined.		28-159
tlbsx.		If found, (RT) ← Index of TLB entry. $CR[CR0]_{EQ}$ ← 1. If not found, (RT) Undefined. $CR[CR0]_{EQ}$ ← 1.	$CR[CR0]_{LT,GT,SO}$	
tlbsync		**tlbsync** does not complete until all previous TLB-update instructions executed by this processor have been received and completed by all other processors. For IOP 480 CPU, **tlbsync** is a no-op.		28-160
tlbwe	RS, RA,WS	If WS = 0: Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. TLBHI[(RA)] ← (RS) TLB[(RA)]TID ← (PID)24:31 If WS = 1: Write TLBLO portion of the selected TLB entry from RS. TLBLO[(RA)] ← (RS)		28-161
tlbwehi	RS, RA	Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. TLBHI[(RA)] ← (RS) TLB[(RA)]TID ← (PID)24:31 *Extended mnemonic* for tlbwe RS,RA,0		28-161

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbwelo	RS, RA	Write TLBLO portion of the selected TLB entry from RS. TLBLO[(RA)] ← (RS) *Extended mnemonic* for tlbwe RS,RA,1		28-161
trap		Trap unconditionally. *Extended mnemonic* for **tw 31,0,0**		
tweq	RA, RB	Trap if (RA) equal to (RB). *Extended mnemonic* for **tw 4,RA,RB**		
twge		Trap if (RA) greater than or equal to (RB). *Extended mnemonic* for **tw 12,RA,RB**		
twgt		Trap if (RA) greater than (RB). *Extended mnemonic* for **tw 8,RA,RB**		
twle		Trap if (RA) less than or equal to (RB). *Extended mnemonic* for **tw 20,RA,RB**		
twlge		Trap if (RA) logically greater than or equal to (RB). *Extended mnemonic* for **tw 5,RA,RB**		
twlgt		Trap if (RA) logically greater than (RB). *Extended mnemonic* for **tw 1,RA,RB**		
twlle		Trap if (RA) logically less than or equal to (RB). *Extended mnemonic* for **tw 6,RA,RB**		
twllt		Trap if (RA) logically less than (RB). *Extended mnemonic* for **tw 2,RA,RB**		
twlng		Trap if (RA) logically not greater than (RB). *Extended mnemonic* for **tw 6,RA,RB**		
twlnl		Trap if (RA) logically not less than (RB). *Extended mnemonic* for **tw 5,RA,RB**		
twlt		Trap if (RA) less than (RB). *Extended mnemonic* for **tw 16,RA,RB**		
twne		Trap if (RA) not equal to (RB). *Extended mnemonic* for **tw 24,RA,RB**		
twng		Trap if (RA) not greater than (RB). *Extended mnemonic* for **tw 20,RA,RB**		
twnl		Trap if (RA) not less than (RB). *Extended mnemonic* for **tw 12,RA,RB**		
tw	TO, RA, RB	Trap exception is generated if, comparing (RA) with (RB), any condition specified by TO is true.		28-163

Table A-1. IOP 480 CPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tweqi	RA, IM	Trap if (RA) equal to EXTS(IM). *Extended mnemonic* for **twi 4,RA,IM**		
twgei		Trap if (RA) greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**		
twgti		Trap if (RA) greater than EXTS(IM). *Extended mnemonic* for **twi 8,RA,IM**		
twlei		Trap if (RA) less than or equal to EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**		
twlgei		Trap if (RA) logically greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**		
twlgti		Trap if (RA) logically greater than EXTS(IM). *Extended mnemonic* for **twi 1,RA,IM**		
twllei		Trap if (RA) logically less than or equal to EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**		
twllti		Trap if (RA) logically less than EXTS(IM). *Extended mnemonic* for **twi 2,RA,IM**		
twlngi		Trap if (RA) logically not greater than EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**		
twlnli		Trap if (RA) logically not less than EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**		
twlti		Trap if (RA) less than EXTS(IM). *Extended mnemonic* for **twi 16,RA,IM**		
twnei		Trap if (RA) not equal to EXTS(IM). *Extended mnemonic* for **twi 24,RA,IM**		
twngi		Trap if (RA) not greater than EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**		
twnli		Trap if (RA) not less than EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**		
twi	TO, RA, IM	Trap exception is generated if, comparing (RA) with EXTS(IM), any condition specified by TO is true.		
wrtee	RS	Write value of RS_{16} to the External Enable bit (MSR[EE]).		28-169
wrteei	E	Write value of E to the External Enable bit (MSR[EE]).		
xor	RA, RS, RB	XOR (RS) with (RB). Place result in RA.		
xor.			CR[CR0]	
xori	RA, RS, IM	XOR (RS) with ($^{16}0$ || IM). Place result in RA.		
xoris	RA, RS, IM	XOR (RS) with (IM || $^{16}0$). Place result in RA.		

A.2 INSTRUCTIONS SORTED BY OPCODE

All instructions are four bytes long and Lword-aligned. All instructions have a primary opcode field (shown as field OPCD in Figure A-1 through Figure A-9 beginning on page A-38) in bits 0:5. Some instructions also have a secondary opcode field (shown as field XO in Figure A-1 through Figure A-9). IOP 480 CPU instructions sorted by primary and secondary opcode may be found in Table A-2.

The “Form” indicated in the table refers to the arrangement of valid field combinations within the four-byte instruction. See Section A.3, “Instruction Formats,” on page A-36 for illustration of the field layouts associated with each form.

Form X has a 10-bit secondary opcode field, while form XO uses only the low-order 9-bits of that field. Form XO uses the high-order secondary opcode bit (the tenth bit) as a variable; therefore, every form XO instruction really consumes two secondary opcodes from the 10-bit secondary-opcode space. The implicitly consumed secondary opcode is listed in parentheses for form XO instructions in Table A-2.

Table A-2. IOP 480 CPU Instructions by Opcode

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
3		D	twi	TO, RA, IM	28-166
7		D	mulli	RT, RA, IM	28-110
8		D	subfic	RT, RA, IM	28-152
10		D	cmpli	BF, 0, RA, IM	28-36
11		D	cmpi	BF, 0, RA, IM	28-34
12		D	addic	RT, RA, IM	28-11
13		D	addic.	RT, RA, IM	28-12
14		D	addi	RT, RA, IM	28-10
15		D	addis	RT, RA, IM	28-13
16		B	bc	BO, BI, target	28-21
			bca		
			bcl		
			bcla		
17		SC	sc		28-125
18		I	b	target	28-20
			ba		
			bl		
			bla		
19	0	XL	mcrf	BF, BFA	28-96
19	16	XL	bclr	BO, BI	28-29
			bclrl		
19	33	XL	crnor	BT, BA, BB	28-42
19	50	XL	rfi		28-120
19	51	XL	rfci		28-119
19	129	XL	crandc	BT, BA, BB	28-39

Table A-2. IOP 480 CPU Instructions by Opcode (Continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
19	150	XL	isync		28-71
19	193	XL	crxor	BT, BA, BB	28-45
19	225	XL	crnand	BT, BA, BB	28-41
19	257	XL	crand	BT, BA, BB	28-38
19	289	XL	creqv	BT, BA, BB	28-40
19	417	XL	crorc	BT, BA, BB	28-44
19	449	XL	cror	BT, BA, BB	28-43
19	528	XL	bcctr	BO, BI	28-26
			bcctrl		
20		M	rlwimi	RA, RS, SH, MB, ME	28-121
			rlwimi.		
21		M	rlwinm	RA, RS, SH, MB, ME	28-122
			rlwinm.		
23		M	rlwnm	RA, RS, RB, MB, ME	28-124
			rlwnm.		
24		D	ori	RA, RS, IM	28-117
25		D	oris	RA, RS, IM	28-118
26		D	xori	RA, RS, IM	28-172
27		D	xoris	RA, RS, IM	28-173
28		D	andi.	RA, RS, IM	28-18
29		D	andis.	RA, RS, IM	28-19
31	0	X	cmp	BF, 0, RA, RB	28-33
31	4	X	tw	TO, RA, RB	28-163
31	8 (520)	XO	subfc	RT, RA, RB	28-150
			subfc.		
			subfco		
			subfco.		
31	10 (522)	XO	addc	RT, RA, RB	28-8
			addc.		
			addco		
			addco.		
31	11 (523)	XO	mulhwu	RT, RA, RB	28-109
			mulhwu.		
31	19	X	mfcr	RT	28-98
31	20	X	lwarx	RT, RA, RB	28-90
31	23	X	lwzx	RT, RA, RB	28-95

Table A-2. IOP 480 CPU Instructions by Opcode (Continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	24	X	slw	RA, RS, RB	28-126
			slw.		
31	26	X	cntlzw	RA, RS	28-37
			cntlzw.		
31	28	X	and	RA, RS, RB	28-16
			and.		
31	32	X	cmpl	BF, 0, RA, RB	28-35
31	40 (552)	XO	subf	RT, RA, RB	28-149
			subf.		
			subfo		
			subfo.		
31	54	X	dcbst	RA, RB	28-50
31	55	X	lwzux	RT, RA, RB	28-94
31	60	X	andc	RA, RS, RB	28-17
			andc.		
31	75 (587)	XO	mulhw	RT, RA, RB	28-108
			mulhw.		
31	83	X	mfmsr	RT	28-100
31	86	X	dcbf	RA, RB	28-48
31	87	X	lbzx	RT, RA, RB	28-75
31	104 (616)	XO	neg	RT, RA	28-113
			neg.		
			nego		
			nego.		
31	119	X	lbzux	RT, RA, RB	28-74
31	124	X	nor	RA, RS, RB	28-114
			nor.		
31	131	X	wrtee	RS	28-169
31	136 (648)	XO	subfe	RT, RA, RB	28-151
			subfe.		
			subfeo		
			subfeo.		
31	138 (650)	XO	adde	RT, RA, RB	28-9
			adde.		
			addeo		
			addeo.		

Table A-2. IOP 480 CPU Instructions by Opcode (Continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	144	XFX	mtcrf	FXM, RS	28-103
31	146	X	mtmsr	RS	28-105
31	150	X	stwcx.	RS, RA, RB	28-145
31	151	X	stwx	RS, RA, RB	28-148
31	163	X	wrteei	E	28-170
31	183	X	stwux	RS, RA, RB	28-147
31	200 (712)	XO	subfze	RT, RA, RB	28-154
			subfze.		
			subfzeo		
			subfzeo.		
31	202 (714)	XO	addze	RT, RA	28-15
			addze.		
			addzeo		
			addzeo.		
31	215	X	stbx	RS, RA, RB	28-133
31	232 (744)	XO	subfme	RT, RA, RB	28-153
			subfme.		
			subfmeo		
			subfmeo.		
31	234 (746)	XO	addme	RT, RA	28-14
			addme.		
			addmeo		
			addmeo.		
31	235 (747)	XO	mullw	RT, RA, RB	28-111
			mullw.		
			mullwo		
			mullwo.		
31	246	X	dcbtst	RA,RB	28-52
31	247	X	stbux	RS, RA, RB	28-132
31	262	X	icbt	RA, RB	28-65
31	266 (778)	XO	add	RT, RA, RB	28-7
			add.		
			addo		
			addo.		
31	278	X	dcbt	RA, RB	28-51
31	279	X	lhzx	RT, RA, RB	28-84

Table A-2. IOP 480 CPU Instructions by Opcode (Continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	284	X	eqv	RA, RS, RB	28-61
			eqv.		
31	311	X	lhzux	RT, RA, RB	28-83
31	316	X	xor	RA, RS, RB	28-171
			xor.		
31	323	XFX	mfdcr	RT, DCRN	28-99
31	339	XFX	mfspr	RT, SPRN	28-101
31	343	X	lhax	RT, RA, RB	28-79
31	370	X	tlbia		28-156
31	375	X	lhaux	RT, RA, RB	28-78
31	407	X	sthx	RS, RA, RB	28-138
31	412	X	orc	RA, RS, RB	28-116
			orc.		
31	439	X	sthux	RS, RA, RB	28-137
31	444	X	or	RA, RS, RB	28-115
			or.		
31	451	XFX	mtdcr	DCRN, RS	28-104
31	454	X	dccci	RA, RB	28-55
31	459 (971)	XO	divwu	RT, RA, RB	28-59
			divwu.		
			divwuo		
			divwuo.		
31	467	XFX	mtspr	SPRN, RS	28-106
31	470	X	dcbi	RA, RB	28-49
31	476	X	nand	RA, RS, RB	28-112
			nand.		
31	486	X	dcread	RT, RA, RB	28-56
31	491 (1003)	XO	divw	RT, RA, RB	28-58
			divw.		
			divwo		
			divwo.		
31	512	X	mcrxr	BF	28-97
31	533	X	lswx	RT, RA, RB	28-88
31	534	X	lwbrx	RT, RA, RB	28-91
31	536	X	srw	RA, RS, RB	28-129
			srw.		

Table A-2. IOP 480 CPU Instructions by Opcode (Continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	566	X	tlbsync		28-160
31	597	X	lswi	RT, RA, NB	28-86
31	598	X	sync		28-155
31	661	X	stswx	RS, RA, RB	28-141
31	662	X	stwbrx	RS, RA, RB	28-144
31	725	X	stswi	RS, RA, NB	28-140
31	790	X	lhbrx	RT, RA, RB	28-80
31	792	X	sraw	RA, RS, RB	28-127
			sraw.		
31	824	X	srawi	RA, RS, SH	28-128
			srawi.		
31	854	X	eieio		28-60
31	914	X	tlbsx	RT,RA,RB	28-159
			tlbsx.		
31	918	X	sthbrx	RS, RA, RB	28-135
31	922	X	extsh	RA, RS	28-63
			extsh.		
31	946	X	tlbre	RT, RA,WS	28-157
31	954	X	extsb	RA, RS	28-62
			extsb.		
31	966	X	iccci	RA, RB	28-67
31	978	X	tlbwe	RS, RA,WS	28-161
31	982	X	icbi	RA, RB	28-64
31	998	X	icread	RA, RB	28-69
31	1014	X	dcbz	RA, RB	28-53
31	TBD	X	dcba	RA, RB	28-46
32		D	lwz	RT, D(RA)	28-92
33		D	lwzu	RT, D(RA)	28-93
34		D	lbz	RT, D(RA)	28-72
35		D	lbzu	RT, D(RA)	28-73
36		D	stw	RS, D(RA)	28-143
37		D	stwu	RS, D(RA)	28-146
38		D	stb	RS, D(RA)	28-130
39		D	stbu	RS, D(RA)	28-131
40		D	lhz	RT, D(RA)	28-81
41		D	lhzu	RT, D(RA)	28-82

Table A-2. IOP 480 CPU Instructions by Opcode (Continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
42		D	lha	RT, D(RA)	28-76
43		D	lhau	RT, D(RA)	28-77
44		D	sth	RS, D(RA)	28-134
45		D	sthu	RS, D(RA)	28-136
46		D	lmw	RT, D(RA)	28-85
47		D	stmw	RS, D(RA)	28-139

A.3 INSTRUCTION FORMATS

Instructions are four bytes long. Instruction addresses are always Lword-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions have an extended opcode in another field. The remaining instruction bits contain additional fields. All instruction fields belong to one of the following categories:

- Defined

  These instructions contain values, such as opcodes, that cannot be altered. The instruction format diagrams specify the values of defined fields.

- Variable

  These fields contain operands, such as general purpose register selectors and immediate values, that may vary from execution to execution. The instruction format diagrams specify the operands in variable fields.

- Reserved

  Bits in a reserved field should be set to 0. In the instruction format diagrams, reserved fields are shaded.

If any bit in a defined field does not contain the expected value, the instruction is illegal and an illegal instruction exception occurs. If any bit in a reserved field does not contain 0, the instruction form is invalid and its result is architecturally undefined. IOP 480 CPU executes all invalid instruction forms without causing an illegal instruction exception.

A.3.1 Instruction Fields

IOP 480 CPU instructions contain various combinations of the following fields, as indicated in the instruction format diagrams. The numbers, enclosed in parentheses, that follow the field names indicate the bit positions; bit fields are indicated by starting and stopping bit positions separated by colons.

AA (30) Absolute address bit.

- 0 The immediate field represents an address relative to the current instruction address (CIA). The effective address (EA) of the branch is either the sum of the LI field sign-extended to 32 bits and the branch instruction address, or the sum of the BD field sign-extended to 32 bits and the branch instruction address.
- 1 The immediate field represents an absolute address. The EA of the branch is either the LI field or the BD field, sign-extended to 32 bits.

BA (11:15) Specifies a bit in the condition register (CR) used as a source of a CR-logical instruction.

BB (16:20) Specifies a bit in the CR used as a source of a CR-logical instruction.

BD (16:29) An immediate field specifying a 14-bit signed *twos* complement branch displacement. This field is concatenated on the right with 0b00 and sign-extended to 32 bits.

BF (6:8) Specifies a field in the CR used as a target in a compare or **mcrf** instruction.

BFA (11:13) Specifies a field in the CR used as a source in a **mcrf** instruction.

BI (11:15) Specifies a bit in the CR used as a source for the condition of a conditional branch instruction.

BO (6:10) Specifies options for conditional branch instructions. See Section 24.6.4, “BO Field on Conditional Branches,” on page 24-22.

BT (6:10) Specifies a bit in the CR used as a target as the result of a CR-Logical instruction.

D (16:31) Specifies a 16-bit signed *twos*-complement integer displacement for load/store instructions.

DCRN (11:20) Specifies a device control register (DCR).

FXM (12:19) Field mask used to identify CR fields to be updated by the **mtcrf** instruction.

IM (16:31) An immediate field used to specify a 16-bit value (either signed integer or unsigned).

LI (6:29) An immediate field specifying a 24-bit signed *twos* complement branch displacement; this field is concatenated on the right with b'00' and sign-extended to 32 bits.

LK (31) Link bit.

0 Do not update the link register (LR).

1 Update the LR with the address of the next instruction.

MB (21:25) Mask begin.

Used in rotate-and-mask instructions to specify the beginning bit of a mask.

ME (26:30) Mask end.

Used in rotate-and-mask instructions to specify the ending bit of a mask.

NB (16:20) Specifies the number of bytes to move in an immediate string load or store.

OPCD (0:5) Primary opcode. Primary opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The OPCD field name does not appear in instruction descriptions.

OE (21) Enables setting the OV and SO fields in the fixed-point exception register (XER) for extended arithmetic.

RA (11:15) A GPR used as a source or target.

RB (16:20) A GPR used as a source.

Rc (31) Record bit.

0 Do not set the CR.

1 Set the CR to reflect the result of an operation.

See Section 24.2.3, “Condition Register (CR),” on page 24-7, for a further discussion of how the CR bits are set.

RS (6:10) A GPR used as a source.

RT (6:10) A GPR used as a target.

SH (16:20) Specifies a shift amount.

SPRF (11:20) Specifies a special purpose register (SPR).

TO (6:10) Specifies the conditions on which to trap, as described under **tw** and **twi** instructions.

XO (21:30) Extended opcode for instructions without an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.

XO (22:30) Extended opcode for instructions with an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.

A.3.2 Instruction Format Diagrams

The “Forms” shown in Figure A-1 through Figure A-9 are valid combinations of instruction fields for IOP 480 CPU. Table A-2 on page A-30 indicates which “Form” is utilized by each IOP 480 CPU opcode. Fields indicated by slashes (/, //, or ///) are reserved. These figures have been adapted from the PowerPC User Instruction Set Architecture.

I-Form

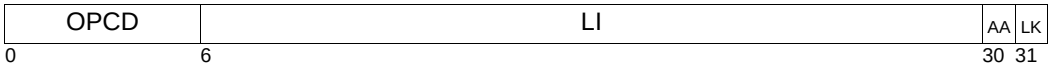


Figure A-1. Instruction Format

B-Form

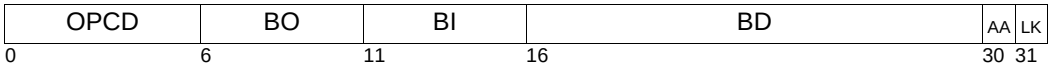


Figure A-2. B Instruction Format

SC-Form

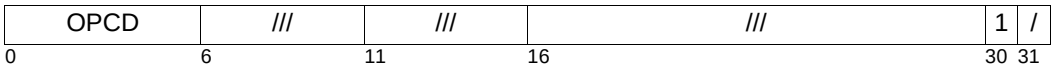


Figure A-3. SC Instruction Format

D-Form

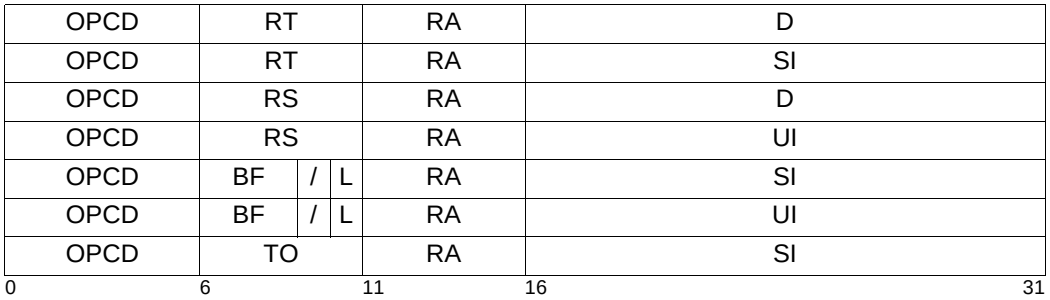


Figure A-4. D Instruction Format

X-Form

OPCD	RT	RA	RB	XO	Rc
OPCD	RT	RA	RB	XO	/
OPCD	RT	RA	NB	XO	/
OPCD	RT	RA	WS	XO	/
OPCD	RT	///	RB	XO	/
OPCD	RT	///	///	XO	/
OPCD	RS	RA	RB	XO	Rc
OPCD	RS	RA	RB	XO	1
OPCD	RS	RA	RB	XO	/
OPCD	RS	RA	NB	XO	/
OPCD	RS	RA	WS	XO	/
OPCD	RS	RA	SH	XO	Rc
OPCD	RS	RA	///	XO	Rc
OPCD	RS	///	RB	XO	/
OPCD	RS	///	///	XO	/
OPCD	BF / L	RA	RB	XO	/
OPCD	BF //	BFA //	///	XO	/
OPCD	BF //	///	U /	XO	Rc
OPCD	BF //	///	///	XO	/
OPCD	TO	RA	RB	XO	/
OPCD	BT	///	///	XO	Rc
OPCD	///	RA	RB	XO	/
OPCD	///	///	RB	XO	/
OPCD	///	///	///	XO	/
OPCD	///	///	E //	XO	/
0	6	11	16	21	31

Figure A-5. X Instruction Format

XL-Form

OPCD	BT		BA		BB	XO	/
OPCD	BO		BI		///	XO	LK
OPCD	BF	//	BFA	//	///	XO	/
OPCD	///		///		///	XO	/
0	6		11		16	21	31

Figure A-6. XL Instruction Format

XFX-Form

OPCD	RT		SPRF		XO	/
OPCD	RT		DCRF		XO	/
OPCD	RT	/	FXM	/	XO	/
OPCD	RS		SPRF		XO	/
OPCD	RS		DCRF		XO	/
0	6	11			21	31

Figure A-7. XFX Instruction Format

XO-Form

OPCD	RT	RA	RB	OE	XO	Rc
OPCD	RT	RA	RB	/	XO	Rc
OPCD	RT	RA	///	OE	XO	Rc
0	6	11	16	21	22	31

Figure A-8. XO Instruction Format

M-Form

OPCD	RS	RA	RB	MB	ME	Rc
OPCD	RS	RA	SH	MB	ME	Rc
0	6	11	16	21	26	31

Figure A-9. M Instruction Format

B IOP 480 CPU INSTRUCTIONS BY CATEGORY

B.1 INSTRUCTION SET SUMMARY CATEGORIES

Section 28, “IOP 480 CPU Instruction Set,” contains detailed descriptions of the instructions, their operands, and notation.

Table B-1 summarizes the instruction categories in the IOP 480 CPU instruction set. The instructions within each category are listed in subsequent tables.

Table B-1. IOP 480 CPU Instruction Set Functional Summary

Storage Reference	load, store
Arithmetic and Logical	add, subtract, negate, multiply, divide, and, andc, or, orc, xor, nand, nor, xnor, sign extension, count leading zeros
Comparison	compare, compare logical, compare immediate
Branch	branch, branch conditional, branch to LR, branch to CTR
CR Logical	crand, crandc, cror, crorc, crnand, crnor, crxor, crxnor, move CR field
Rotate/Shift	rotate and insert, rotate and mask, shift left, shift right
Cache Control	invalidate, touch, zero, flush, store, read
Interrupt Control	write to external interrupt enable bit, move to/from MSR, return from interrupt, return from critical interrupt
Processor Management	system call, synchronize, trap, move to/from DCRs, move to/from SPRs, move to/from CR

B.2 INSTRUCTIONS SPECIFIC TO POWERPC EMBEDDED CONTROLLERS

To meet the functional requirements of processors for embedded systems and real-time applications, the PowerPC-Embedded Controller family defines instructions that are not part of the PowerPC Architecture.

Table B-2 summarizes IOP 480 CPU instructions specific to PowerPC Embedded Controller family.

Table B-2. Instructions Specific to PowerPC-Embedded Controllers

Mnemonic	Operands	Function	Other Registers Changed	Page
dccci	RA, RB	Invalidate the data cache congruence class associated with the effective address (RA|0) + (RB).		28-55
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the effective address (RA|0) + (RB). Place the results in RT.		28-56
icbt	RA, RB	Load the instruction cache block which contains the effective address (RA|0) + (RB).		28-65
iccci	RA, RB	Invalidate instruction cache congruence class associated with the effective address (RA|0) + (RB).		28-67
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the effective address (RA|0) + (RB). Place the results in ICDBDR.		28-69
mfdcr	RT, DCRN	Move from DCR to RT, (RT) ← (DCR(DCRN)).		28-99
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) ← (RS).		28-104
rfci		Return from critical interrupt (PC) ← (SRR2). (MSR) ← (SRR3).		28-119
wrtee	RS	Write value of RS_{16} to MSR[EE].		28-169
wrteei	E	Write value of E to MSR[EE].		28-170
tlbre	RT, RA,WS	If WS = 0: Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← TLB[(RA)]TID If WS = 1: Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)]		28-157

Table B-2. Instructions Specific to PowerPC-Embedded Controllers (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbsx	RT,RA,RB	Search the TLB array for a valid entry which translates the effective address EA = (RA|0) + (RB). If found, (RT) ← Index of TLB entry. If not found, (RT) Undefined.		28-159
tlbsx.		If found, (RT) ← Index of TLB entry. $CR[CR0]_{EQ}$ ← 1. If not found, (RT) Undefined. $CR[CR0]_{EQ}$ ← 1.	$CR[CR0]_{LT,GT,SO}$	
tlbwe	RS, RA,WS	If WS = 0: Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. TLBHI[(RA)] ← (RS) TLB[(RA)]TID ← (PID)24:31 If WS = 1: Write TLBLO portion of the selected TLB entry from RS. TLBLO[(RA)] ← (RS)		28-161
wrtee	RS	Write value of RS_{16} to the External Enable bit (MSR[EE]).		28-169
wrteei	E	Write value of E to the External Enable bit (MSR[EE]).		28-170

B.3 PRIVILEGED INSTRUCTIONS

Table B-3 lists instructions that are under control of the MSR[PR] bit. These instructions are not allowed to be executed when MSR[PR] = 1.

Table B-3. Privileged Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
dcbi	RA, RB	Invalidate the data cache block which contains the effective address (RA|0) + (RB).		28-49
dccci	RA, RB	Invalidate the data cache congruence class associated with the effective address (RA|0) + (RB).		28-55
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the effective address (RA|0) + (RB). Place the results in RT.		28-56
icbt	RA, RB	Load the instruction cache block which contains the effective address (RA|0) + (RB).		28-65
iccci	RA, RB	Invalidate instruction cache congruence class associated with the effective address (RA|0) + (RB).		28-67
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the effective address (RA|0) + (RB). Place the results in ICDBDR.		28-69
mfdcr	RT, DCRN	Move from DCR to RT, (RT) ← (DCR(DCRN)).		28-99
mfmsr	RT	Move from MSR to RT, (RT) ← (MSR).		28-100
mfspr	RT, SPRN	Move from SPR to RT, (RT) ← (SPR(SPRN)). Privileged for all SPRs except LR, CTR, TBHU, TBLU, and XER.		28-101
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) ← (RS).		28-104
mtmsr	RS	Move to MSR from RS, (MSR) ← (RS).		28-105
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) ← (RS). Privileged for all SPRs except LR, CTR, TBHU, TBLU, and XER.		28-106
rfci		Return from critical interrupt (PC) ← (SRR2). (MSR) ← (SRR3).		28-119
rfi		Return from interrupt. (PC) ← (SRR0). (MSR) ← (SRR1).		28-120
tlbia		All of the entries in the TLB are invalidated and become unavailable for translation by clearing the valid (V) bit in the TLBHI portion of each TLB entry. The remaining fields in the TLB entries are unmodified.		28-156

Table B-3. Privileged Instructions (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbre	RT, RA,WS	If WS = 0: Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← TLB[(RA)]TID If WS = 1: Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)]		28-157
tlbsx	RT,RA,RB	Search the TLB array for a valid entry which translates the effective address EA = (RA|0) + (RB). If found, (RT) ← Index of TLB entry. If not found, (RT) Undefined.		28-159
tlbsx.		If found, (RT) ← Index of TLB entry. $CR[CR0]_{EQ}$ ← 1. If not found, (RT) Undefined. $CR[CR0]_{EQ}$ ← 1.	$CR[CR0]_{LT,GT,SO}$	
tlbsync		**tlbsync** does not complete until all previous TLB-update instructions executed by this processor have been received and completed by all other processors. For IOP 480 CPU, **tlbsync** is a no-op.		28-21
tlbwe	RS, RA,WS	If WS = 0: Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. TLBHI[(RA)] ← (RS) TLB[(RA)]TID ← (PID)24:31 If WS = 1: Write TLBLO portion of the selected TLB entry from RS. TLBLO[(RA)] ← (RS)		28-161
wrtee	RS	Write value of RS_{16} to the External Enable bit (MSR[EE]).		28-169
wrteei	E	Write value of E to the External Enable bit (MSR[EE]).		28-170

B.4 ASSEMBLER EXTENDED MNEMONICS

In the appendix “Assembler Extended Mnemonics” of the PowerPC Architecture, it is required that a PowerPC assembler support at least a minimal set of extended mnemonics. These mnemonics encode to the opcodes of other instructions; the only benefit of extended mnemonics is improved usability. Code using extended mnemonics can be easier to write and to understand. Table B-4 lists the extended mnemonics required for IOP 480 CPU.

Note the following for every Branch Conditional mnemonic:

Bit 4 of the BO field provides a hint about the most likely outcome of a conditional branch (see Section 24.6.5, “Branch Prediction,” on page 24-23 for a full discussion of Branch Prediction). Assemblers should set BO_4 = 0 unless a specific reason exists otherwise. In the BO field values specified in the table below, BO_4 = 0 has always been assumed. The assembler must allow the programmer to specify Branch Prediction. To do this, the assembler supports a suffix to every conditional branch mnemonic, as follows:

- **+** Predict branch to be taken.
- – Predict branch not to be taken.

As specific examples, **bc** also could be coded as **bc+** or **bc**–, and **bne** also could be coded **bne+** or **bne**–. These alternate codings set BO_4 = 1 only if the requested prediction differs from the Standard Prediction (see Section 24.6.5, “Branch Prediction,” on page 24-23).

Table B-4. IOP 480 CPU Extended Mnemonics

Mnemonic	Operands	Function	Other Registers Changed	Page
bctr		Branch unconditionally, to address in CTR. *Extended mnemonic* for **bcctr 20,0**		28-26
bctrl		*Extended mnemonic* for **bcctrl 20,0**	(LR) ← CIA + 4.	
bdnz	target	Decrement CTR. Branch if CTR ≠ 0. *Extended mnemonic* for **bc 16,0,target**		28-21
bdnza		*Extended mnemonic* for **bca 16,0,target**		
bdnzl		*Extended mnemonic* for **bcl 16,0,target**	(LR) ← CIA + 4.	
bdnzla		*Extended mnemonic* for **bcla 16,0,target**	(LR) ← CIA + 4.	
bdnzlr		Decrement CTR. Branch if CTR ≠ 0, to address in LR. *Extended mnemonic* for **bclr 16,0**		28-29
bdnzlrl		*Extended mnemonic* for **bclrl 16,0**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdnzf	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 0,cr_bit,target**		28-21
bdnzfa		*Extended mnemonic* for **bca 0,cr_bit,target**		
bdnzfl		*Extended mnemonic* for **bcl 0,cr_bit,target**	(LR) ← CIA + 4.	
bdnzfla		*Extended mnemonic* for **bcla 0,cr_bit,target**	(LR) ← CIA + 4.	
bdnzflr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 0, to address in LR. *Extended mnemonic* for **bclr 0,cr_bit**		28-29
bdnzflrl		*Extended mnemonic* for **bclrl 0,cr_bit**	(LR) ← CIA + 4.	
bdnzt	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 8,cr_bit,target**		28-21
bdnzta		*Extended mnemonic* for **bca 8,cr_bit,target**		
bdnztl		*Extended mnemonic* for **bcl 8,cr_bit,target**	(LR) ← CIA + 4.	
bdnztla		*Extended mnemonic* for **bcla 8,cr_bit,target**	(LR) ← CIA + 4.	
bdnztlr	cr_bit	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 8,cr_bit**		28-29
bdnztlrl		*Extended mnemonic* for **bclrl 8,cr_bit**	(LR) ← CIA + 4.	
bdz	target	Decrement CTR. Branch if CTR = 0. *Extended mnemonic* for **bc 18,0,target**		28-21
bdza		*Extended mnemonic* for **bca 18,0,target**		
bdzl		*Extended mnemonic* for **bcl 18,0,target**	(LR) ← CIA + 4.	
bdzla		*Extended mnemonic* for **bcla 18,0,target**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdzlr		Decrement CTR. Branch if CTR = 0, to address in LR. *Extended mnemonic* for **bclr 18,0**		28-29
bdzlrl		*Extended mnemonic* for **bclrl 18,0**	(LR) ← CIA + 4.	
bdzf	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0. *Extended mnemonic* for **bc 2,cr_bit,target**		28-21
bdzfa		*Extended mnemonic* for **bca 2,cr_bit,target**		
bdzfl		*Extended mnemonic* for **bcl 2,cr_bit,target**	(LR) ← CIA + 4.	
bdzfla		*Extended mnemonic* for **bcla 2,cr_bit,target**	(LR) ← CIA + 4.	
bdzflr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 0 to address in LR. *Extended mnemonic* for **bclr 2,cr_bit**		28-29
bdzflrl		*Extended mnemonic* for **bclrl 2,cr_bit**	(LR) ← CIA + 4.	
bdzt	cr_bit, target	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1. *Extended mnemonic* for **bc 10,cr_bit,target**		28-21
bdzta		*Extended mnemonic* for **bca 10,cr_bit,target**		
bdztl		*Extended mnemonic* for **bcl 10,cr_bit,target**	(LR) ← CIA + 4.	
bdztla		*Extended mnemonic* for **bcla 10,cr_bit,target**	(LR) ← CIA + 4.	
bdztlr	cr_bit	Decrement CTR. Branch if CTR = 0 AND CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 10,cr_bit**		28-29
bdztlrl		*Extended mnemonic* for **bclrl 10,cr_bit**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
beq	[cr_field,] target	Branch if equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+2,target**		28-21
beqa		*Extended mnemonic* for **bca 12,4∗cr_field+2,target**		
beql		*Extended mnemonic* for **bcl 12,4∗cr_field+2,target**	(LR) ← CIA + 4.	
beqla		*Extended mnemonic* for **bcla 12,4∗cr_field+2,target**	(LR) ← CIA + 4.	
beqctr	[cr_field]	Branch if equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+2**		28-26
beqctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+2**	(LR) ← CIA + 4.	
beqlr	[cr_field]	Branch if equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+2**		28-29
beqlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+2**	(LR) ← CIA + 4.	
bf	cr_bit, target	Branch if CR_{cr_bit} = 0. *Extended mnemonic* for **bc 4,cr_bit,target**		28-21
bfa		*Extended mnemonic* for **bca 4,cr_bit,target**		
bfl		*Extended mnemonic* for **bcl 4,cr_bit,target**	(LR) ← CIA + 4.	
bfla		*Extended mnemonic* for **bcla 4,cr_bit,target**	(LR) ← CIA + 4.	
bfctr	cr_bit	Branch if CR_{cr_bit} = 0, to address in CTR. *Extended mnemonic* for **bcctr 4,cr_bit**		28-26
bfctrl		*Extended mnemonic* for **bcctrl 4,cr_bit**	(LR) ← CIA + 4.	
bflr	cr_bit	Branch if CR_{cr_bit} = 0, to address in LR. *Extended mnemonic* for **bclr 4,cr_bit**		28-29
bflrl		*Extended mnemonic* for **bclrl 4,cr_bit**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bge	[cr_field,] target	Branch if greater than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+0,target**		28-21
bgea		*Extended mnemonic* for **bca 4,4∗cr_field+0,target**		
bgel		*Extended mnemonic* for **bcl 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bgela		*Extended mnemonic* for **bcla 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bgectr	[cr_field]	Branch if greater than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+0**		28-26
bgectrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bgelr	[cr_field]	Branch if greater than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+0**		28-29
bgelrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bgt	[cr_field,] target	Branch if greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+1,target**		28-21
bgta		*Extended mnemonic* for **bca 12,4∗cr_field+1,target**		
bgtl		*Extended mnemonic* for **bcl 12,4∗cr_field+1,target**	(LR) ← CIA + 4.	
bgtla		*Extended mnemonic* for **bcla 12,4∗cr_field+1,target**	(LR) ← CIA + 4.	
bgtctr	[cr_field]	Branch if greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+1**		28-26
bgtctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+1**	(LR) ← CIA + 4.	
bgtlr	[cr_field]	Branch if greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+1**		28-29
bgtlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+1**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
ble	[cr_field,] target	Branch if less than or equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+1,target**		28-21
blea		*Extended mnemonic* for **bca 4,4∗cr_field+1,target**		
blel		*Extended mnemonic* for **bcl 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	
blela		*Extended mnemonic* for **bcla 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	
blectr	[cr_field]	Branch if less than or equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+1**		28-26
blectrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	
blelr	[cr_field]	Branch if less than or equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+1**		28-29
blelrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	
blr		Branch unconditionally, to address in LR. *Extended mnemonic* for **bclr 20,0**		28-29
blrl		*Extended mnemonic* for **bclrl 20,0**	(LR) ← CIA + 4.	
blt	[cr_field,] target	Branch if less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+0,target**		28-21
blta		*Extended mnemonic* for **bca 12,4∗cr_field+0,target**		
bltl		*Extended mnemonic* for **bcl 12,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bltla		*Extended mnemonic* for **bcla 12,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bltctr	[cr_field]	Branch if less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+0**		28-26
bltctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+0**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bltlr	[cr_field]	Branch if less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+0**		28-29
bltlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+0**	(LR) ← CIA + 4.	
bne	[cr_field,] target	Branch if not equal. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+2,target**		28-21
bnea		*Extended mnemonic* for **bca 4,4∗cr_field+2,target**		
bnel		*Extended mnemonic* for **bcl 4,4∗cr_field+2,target**	(LR) ← CIA + 4.	
bnela		*Extended mnemonic* for **bcla 4,4∗cr_field+2,target**	(LR) ← CIA + 4.	
bnectr	[cr_field]	Branch if not equal, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+2**		28-26
bnectrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+2**	(LR) ← CIA + 4.	
bnelr	[cr_field]	Branch if not equal, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+2**		28-29
bnelrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+2**	(LR) ← CIA + 4.	
bng	[cr_field,] target	Branch if not greater than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+1,target**		28-21
bnga		*Extended mnemonic* for **bca 4,4∗cr_field+1,target**		
bngl		*Extended mnemonic* for **bcl 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	
bngla		*Extended mnemonic* for **bcla 4,4∗cr_field+1,target**	(LR) ← CIA + 4.	
bngctr	[cr_field]	Branch if not greater than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+1**		28-26
bngctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bnglr	[cr_field]	Branch if not greater than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+1**		28-29
bnglrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	
bnl	[cr_field,] target	Branch if not less than. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+0,target**		28-21
bnla		*Extended mnemonic* for **bca 4,4∗cr_field+0,target**		
bnll		*Extended mnemonic* for **bcl 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bnlla		*Extended mnemonic* for **bcla 4,4∗cr_field+0,target**	(LR) ← CIA + 4.	
bnlctr	[cr_field]	Branch if not less than, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+0**		28-26
bnlctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bnllr	[cr_field]	Branch if not less than, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+0**		28-29
bnllrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	
bns	[cr_field,] target	Branch if not summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+3,target**		28-21
bnsa		*Extended mnemonic* for **bca 4,4∗cr_field+3,target**		
bnsl		*Extended mnemonic* for **bcl 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bnsla		*Extended mnemonic* for **bcla 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bnsctr	[cr_field]	Branch if not summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+3**		28-26
bnsctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bnslr	[cr_field]	Branch if not summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+3**		28-29
bnslrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	
bnu	[cr_field,] target	Branch if not unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 4,4∗cr_field+3,target**		28-21
bnua		*Extended mnemonic* for **bca 4,4∗cr_field+3,target**		
bnul		*Extended mnemonic* for **bcl 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bnula		*Extended mnemonic* for **bcla 4,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bnuctr	[cr_field]	Branch if not unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 4,4∗cr_field+3**		28-26
bnuctrl		*Extended mnemonic* for **bcctrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	
bnulr	[cr_field]	Branch if not unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 4,4∗cr_field+3**		28-29
bnulrl		*Extended mnemonic* for **bclrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	
bso	[cr_field,] target	Branch if summary overflow. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+3,target**		28-21
bsoa		*Extended mnemonic* for **bca 12,4∗cr_field+3,target**		
bsol		*Extended mnemonic* for **bcl 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bsola		*Extended mnemonic* for **bcla 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bsoctr	[cr_field]	Branch if summary overflow, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+3**		28-26
bsoctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bsolr	[cr_field]	Branch if summary overflow, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+3**		28-29
bsolrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	
bt	cr_bit, target	Branch if CR_{cr_bit} = 1. *Extended mnemonic* for **bc 12,cr_bit,target**		28-21
bta		*Extended mnemonic* for **bca 12,cr_bit,target**		
btl		*Extended mnemonic* for **bcl 12,cr_bit,target**	(LR) ← CIA + 4.	
btla		*Extended mnemonic* for **bcla 12,cr_bit,target**	(LR) ← CIA + 4.	
btctr	cr_bit	Branch if CR_{cr_bit} = 1, to address in CTR. *Extended mnemonic* for **bcctr 12,cr_bit**		28-26
btctrl		*Extended mnemonic* for **bcctrl 12,cr_bit**	(LR) ← CIA + 4.	
btlr	cr_bit	Branch if CR_{cr_bit} = 1, to address in LR. *Extended mnemonic* for **bclr 12,cr_bit**		28-29
btlrl		*Extended mnemonic* for **bclrl 12,cr_bit**	(LR) ← CIA + 4.	
bun	[cr_field,] target	Branch if unordered. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bc 12,4∗cr_field+3,target**		28-21
buna		*Extended mnemonic* for **bca 12,4∗cr_field+3,target**		
bunl		*Extended mnemonic* for **bcl 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bunla		*Extended mnemonic* for **bcla 12,4∗cr_field+3,target**	(LR) ← CIA + 4.	
bunctr	[cr_field]	Branch if unordered, to address in CTR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bcctr 12,4∗cr_field+3**		28-26
bunctrl		*Extended mnemonic* for **bcctrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bunlr	[cr_field]	Branch if unordered, to address in LR. Use CR0 if cr_field is omitted. *Extended mnemonic* for **bclr 12,4∗cr_field+3**		28-29
bunlrl		*Extended mnemonic* for **bclrl 12,4∗cr_field+3**	(LR) ← CIA + 4.	
clrlwi	RA, RS, n	Clear left immediate. (n < 32) $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,n,31**		28-122
clrlwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,n,31**	CR[CR0]	
clrlslwi	RA, RS, b, n	Clear left and shift left immediate. (n ≤ b < 32) $(RA)_{b-n:31-n} \leftarrow (RS)_{b:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ $(RA)_{0:b-n-1} \leftarrow {}^{b-n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,b–n,31–n**		28-122
clrlslwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,b–n,31–n**	CR[CR0]	
clrrwi	RA, RS, n	Clear right immediate. (n < 32) $(RA)_{32-n:31} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,0,0,31–n**		28-122
clrrwi.		*Extended mnemonic* for **rlwinm. RA,RS,0,0,31–n**	CR[CR0]	
cmplw	[BF,] RA, RB	Compare Logical Lword. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpl BF,0,RA,RB**		28-35
cmplwi	[BF,] RA, IM	Compare Logical Lword Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpli BF,0,RA,IM**		28-36
cmpw	[BF,] RA, RB	Compare Lword. Use CR0 if BF is omitted. *Extended mnemonic* for **cmp BF,0,RA,RB**		28-33
cmpwi	[BF,] RA, IM	Compare Lword Immediate. Use CR0 if BF is omitted. *Extended mnemonic* for **cmpi BF,0,RA,IM**		28-34
crclr	bx	Condition register clear. *Extended mnemonic* for **crxor bx,bx,bx**		28-45
crmove	bx, by	Condition register move. *Extended mnemonic* for **cror bx,by,by**		28-43

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
crnot	bx, by	Condition register not. *Extended mnemonic* for **crnor bx,by,by**		28-42
crset	bx	Condition register set. *Extended mnemonic* for **creqv bx,bx,bx**		28-40
extlwi	RA, RS, n, b	Extract and left justify immediate. (n > 0) $(RA)_{0:n-1} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{n:31} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b,0,n–1**		28-122
extlwi.		*Extended mnemonic* for **rlwinm. RA,RS,b,0,n–1**	CR[CR0]	
extrwi	RA, RS, n, b	Extract and right justify immediate. (n > 0) $(RA)_{32-n:31} \leftarrow (RS)_{b:b+n-1}$ $(RA)_{0:31-n} \leftarrow {}^{32-n}0$ *Extended mnemonic* for **rlwinm RA,RS,b+n,32–n,31**		28-122
extrwi.		*Extended mnemonic* for **rlwinm. RA,RS,b+n,32–n,31**	CR[CR0]	
inslwi	RA, RS, n, b	Insert from left immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{0:n-1}$ *Extended mnemonic* for **rlwimi RA,RS,32–b,b,b+n–1**		28-121
inslwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b,b,b+n–1**	CR[CR0]	
insrwi	RA, RS, n, b	Insert from right immediate. (n > 0) $(RA)_{b:b+n-1} \leftarrow (RS)_{32-n:31}$ *Extended mnemonic* for **rlwimi RA,RS,32–b–n,b,b+n–1**		28-121
insrwi.		*Extended mnemonic* for **rlwimi. RA,RS,32–b–n,b,b+n–1**	CR[CR0]	
la	RT, D(RA)	Load address. (RA ≠ 0) D is an offset from a base address that is assumed to be (RA). (RT) ← (RA) + EXTS(D) *Extended mnemonic* for **addi RT,RA,D**		28-10
li	RT, IM	Load immediate. (RT) ← EXTS(IM) *Extended mnemonic* for **addi RT,0,value**		28-10
lis	RT, IM	Load immediate shifted. $(RT) \leftarrow (IM \parallel {}^{16}0)$ *Extended mnemonic* for **addis RT,0,value**		28-13

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mfcdbcr **mfctr** **mfdac1** **mfdbsr** **mfdccr** **mfdcwr** **mfdear** **mfesr** **mfevpr** **mfiac1** **mficcr** **mficdbdr** **mflr** **mfpit** **mfpvr** **mfsgr** **mfsprg0** **mfsprg1** **mfsprg2** **mfsprg3** **mfsrr0** **mfsrr1** **mfsrr2** **mfsrr3** **mftbhi** **mftbhu** **mftblo** **mftblu** **mftcr** **mftsr** **mfxer**	RT	Move from special purpose register (SPR) SPRN. *Extended mnemonic* for **mfspr RT,SPRN** See Table 29-2 on page 29-2 for listing of valid SPRN values.		28-101
mr	RT, RS	Move register. (RT) ← (RS) *Extended mnemonic* for **or RT,RS,RS**		28-115
mr.		*Extended mnemonic* for **or. RT,RS,RS**	CR[CR0]	
mtcr	RS	Move to Condition Register. *Extended mnemonic* for **mtcrf 0xFF,RS**		28-103

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mtcdbcr **mtctr** **mtdac1** **mtdbcr** **mtdbsr** **mtdccr** **mtdcwr** **mtesr** **mtevpr** **mtiac1** **mticcr** **mticdbdr** **mtlr** **mtpit** **mtpvr** **mtsgr** **mtsprg0** **mtsprg1** **mtsprg2** **mtsprg3** **mtsrr0** **mtsrr1** **mtsrr2** **mtsrr3** **mttbhi** **mttblo** **mttcr** **mttsr** **mtxer**	RS	Move to SPR SPRN. *Extended mnemonic* for **mtspr SPRN,RS** See Table 29-2 on page 29-2 for listing of valid SPRN values.		28-106
nop		Preferred no-op, triggers optimizations based on no-ops. *Extended mnemonic* for **ori 0,0,0**		28-117
not	RA, RS	Complement register. (RA) ← ¬(RS) *Extended mnemonic* for **nor RA,RS,RS**		28-114
not.		*Extended mnemonic* for **nor. RA,RS,RS**	CR[CR0]	
rotlw	RA, RS, RB	Rotate left. (RA) ← ROTL((RS), $(RB)_{27:31}$) *Extended mnemonic* for **rlwnm RA,RS,RB,0,31**		28-124
rotlw.		*Extended mnemonic* for **rlwnm. RA,RS,RB,0,31**	CR[CR0]	
rotlwi	RA, RS, n	Rotate left immediate. (RA) ← ROTL((RS), n) *Extended mnemonic* for **rlwinm RA,RS,n,0,31**		28-122
rotlwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31**	CR[CR0]	

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
rotrwi	RA, RS, n	Rotate right immediate. (RA) ← ROTL((RS), 32–n) *Extended mnemonic* for **rlwinm RA,RS,32–n,0,31**		28-122
rotrwi.		*Extended mnemonic* for **rlwinm. RA,RS,32–n,0,31**	CR[CR0]	
slwi	RA, RS, n	Shift left immediate. (n < 32) $(RA)_{0:31-n} \leftarrow (RS)_{n:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,n,0,31–n**		28-122
slwi.		*Extended mnemonic* for **rlwinm. RA,RS,n,0,31–n**	CR[CR0]	
srwi	RA, RS, n	Shift right immediate. (n < 32) $(RA)_{n:31} \leftarrow (RS)_{0:31-n}$ $(RA)_{0:n-1} \leftarrow {}^{n}0$ *Extended mnemonic* for **rlwinm RA,RS,32–n,n,31**		28-122
srwi.		*Extended mnemonic* for **rlwinm. RA,RS,32–n,n,31**	CR[CR0]	
sub	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. *Extended mnemonic* for **subf RT,RB,RA**		28-149
sub.		*Extended mnemonic* for **subf. RT,RB,RA**	CR[CR0]	
subo		*Extended mnemonic* for **subfo RT,RB,RA**	XER[SO, OV]	
subo.		*Extended mnemonic* for **subfo. RT,RB,RA**	CR[CR0] XER[SO, OV]	
subc	RT, RA, RB	Subtract (RB) from (RA). (RT) ← ¬(RB) + (RA) + 1. Place carry-out in XER[CA]. *Extended mnemonic* for **subfc RT,RB,RA**		28-150
subc.		*Extended mnemonic* for **subfc. RT,RB,RA**	CR[CR0]	
subco		*Extended mnemonic* for **subfco RT,RB,RA**	XER[SO, OV]	
subco.		*Extended mnemonic* for **subfco. RT,RB,RA**	CR[CR0] XER[SO, OV]	
subi	RT, RA, IM	Subtract EXTS(IM) from (RA|0). Place result in RT. *Extended mnemonic* for **addi RT,RA,–IM**		28-10

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
subic	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic RT,RA,–IM**		28-11
subic.	RT, RA, IM	Subtract EXTS(IM) from (RA). Place result in RT. Place carry-out in XER[CA]. *Extended mnemonic* for **addic. RT,RA,–IM**	CR[CR0]	28-12
subis	RT, RA, IM	Subtract (IM $\|\|$ $^{16}0$) from (RA|0). Place result in RT. *Extended mnemonic* for **addis RT,RA,–IM**		28-13
tlbrehi	RT, RA	Load TLBHI portion of the selected TLB entry into RT. Load the PID register with the contents of the TID field of the selected TLB entry. (RT) ← TLBHI[(RA)] (PID) ← TLB[(RA)]TID *Extended mnemonic* for tlbre RT,RA,0		28-157
tlbrelo	RT, RA	Load TLBLO portion of the selected TLB entry into RT. (RT) ← TLBLO[(RA)] *Extended mnemonic* for tlbre RT,RA,1		28-157
tlbwehi	RS, RA	Write TLBHI portion of the selected TLB entry from RS. Write the TID field of the selected TLB entry from the PID register. TLBHI[(RA)] ← (RS) TLB[(RA)]TID ← (PID)24:31 *Extended mnemonic* for tlbwe RS,RA,0		28-161
tlbwelo	RS, RA	Write TLBLO portion of the selected TLB entry from RS. TLBLO[(RA)] ← (RS) *Extended mnemonic* for tlbwe RS,RA,1		28-161

Table B-4. IOP 480 CPU Extended Mnemonics (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tweqi	RA, IM	Trap if (RA) equal to EXTS(IM). *Extended mnemonic* for **twi 4,RA,IM**		
twgei		Trap if (RA) greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**		
twgti		Trap if (RA) greater than EXTS(IM). *Extended mnemonic* for **twi 8,RA,IM**		
twlei		Trap if (RA) less than or equal to EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**		
twlgei		Trap if (RA) logically greater than or equal to EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**		
twlgti		Trap if (RA) logically greater than EXTS(IM). *Extended mnemonic* for **twi 1,RA,IM**		
twllei		Trap if (RA) logically less than or equal to EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**		
twllti		Trap if (RA) logically less than EXTS(IM). *Extended mnemonic* for **twi 2,RA,IM**		
twlngi		Trap if (RA) logically not greater than EXTS(IM). *Extended mnemonic* for **twi 6,RA,IM**		
twlnli		Trap if (RA) logically not less than EXTS(IM). *Extended mnemonic* for **twi 5,RA,IM**		
twlti		Trap if (RA) less than EXTS(IM). *Extended mnemonic* for **twi 16,RA,IM**		
twnei		Trap if (RA) not equal to EXTS(IM). *Extended mnemonic* for **twi 24,RA,IM**		
twngi		Trap if (RA) not greater than EXTS(IM). *Extended mnemonic* for **twi 20,RA,IM**		
twnli		Trap if (RA) not less than EXTS(IM). *Extended mnemonic* for **twi 12,RA,IM**		

B.5 STORAGE REFERENCE INSTRUCTIONS

IOP 480 CPU uses load and store instructions to transfer data between memory and the general purpose registers. Load and store instructions operate on byte, word and Lword data. The Storage Reference instructions also support loading or storing multiple registers, character strings, and byte-reversed data. Table B-5 shows the Storage Reference instructions available for use in IOP 480 CPU.

Table B-5. Storage Reference Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
lbz	RT, D(RA)	Load byte from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		28-72
lbzu	RT, D(RA)	Load byte from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1). Update the base address, (RA) ← EA.		28-73
lbzux	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1). Update the base address, (RA) ← EA.		28-74
lbzx	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		28-75
lha	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)).		28-76
lhau	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		28-77
lhaux	RT, RA, RB	Load word from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		28-78
lhax	RT, RA, RB	Load word from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)).		28-79
lhbrx	RT, RA, RB	Load word from EA = (RA|0) + (RB) then reverse byte order and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA+1,1) || MS(EA,1).		28-80
lhz	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		28-81
lhzu	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		28-82

Table B-5. Storage Reference Instructions (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lhzux	RT, RA, RB	Load word from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		28-83
lhzx	RT, RA, RB	Load word from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		28-84
lmw	RT, D(RA)	Load multiple words starting from EA = (RA|0) + EXTS(D). Place into consecutive registers, RT through GPR(31). RA is not altered unless RA = GPR(31).		28-85
lswi	RT, RA, NB	Load consecutive bytes from EA=(RA|0). Number of bytes n=32 if NB=0, else n=NB. Stack bytes into words in CEIL(n/4) consecutive registers starting with RT, to R_{FINAL} ← ((RT + CEIL(n/4) – 1) % 32). GPR(0) is consecutive to GPR(31). RA is not altered unless RA = R_{FINAL}.		28-86
lswx	RT, RA, RB	Load consecutive bytes from EA=(RA|0)+(RB). Number of bytes n=XER[TBC]. Stack bytes into words in CEIL(n/4) consecutive registers starting with RT, to R_{FINAL} ← ((RT + CEIL(n/4) – 1) % 32). GPR(0) is consecutive to GPR(31). RA is not altered unless RA = R_{FINAL}. RB is not altered unless RB = R_{FINAL}. If n=0, content of RT is undefined.		28-88
lwarx	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Set the Reservation bit.		28-90
lwbrx	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) then reverse byte order, (RT) ← MS(EA+3,1) || MS(EA+2,1) || MS(EA+1,1) || MS(EA,1).		28-91
lwz	RT, D(RA)	Load Lword from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4).		28-92
lwzu	RT, D(RA)	Load Lword from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		28-93
lwzux	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		28-94
lwzx	RT, RA, RB	Load Lword from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4).		28-95

Table B-5. Storage Reference Instructions (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
stb	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D).		28-130
stbu	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		28-131
stbux	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		28-132
stbx	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB).		28-133
sth	RS, D(RA)	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D).		28-134
sthbrx	RS, RA, RB	Store word $(RS)_{16:31}$ byte-reversed in memory at EA = (RA|0) + (RB). MS(EA, 2) ← $(RS)_{24:31} \parallel (RS)_{16:23}$		28-135
sthu	RS, D(RA)	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		28-136
sthux	RS, RA, RB	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		28-137
sthx	RS, RA, RB	Store word $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB).		28-138
stmw	RS, D(RA)	Store consecutive words from RS through GPR(31) in memory starting at EA = (RA|0) + EXTS(D).		28-139
stswi	RS, RA, NB	Store consecutive bytes in memory starting at EA=(RA|0). Number of bytes n=32 if NB=0, else n=NB. Bytes are unstacked from CEIL(n/4) consecutive registers starting with RS. GPR(0) is consecutive to GPR(31).		28-140
stswx	RS, RA, RB	Store consecutive bytes in memory starting at EA=(RA|0)+(RB). Number of bytes n=XER[TBC]. Bytes are unstacked from CEIL(n/4) consecutive registers starting with RS. GPR(0) is consecutive to GPR(31).		28-141
stw	RS, D(RA)	Store Lword (RS) in memory at EA = (RA|0) + EXTS(D).		28-143
stwbrx	RS, RA, RB	Store Lword (RS) byte-reversed in memory at EA = (RA|0) + (RB). MS(EA, 4) ← $(RS)_{24:31} \parallel (RS)_{16:23} \parallel (RS)_{8:15} \parallel (RS)_{0:7}$		28-144

Table B-5. Storage Reference Instructions (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page

stwcx.	RS, RA, RB	Store Lword (RS) in memory at
		EA = (RA|0) + (RB)
		only if reservation bit is set.
		if RESERVE = 1 then
		MS(EA, 4) ← (RS)
		RESERVE ← 0
		(CR[CR0]) ← $^{2}0 \parallel 1 \parallel XER_{so}$
		else
		(CR[CR0]) ← $^{2}0 \parallel 0 \parallel XER_{so.}$
				28-145

stwu	RS, D(RA)	Store Lword (RS) in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		28-146
stwux	RS, RA, RB	Store Lword (RS) in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		28-147
stwx	RS, RA, RB	Store Lword (RS) in memory at EA = (RA|0) + (RB).		28-148

B.6 ARITHMETIC AND LOGICAL INSTRUCTIONS

Table B-6 shows the set of arithmetic and logical instructions supported by IOP 480 CPU. Arithmetic operations are performed on integer or ordinal operands stored in registers. Instructions using two operands are defined in a three operand format where the operation is performed on the operands stored in two registers and the result is placed in a third register. Instructions using one operand are defined in a two operand format where the operation is performed on the operand in one register and the result is placed in another register. Several instructions also have immediate formats in which one operand is coded as part of the instruction itself. Most arithmetic and logical instructions can optionally set the condition code register based on the outcome of the instruction.

Table B-6. Arithmetic and Logical Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
add	RT, RA, RB	Add (RA) to (RB). Place result in RT.		28-7
add.			CR[CR0]	
addo			XER[SO, OV]	
addo.			CR[CR0] XER[SO, OV]	
addc	RT, RA, RB	Add (RA) to (RB). Place result in RT. Place carry-out in XER[CA].		28-8
addc.			CR[CR0]	
addco			XER[SO, OV]	
addco.			CR[CR0] XER[SO, OV]	
adde	RT, RA, RB	Add XER[CA], (RA), (RB). Place result in RT. Place carry-out in XER[CA].		28-9
adde.			CR[CR0]	
addeo			XER[SO, OV]	
addeo.			CR[CR0] XER[SO, OV]	
addi	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT.		28-10
addic	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].		28-11
addic.	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].	CR[CR0]	28-12
addis	RT, RA, IM	Add (IM $\|\|$ $^{16}0$) to (RA|0). Place result in RT.		28-13
addme	RT, RA	Add XER[CA], (RA), (-1). Place result in RT. Place carry-out in XER[CA].		28-14
addme.			CR[CR0]	
addmeo			XER[SO, OV]	
addmeo.			CR[CR0] XER[SO, OV]	

Table B-6. Arithmetic and Logical Instructions (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
addze	RT, RA	Add XER[CA] to (RA). Place result in RT. Place carry-out in XER[CA].		28-15
addze.			CR[CR0]	
addzeo			XER[SO, OV]	
addzeo.			CR[CR0] XER[SO, OV]	
and	RA, RS, RB	AND (RS) with (RB). Place result in RA.		28-16
and.			CR[CR0]	
andc	RA, RS, RB	AND (RS) with ¬(RB). Place result in RA.		28-17
andc.			CR[CR0]	
andi.	RA, RS, IM	AND (RS) with ($^{16}0$ ‖ IM). Place result in RA.	CR[CR0]	28-18
andis.	RA, RS, IM	AND (RS) with (IM ‖ $^{16}0$). Place result in RA.	CR[CR0]	28-19
cntlzw	RA, RS	Count leading zeros in RS. Place result in RA.		28-37
cntlzw.			CR[CR0]	
divw	RT, RA, RB	Divide (RA) by (RB), signed. Place result in RT.		28-58
divw.			CR[CR0]	
divwo			XER[SO, OV]	
divwo.			CR[CR0] XER[SO, OV]	
divwu	RT, RA, RB	Divide (RA) by (RB), unsigned. Place result in RT.		28-59
divwu.			CR[CR0]	
divwuo			XER[SO, OV]	
divwuo.			CR[CR0] XER[SO, OV]	
eqv	RA, RS, RB	Equivalence of (RS) with (RB). (RA) ← ¬((RS) ⊕ (RB))		28-61
eqv.			CR[CR0]	
extsb	RA, RS	Extend the sign of byte $(RS)_{24:31}$. Place the result in RA.		28-62
extsb.			CR[CR0]	
extsh	RA, RS	Extend the sign of word $(RS)_{16:31}$. Place the result in RA.		28-63
extsh.			CR[CR0]	
mulhw	RT, RA, RB	Multiply (RA) and (RB), signed. Place hi-order result in RT. $prod_{0:63}$ ← (RA) × (RB) (signed). (RT) ← $prod_{0:31.}$		28-108
mulhw.			CR[CR0]	
mulhwu	RT, RA, RB	Multiply (RA) and (RB), unsigned. Place hi-order result in RT. $prod_{0:63}$ ← (RA) × (RB) (unsigned). (RT) ← $prod_{0:31.}$		28-109
mulhwu.			CR[CR0]	

Table B-6. Arithmetic and Logical Instructions (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mulli	RT, RA, IM	Multiply (RA) and IM, signed. Place lo-order result in RT. $prod_{0:47} \leftarrow$ (RA) $\times$ IM (signed) (RT) $\leftarrow$ $prod_{16:47}$		28-110
mullw	RT, RA, RB	Multiply (RA) and (RB), signed. Place lo-order result in RT. $prod_{0:63} \leftarrow$ (RA) $\times$ (RB) (signed). (RT) $\leftarrow$ $prod_{32:63.}$		28-111
mullw.			CR[CR0]	
mullwo			XER[SO, OV]	
mullwo.			CR[CR0] XER[SO, OV]	
nand	RA, RS, RB	NAND (RS) with (RB). Place result in RA.		28-112
nand.			CR[CR0]	
neg	RT, RA	Negative (two's complement) of RA. (RT) $\leftarrow$ $\neg$(RA) + 1		28-113
neg.			CR[CR0]	
nego			XER[SO, OV]	
nego.			CR[CR0] XER[SO, OV]	
nor	RA, RS, RB	NOR (RS) with (RB). Place result in RA.		28-114
nor.			CR[CR0]	
or	RA, RS, RB	OR (RS) with (RB). Place result in RA.		28-115
or.			CR[CR0]	
orc	RA, RS, RB	OR (RS) with $\neg$(RB). Place result in RA.		28-116
orc.			CR[CR0]	
ori	RA, RS, IM	OR (RS) with ($^{16}0$ $\|\|$ IM). Place result in RA.		28-117
oris	RA, RS, IM	OR (RS) with (IM $\|\|$ $^{16}0$). Place result in RA.		28-118
subf	RT, RA, RB	Subtract (RA) from (RB). (RT) $\leftarrow$ $\neg$(RA) + (RB) + 1.		28-149
subf.			CR[CR0]	
subfo			XER[SO, OV]	
subfo.			CR[CR0] XER[SO, OV]	
subfc	RT, RA, RB	Subtract (RA) from (RB). (RT) $\leftarrow$ $\neg$(RA) + (RB) + 1. Place carry-out in XER[CA].		28-150
subfc.			CR[CR0]	
subfco			XER[SO, OV]	
subfco.			CR[CR0] XER[SO, OV]	

Table B-6. Arithmetic and Logical Instructions (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
subfe	RT, RA, RB	Subtract (RA) from (RB) with carry-in. (RT) ← ¬(RA) + (RB) + XER[CA]. Place carry-out in XER[CA].		28-151
subfe.			CR[CR0]	
subfeo			XER[SO, OV]	
subfeo.			CR[CR0] XER[SO, OV]	
subfic	RT, RA, IM	Subtract (RA) from EXTS(IM). (RT) ← ¬(RA) + EXTS(IM) + 1. Place carry-out in XER[CA].		28-152
subfme	RT, RA, RB	Subtract (RA) from (–1) with carry-in. (RT) ← ¬(RA) + (–1) + XER[CA]. Place carry-out in XER[CA].		28-153
subfme.			CR[CR0]	
subfmeo			XER[SO, OV]	
subfmeo.			CR[CR0] XER[SO, OV]	
subfze	RT, RA, RB	Subtract (RA) from zero with carry-in. (RT) ← ¬(RA) + XER[CA]. Place carry-out in XER[CA].		28-154
subfze.			CR[CR0]	
subfzeo			XER[SO, OV]	
subfzeo.			CR[CR0] XER[SO, OV]	
xor	RA, RS, RB	XOR (RS) with (RB). Place result in RA.		28-171
xor.			CR[CR0]	
xori	RA, RS, IM	XOR (RS) with ($^{16}0$ ‖ IM). Place result in RA.		28-172
xoris	RA, RS, IM	XOR (RS) with (IM ‖ $^{16}0$). Place result in RA.		28-173

B.7 CONDITION REGISTER LOGICAL INSTRUCTIONS

Condition Register (CR) logical instructions allow the user to combine the results of several comparisons without incurring the overhead of conditional branching. These instructions can significantly improve code performance if multiple conditions are tested prior to making a branch decision. Table B-7 summarizes the CR logical instructions.

Table B-7. Condition Register Logical Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
crand	BT, BA, BB	AND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-38
crandc	BT, BA, BB	AND bit (CR_{BA}) with ¬(CR_{BB}). Place result in CR_{BT}.		28-39
creqv	BT, BA, BB	Equivalence of bit CR_{BA} with CR_{BB}. $CR_{BT} \leftarrow \neg(CR_{BA} \oplus CR_{BB})$		28-40
crnand	BT, BA, BB	NAND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-41
crnor	BT, BA, BB	NOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-42
cror	BT, BA, BB	OR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-43
crorc	BT, BA, BB	OR bit (CR_{BA}) with ¬(CR_{BB}). Place result in CR_{BT}.		28-44
crxor	BT, BA, BB	XOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		28-45
mcrf	BF, BFA	Move CR field, (CR[CRn]) ← (CR[CRm]) where m ← BFA and n ← BF.		28-96

B.8 BRANCH INSTRUCTIONS

The architecture provides conditional and unconditional branches to any storage location. The conditional branch instructions test condition codes set previously and branch accordingly. Conditional branch instructions may decrement and test the Count Register (CTR) as part of determination of the branch condition and may save the return address in the Link Register (LR). The target address for a branch may be a displacement from the current instruction address (CIA), or may be contained in the LR or CTR, or may be an absolute address.

Table B-8. Branch Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
b	target	Branch unconditional relative. LI ← $(target - CIA)_{6:29}$ NIA ← CIA + EXTS(LI ‖ $^{2}0$)		28-20
ba		Branch unconditional absolute. LI ← $target_{6:29}$ NIA ← EXTS(LI ‖ $^{2}0$)		
bl		Branch unconditional relative. LI ← $(target - CIA)_{6:29}$ NIA ← CIA + EXTS(LI ‖ $^{2}0$)	(LR) ← CIA + 4.	
bla		Branch unconditional absolute. LI ← $target_{6:29}$ NIA ← EXTS(LI ‖ $^{2}0$)	(LR) ← CIA + 4.	
bc	BO, BI, target	Branch conditional relative. BD ← $(target - CIA)_{16:29}$ NIA ← CIA + EXTS(BD ‖ $^{2}0$)	CTR if BO_2 = 0.	28-21
bca		Branch conditional absolute. BD ← $target_{16:29}$ NIA ← EXTS(BD ‖ $^{2}0$)	CTR if BO_2 = 0.	
bcl		Branch conditional relative. BD ← $(target - CIA)_{16:29}$ NIA ← CIA + EXTS(BD ‖ $^{2}0$)	CTR if BO_2 = 0. (LR) ← CIA + 4.	
bcla		Branch conditional absolute. BD ← $target_{16:29}$ NIA ← EXTS(BD ‖ $^{2}0$)	CTR if BO_2 = 0. (LR) ← CIA + 4.	
bcctr	BO, BI	Branch conditional to address in CTR. Using (CTR) at exit from instruction, NIA ← $CTR_{0:29}$ ‖ $^{2}0$.	CTR if BO_2 = 0.	28-26
bcctrl			CTR if BO_2 = 0. (LR) ← CIA + 4.	
bclr	BO, BI	Branch conditional to address in LR. Using (LR) at entry to instruction, NIA ← $LR_{0:29}$ ‖ $^{2}0$.	CTR if BO_2 = 0.	28-29
bclrl			CTR if BO_2 = 0. (LR) ← CIA + 4.	

B.9 COMPARISON INSTRUCTIONS

Comparison instructions perform arithmetic and logical comparisons between two operands and set one of the eight condition code register fields based on the outcome of the comparison. Table B-9 shows the comparison instructions supported by IOP 480 CPU.

Table B-9. Comparison Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
cmp	BF, 0, RA, RB	Compare (RA) to (RB), signed. Results in CR[CRn], where n = BF.		28-33
cmpi	BF, 0, RA, IM	Compare (RA) to EXTS(IM), signed. Results in CR[CRn], where n = BF.		28-34
cmpl	BF, 0, RA, RB	Compare (RA) to (RB), unsigned. Results in CR[CRn], where n = BF.		28-35
cmpli	BF, 0, RA, IM	Compare (RA) to ($^{16}0$ || IM), unsigned. Results in CR[CRn], where n = BF.		28-36

B.10 ROTATE AND SHIFT INSTRUCTIONS

Rotate and shift instructions rotate and shift operands which are stored in the general purpose registers.

Rotate instructions can also mask rotated operands. Table B-10 lists IOP 480 CPU rotate and shift instructions.

Table B-10. Rotate and Shift Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
rlwimi	RA, RS, SH, MB, ME	Rotate left Lword immediate, then insert according to mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m) ∨ ((RA) ∧ ¬m)		28-121
rlwimi.			CR[CR0]	
rlwinm	RA, RS, SH, MB, ME	Rotate left Lword immediate, then AND with mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m)		28-122
rlwinm.			CR[CR0]	
rlwnm	RA, RS, RB, MB, ME	Rotate left Lword, then AND with mask. r ← ROTL((RS), $(RB)_{27:31}$) m ← MASK(MB, ME) (RA) ← (r ∧ m)		28-124
rlwnm.			CR[CR0]	
slw	RA, RS, RB	Shift left (RS) by $(RB)_{27:31}$. n ← $(RB)_{27:31.}$ r ← ROTL((RS), n). if $(RB)_{26}$ = 0 then m ← MASK(0, 31 – n) else m ← $^{32}0$. (RA) ← r ∧ m.		28-126
slw.			CR[CR0]	
sraw	RA, RS, RB	Shift right algebraic (RS) by $(RB)_{27:31}$. n ← $(RB)_{27:31}$. r ← ROTL((RS), 32 – n). if $(RB)_{26}$ = 0 then m ← MASK(n, 31) else m ← $^{32}0$. s ← $(RS)_{0.}$ (RA) ← (r ∧ m) ∨ (^{32}s ∧ ¬m). XER[CA] ← s ∧ ((r ∧ ¬m) ≠ 0).		28-127
sraw.			CR[CR0]	
srawi	RA, RS, SH	Shift right algebraic (RS) by SH. n ← SH. r ← ROTL((RS), 32 – n). m ← MASK(n, 31). s ← $(RS)_{0.}$ (RA) ← (r ∧ m) ∨ (^{32}s ∧ ¬m). XER[CA] ← s ∧ ((r ∧ ¬m)≠0).		28-128
srawi.			CR[CR0]	
srw	RA, RS, RB	Shift right (RS) by $(RB)_{27:31}$. n ← $(RB)_{27:31}$. r ← ROTL((RS), 32 – n). if $(RB)_{26}$ = 0 then m ← MASK(n, 31) else m ← $^{32}0$. (RA) ← r ∧ m.		28-129
srw.			CR[CR0]	

B.11 CACHE CONTROL INSTRUCTIONS

Cache control instructions allow the user to indirectly control the contents of the data and instruction caches. The user may fill, flush, invalidate and zero blocks (16-byte lines) in the data cache. The user may also invalidate congruence classes in both caches and invalidate individual lines in the instruction cache.

Table B-11. Cache Control Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
dcba	RA, RB	Speculatively establish the data cache block which contains the effective address (RA|0) + (RB).		28-46
dcbf	RA, RB	Flush (store, then invalidate) the data cache block which contains the effective address (RA|0) + (RB).		28-48
dcbi	RA, RB	Invalidate the data cache block which contains the effective address (RA|0) + (RB).		28-49
dcbst	RA, RB	Store the data cache block which contains the effective address (RA|0) + (RB).		28-50
dcbt	RA, RB	Load the data cache block which contains the effective address (RA|0) + (RB).		28-51
dcbtst	RA,RB	Load the data cache block which contains the effective address (RA|0) + (RB).		28-52
dcbz	RA, RB	Zero the data cache block which contains the effective address (RA|0) + (RB).		28-53
dccci	RA, RB	Invalidate the data cache congruence class associated with the effective address (RA|0) + (RB).		28-55
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the effective address (RA|0) + (RB). Place the results in RT.		28-56
icbi	RA, RB	Invalidate the instruction cache block which contains the effective address (RA|0) + (RB).		28-64
icbt	RA, RB	Load the instruction cache block which contains the effective address (RA|0) + (RB).		28-65
iccci	RA, RB	Invalidate instruction cache congruence class associated with the effective address (RA|0) + (RB).		28-67
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the effective address (RA|0) + (RB). Place the results in ICDBDR.		28-69

B.12 INTERRUPT CONTROL INSTRUCTIONS

The interrupt control instructions allow the user to move data between general purpose registers and the machine state register, return from interrupts and enable or disable maskable external interrupts. Table B-12 shows the Interrupt control instruction set.

Table B-12. Interrupt Control Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
mfmsr	RT	Move from MSR to RT, (RT) ← (MSR).		28-100
mtmsr	RS	Move to MSR from RS, (MSR) ← (RS).		28-105
rfci		Return from critical interrupt (PC) ← (SRR2). (MSR) ← (SRR3).		28-119
rfi		Return from interrupt. (PC) ← (SRR0). (MSR) ← (SRR1).		28-120
wrtee	RS	Write value of RS_{16} to the External Enable bit (MSR[EE]).		28-169
wrteei	E	Write value of E to the External Enable bit (MSR[EE]).		28-170

B.13 PROCESSOR MANAGEMENT INSTRUCTIONS

The processor management instructions move data between GPRs, SPRs and DCRs in IOP 480 CPU; these instructions also provide traps, system calls and synchronization controls.

Table B-13. Processor Management Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
eieio		Storage synchronization. All loads and stores that precede the **eieio** instruction complete before any loads and stores that follow the instruction access main storage. Implemented as **sync**, which is more restrictive.		28-60
isync		Synchronize execution context by flushing the prefetch queue.		28-71
mcrxr	BF	Move XER[0:3] into field CRn, where n←BF. CR[CRn] ← (XER[SO, OV, CA]). (XER[SO, OV, CA]) ← $^{3}0$.		28-97
mfcr	RT	Move from CR to RT, (RT) ← (CR).		28-98
mfdcr	RT, DCRN	Move from DCR to RT, (RT) ← (DCR(DCRN)).		28-99
mfspr	RT, SPRN	Move from SPR to RT, (RT) ← (SPR(SPRN)).		28-101
mtcrf	FXM, RS	Move some or all of the contents of RS into CR as specified by FXM field, mask ← $^{4}(FXM_0)$ || $^{4}(FXM_1)$ || ... || $^{4}(FXM_6)$ || $^{4}(FXM_7)$. (CR)←((RS) ∧ mask) ∨ (CR) ∧ ¬mask).		28-103
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) ← (RS).		28-104
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) ← (RS).		28-106
sc		System call exception is generated. (SRR1) ← (MSR) (SRR0) ← (PC) PC ← $EVPR_{0:15}$ || x'0C00' (MSR[WE, PR, EE, PE, DR, IR]) ← 0 (MSR[LE]) ← (MSR[ILE])		28-125
sync		Synchronization. All instructions that precede **sync** complete before any instructions that follow **sync** begin. When **sync** completes, all storage accesses initiated prior to **sync** are completed.		28-155
tw	TO, RA, RB	Trap exception is generated if, comparing (RA) with (RB), any condition specified by TO is true.		28-163
twi	TO, RA, IM	Trap exception is generated if, comparing (RA) with EXTS(IM), any condition specified by TO is true.		28-166

C REAL CODE EXAMPLE

```
! ------------------------------------------------------------------------
! name:         init.asm
! ------------------------------------------------------------------------
! This code is written for simulation debugging. Therefore, lots of shortcuts
! are in place to speed up the simulation that arent needed for the actual
! silicon (such as skipping Cache initialization). Also, r10 is used only
! as an indication of progression through the code.
#include <arch.inc>
#include <test.inc>
        PROLOG(init)
        addi    r10,0,0x0000            ! use r10 to help debug simulation
        addi    r3 ,0,0                 ! Turn off Guarded Mode for faster
        mtsgr   r3                      ! execution of Instructions
        mtctr   r3                      ! Init common implicit SPRs
        mtxer   r3
        mtlr    r3
        mtcrf   0xff,r3                 ! Zero out the condition register
        addis   r3,0,0xfff7             ! set interrupt vector prefix
        mtevpr  r3
! to speed up simulation, conditionally skip cache initialization
! based on value stored in cr_CacheInit
        addi    r10,0,0x0001            ! use r10 to help debug simulation
        addis   r4, 0,cr_CacheInit@h    ! Cache Init Crunch Address
        ori     r4,r4,cr_CacheInit@l
        lwz     r5, 0(r4)
        cmpi    r5, 0                   ! 0 = init cache; 1 = dont bother.
        bne     PastCacheInit
                                        ! Dont bother doing this unless the
                                        ! Cache will be used, its very slow
        addi    r10,0,0x0002            ! use r10 to help debug simulation
! must invalidate each cache line for half of each cache (data and instruction)
! only half, since cache is 2-way set associative
! since I-cache is 2x D-cache, loop for D-cache size and hit two I-cache lines
        addi    r3,0,1024               ! 1/4 of I-Cache Size, Half of D-Cache
        addi    r4,0,1024               ! 1/4 of I-Cache Size
loop:   iccci   0,r3
        iccci   r4,r3
        dccci   0,r3
        addic.  r3,r3,-16               ! Move Back one Cache Line
        bne     loop                    ! Loop back to do rest until r3 = 0
```

```
        addi    r3,0,0                  ! Init the DCWR
        mtdcwr  r3
        addis   r3,0,0x8000             ! Init Cache Control Registers
        ori     r3,r3,0x0001            ! First and last region are cacheable
        or      r3,r3,r3                ! Two Noops to force first cacheable
                                        ! fetch from middle of cache line
        or      r3,r3,r3
        mticcr  r3
        mtdccr  r3
        sync
        isync
PastCacheInit:
        addi    r10,0,0x0003            ! use r10 to help debug simulation
!
! Other initialization code goes here
!
! initialize IOP480 so it can read a 1 MB code area for SRAM spaces
! this requires enabling MA[17:0], by default only  MA[12:0] are enabled.
        ! first copy IOP480 register base address into r3
        addis   r3,0,0x5000         ! IOP480 register base address hi
        ori     r3,r3,0x0000        ! IOP480 register base address lo
        ! use r4 as the working variable
        ! turn on MA17 (LOCCTL.7)
        lwz     r4, 0x84(r3)        ! LOCCTL is at offset 0x84
        ori     r4,r4,0x80          ! make bit 7 = 1 -- MA17 mode
        stw     r4, 0x84(r3)        ! LOCCTL is at offset 0x84
        ! turn on MA16:13 (LCS0BRD.12 = 0)
        lwz     r4, 0x100(r3)       ! LCS0BRD is at offset 0x100
        andi.   r4,r4,0xefff        ! make bit 12 = 0
        stw     r4, 0x100(r3)       ! LCS0BRD is at offset 0x100
        ! make sure above stores all complete before going on
        sync
        ! last thing, set local init status done bit (DEVINIT.31)
        lwz     r4, 0x80(r3)        ! DEVINIT is at offset 0x80
        oris    r4,r4,0x8000        ! make bit 31 = 1
        stw     r4, 0x80(r3)        ! DEVINIT is at offset 0x80
        ! make sure above stores all complete before going on
        sync
! initialize on-chip UART
! simulation conditionally skips this step (use cache init again)
!        addis   r4, 0,cr_CacheInit@h    ! Cache Init Crunch Address
!        ori     r4,r4,cr_CacheInit@l
```

```
!        lwz     r5, 0(r4)
!        cmpi    r5, 0                   ! 0 = init UART; 1 = dont bother.
        addi    r10,0,0x0004            ! use r10 to help debug simulation
!        bne     PastUartInit
        addi    r10,0,0x0005            ! use r10 to help debug simulation
        andi.   r8,0,0x0000
        ori     r9,0,0x00ff
        addis   r4,r8,0x4000        ! IOP480 UART base address hi
        ori     r4,r4,0x0000        ! IOP480 UART base address lo
        addi    r10,0,0x0006        ! use r10 to help debug simulation
        ! Recommended initialization procedure for the UART:
        !    First use these EXACT values, then program what you want.
        !    By not following these steps, simulation shows that the UART may
        !    not run properly.
        ori     r5,r8,0x0000        ! initial BRDH value
        stb     r5, 0x10(r4)        ! store byte to BRDH
        ori     r5,r8,0x0000        ! initial BRDL value
        stb     r5, 0x14(r4)        ! store byte to BRDL
        ori     r5,r8,0x0078        ! initial SPLS value
        stb     r5, 0x00(r4)        ! store byte to SPLS
        ori     r5,r8,0x00ff        ! initial SPHS value
        stb     r5, 0x08(r4)        ! store byte to SPHS
        ori     r5,r8,0x0000        ! initial SPCTL value
        stb     r5, 0x18(r4)        ! store byte to SPCTL
        ori     r5,r8,0x0000        ! initial SPRC value
        stb     r5, 0x1c(r4)        ! store byte to SPRC
        ori     r5,r8,0x0000        ! initial SPTC value
        stb     r5, 0x20(r4)        ! store byte to SPTC
        ! end of exact UART initialization
        ! now UART should be ready to be programmed with real values
        ! for simulation make baud rate as fast as possible
        ori     r5,r8,0x0000        ! initial BRDH value
        stb     r5, 0x10(r4)        ! store byte to BRDH
        ori     r5,r8,0x0001        ! initial BRDL value
        stb     r5, 0x14(r4)        ! store byte to BRDL
        ori     r5,r8,0x0008        ! initial SPCTL value
        stb     r5, 0x18(r4)        ! store byte to SPCTL
        ori     r5,r8,0x00ff        ! initial SPLS value
        stb     r5, 0x00(r4)        ! store byte to SPLS
        ori     r5,r8,0x00fe        ! initial SPHS value
        stb     r5, 0x08(r4)        ! store byte to SPHS
        ori     r5,r8,0x0008        ! initial SPCTL value
```

```
        stb     r5, 0x18(r4)        ! store byte to SPCTL
        ori     r5,r8,0x00b0        ! initial SPRC value
        stb     r5, 0x1c(r4)        ! store byte to SPRC
        ori     r5,r8,0x00b0        ! initial SPTC value
        stb     r5, 0x20(r4)        ! store byte to SPTC
        ! clear SPLS one more time
        !ori     r5,r8,0x00ff        ! initial SPLS value
        !stb     r5, 0x00(r4)        ! store byte to SPLS
!PastUartInit:
        addi    r10,0,0x0007        ! use r10 to help debug simulation
! now branch to main test code
        ba      0xfff80000          ! Branch to test case
        addi    r10,0,0xf00d        ! use r10 to help debug simulation
        EPILOG(init)
        .space  4
        .data
data:
```

D GENERAL INFORMATION

D.1 ORDERING INSTRUCTIONS

The IOP 480 is a 32-bit 33-MHz PCI Bus Master I/O Accelerator featuring advanced Data Pipe Architecture technology, which includes two DMA engines, programmable Target and Initiator Data Transfer modes, and PCI messaging functions. The IOP 480 offers 3.3V, 5V tolerant PCI and Local signaling supports Universal PCI Adapter designs, 3.3V core, low-power CMOS offered in two package options, a 208-pin PQFP and 225-pin (ball) PBGA. The device is designed to operate at Industrial Temperature range.

The IOP 480 is also available in two Local Bus speed options—60 MHz or 66 MHz.

Table D-1. Available Packages

Package	Local Bus Speed (Max)	Ordering Part Number
208-pin PQFP	60 MHz	IOP 480–AA60PI
225-pin PBGA		IOP 480–AA60BI
208-pin PQFP	66 MHz	IOP 480–AA66PI
225-pin PBGA		IOP 480–AA66BI

D.2 UNITED STATES AND INTERNATIONAL REPRESENTATIVES, AND DISTRIBUTORS

A list of PLX Technology, Inc., representatives and distributors can be found at http://www.plxtech.com.

D.3 TECHNICAL SUPPORT

PLX Technology, Inc., technical support information is listed at http://www.plxtech.com; or call 408 774-9060 or 800 759-3735.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z