PPC405CR

Embedded Processor

User's Manual

Preliminary

PowerPC™

SA14-2518-02

Third Preliminary Edition (October 2000)

This edition of *IBM PPC405CR Embedded Processor User's Manual* applies to the IBM PPC405CR 32-bit embedded processor, until otherwise indicated in new versions or application notes.

Requests for copies of this publication and for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Address technical queries about this product to ppcsupp@us.ibm.com

Address comments about this publication to:

IBM Corporation
Department YM5A
P.O. Box 12195
Research Triangle Park, NC 27709

4 3 2 1

Patents and Trademarks

IBM may have patents or pending patent applications covering the subject matter in this publication. The furnishing of this publication does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904, United States of America.

The following terms are trademarks of IBM Corporation:

IBM
PowerPC
PowerPC Architecture
PowerPC Embedded Processors
RISCTrace
RISCWatch

Other terms which are trademarks are the property of their respective owners.

Contents

Figures

Tables

About This Book

This user's manual provides the architectural overview, programming model, and detailed information about the registers, the instruction set, and operations of the IBM™ PowerPC™ 405CR (PPC405CR) 32-bit RISC embedded processor.

The PPC405CR RISC embedded processor features:

- PowerPC Architecture™
- Single-cycle execution for most instructions
- Instruction cache unit and data cache unit
- Support for Little Endian operation
- Interrupt interface for one critical and one non-critical interrupt signal
- JTAG interface
- Extensive development tool support

Who Should Use This Book

This book is for system hardware and software developers, and for application developers who need to understand the PPC405CR. The audience should understand embedded processor design, embedded system design, operating systems, RISC processing, and design for testability.

How to Use This Book

This book describes the PPC405CR device architecture, programming model, external interfaces, internal registers, and instruction set. This book contains the following chapters, arranged in parts:

This book contains the following appendixes:

To help readers find material in these chapters, the book contains:

Conventions

The following is a list of notational conventions frequently used in this manual.

$\overline{\text{ActiveLow}}$	An overbar indicates an active-low signal.
n	A decimal number
0x*n*	A hexadecimal number
0b*n*	A binary number
=	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 scope of a loop.
leave	Leave innermost do loop or do loop specified in a leave statement.
FLD	An instruction or register field
FLD_b	A bit in a named instruction or register field
$FLD_{b:b}$	A range of bits in a named instruction or register field
$FLD_{b,b,\ldots}$	A list of bits, by number or name, in a named instruction or register field
REG_b	A bit in a named register
$REG_{b:b}$	A range of bits in a named register
$REG_{b,b,\ldots}$	A list of bits, by number or name, in a named register
REG[FLD]	A field in a named register
REG[FLD, FLD . . .]	A list of fields in a named register
REG[FLD:FLD]	A range of fields in a named register
GPR(r)	General Purpose Register (GPR) r, where $0 \le r \le 31$.
(GPR(r))	The contents of GPR r, where $0 \le r \le 31$.
DCR(DCRN)	A Device Control Register (DCR) specified by the DCRF field in an **mfdcr** or **mtdcr** instruction
SPR(SPRN)	An SPR specified by the SPRF field in an **mfspr** or **mtspr** instruction
TBR(TBRN)	A Time Base Register (TBR) specified by the TBRF field in an **mftb** instruction
GPRs	RA, RB, . . .
(Rx)	The contents of a GPR, where *x* is A, B, S, or T
(RA|0)	The contents of the register RA or 0, if the RA field is 0.
CR_{FLD}	The field in the condition register pointed to by a field of an instruction.
$c_{0:3}$	A 4-bit object used to store condition results in compare instructions.
^{n}b	The 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 pseudocode, 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)	The 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 a location in main storage.
EA_b	A bit in an effective address.
$EA_{b:b}$	A range of bits in an effective address.
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 0s elsewhere.
instruction(EA)	An instruction operating on a data or instruction cache block associated with an EA.

Part I. Introducing the PPC405CR Embedded Processor

Chapter 1. Overview

The IBM PowerPC 405CR 32-bit reduced instruction set computer (RISC) embedded processor, referred to as the PPC405CR, is a system-on-a-chip (SOC) that integrates a PowerPC embedded processor core with a rich set of on-chip peripherals:

- SDRAM controller
- External bus controller (EBC)
- Direct memory access (DMA) with scatter/gather support
- Two serial ports
- Inter-integrated circuit (IIC) interface
- General-purpose input/output (GPIO)

This chapter describes:

- PPC405CR features
- The PowerPC Architecture™
- The PPC405CR implementation of the IBM PowerPC Embedded Environment, an extension of the PowerPC Architecture for embedded applications
- PPC405CR organization, including a block diagram and descriptions of the functional units
- PPC405CR registers
- PPC405CR addressing modes

Figure 1-1 illustrates the logical organization of the PPC405CR:

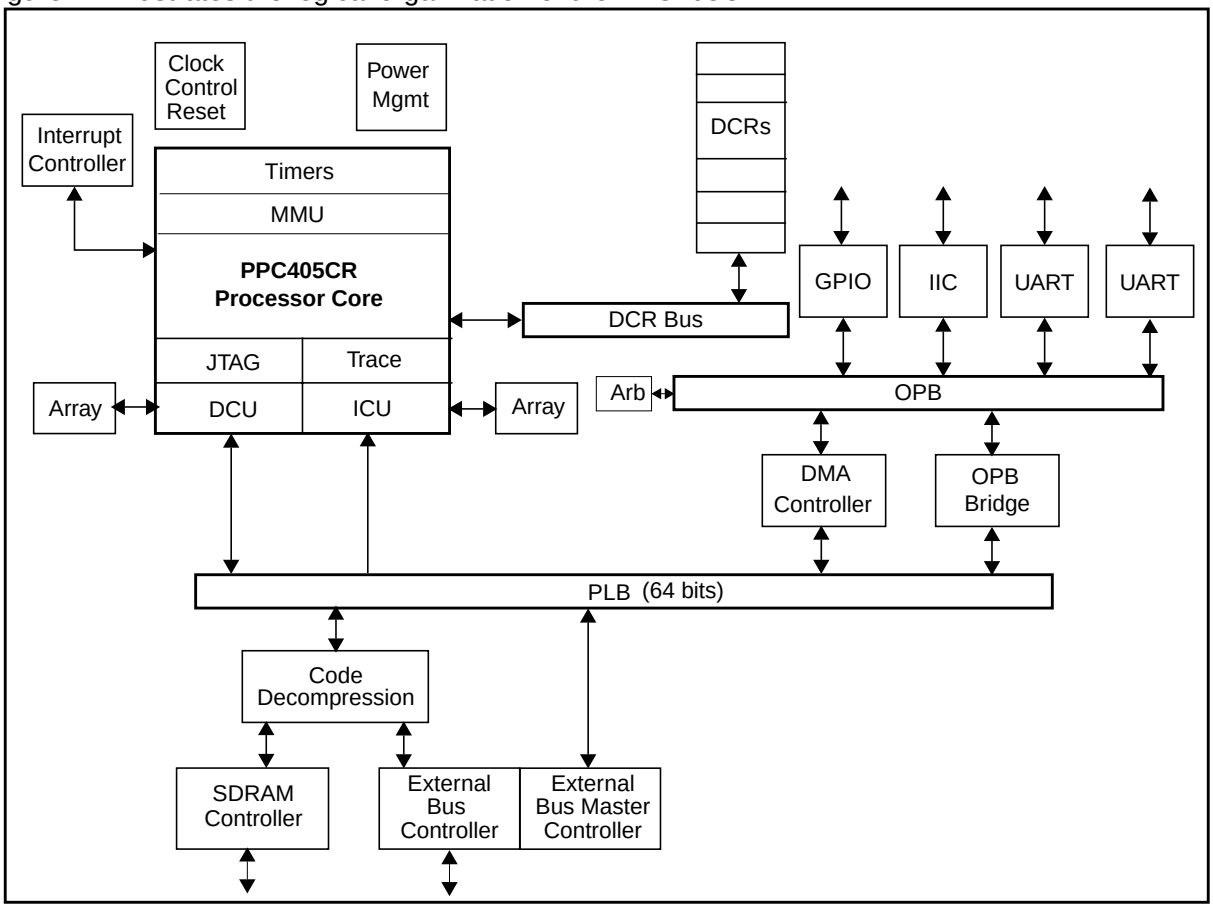


Figure 1-1. PPC405CR Block Diagram

1.1 PPC405CR Features

The PPC405CR provides high performance and low power consumption. The PPC405CR RISC CPU executes at sustained speeds approaching one cycle per instruction. On-chip instruction and data caches reduce chip count and design complexity in systems and improve system throughput.

1.1.1 Bus and Peripheral Features

The PPC405CR multilevel bus architecture and peripherals feature:

- Processor local bus (PLB)
- On-chip peripheral bus (OPB)
- PC-100 synchronous DRAM (SDRAM) controller
  - 32-bit interface for non-ECC applications
  - 40-bit interface (32 data bits and 8 check bits) for ECC applications

- External bus controller (EBC)
  - Flash ROM/Boot ROM interface
  - Direct support for 8-, 16-, or 32-bit SRAM or external peripherals
  - One external master supported
- DMA support for OPB and external peripherals
- Interrupt controller supporting programmable interrupt handling from a variety of sources
- Two 8-bit serial ports (16550 compatible UARTs)
- Inter-integrated circuit (IIC) controller
- General purpose I/O (GPIO) controller

1.1.2 PowerPC Processor Core Features

The PowerPC RISC fixed-point CPU features:

- PowerPC User Instruction Set Architecture (UISA) and extensions for embedded applications
- Thirty-two 32-bit general purpose registers (GPRs)
- Static branch prediction
- Five-stage pipeline with single-cycle execution of most instructions, including loads/stores
- Unaligned load/store support to cache arrays, and main memory Hardware multiply/divide for faster integer arithmetic (4-cycle multiply, 35-cycle divide)
- Multiply-accumulate instructions
- Enhanced string and multiple-word handling
- True little endian operation
- Programmable Interval Timer (PIT), Fixed Interval Timer (FIT), and watchdog timer
- Forward and reverse trace from a trigger event
- Storage control
  - Separate, configurable, two-way set-associative instruction and data cache units
  - Eight words (32 bytes) per cache line
  - 16KB instruction and 8KB data cache arrays
  - Instruction cache unit (ICU) non-blocking during line fills, data cache unit (DCU) non-blocking during line fills and flushes
  - Read and write line buffers
  - Instruction fetch hits are supplied from line buffer
  - Data load/store hits are supplied to line buffer
  - Programmable ICU prefetching of next sequential line into line buffer
  - Programmable ICU prefetching of non-cacheable instructions, full line (eight words) or half line (four words)
  - Write-back or write-through DCU write strategies
  - Programmable allocation on loads and stores

  - Operand forwarding during cache line fills
- Memory Management
  - Translation of the 4GB 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
  - WIU0GE (write-through, cachability, user-defined 0, guarded, endian) storage attribute control for each virtual memory region
- WIU0GE storage attribute control for thirty-two real 128MB regions
- PowerPC timer facilities
  - 64-bit time base
  - PIT, FIT, and watchdog timers
  - Synchronous external time base clock input
- Debug Support
  - Enhanced debug support with logical operators
  - Four instruction address compares (IACs)
  - Two data address compares (DACs)
  - Two data value compares (DVCs)
  - JTAG instruction to write to ICU
  - Forward or backward instruction tracing
- Minimized interrupt latency
- Advanced power management support

1.2 PowerPC Architecture

The PowerPC Architecture comprises three levels of standards:

- PowerPC User Instruction Set Architecture (UISA), 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 the instruction set 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 is described in *The PowerPC Architecture: A Specification for a New Family of RISC Processors*.

The PowerPC Architecture provides compatibility of PowerPC Book I application code across all PowerPC implementations to help maximize the portability of applications developed for PowerPC processors. This is accomplished through compliance with the first level of the architectural definition, the PowerPC UISA, which is common to all PowerPC implementations.

1.3 The PPC405CR as a PowerPC Implementation

The PPC405CR implements the PowerPC UISA, user-level registers, programming model, data types, addressing modes, and 32-bit fixed-point operations. The PPC405CR fully complies with the PowerPC UISA. The UISA 64-bit and floating point operations are not implemented. The floating point operations, which cause exceptions, can then be emulated by software.

Most of the features of the PPC405CR processor core are compatible with the PowerPC Virtual Environment and Operating Environment Architectures, as implemented in PowerPC processors such as the 6xx/7xx family. The PPC405CR processor core also provides a number of optimizations and extensions to these layers of the PowerPC Architecture. The full architecture of the PPC405CR 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
- An architected DCR address space for integrated peripheral control
- The addition of several instructions to support these modified and extended resources

Finally, some of the specific implementation features of the PPC405CR are beyond the scope of the PowerPC Architecture. These features are included to enhance performance, integrate functionality, and reduce system complexity in embedded control applications.

1.4 RISC Processor Core Organization

The processor core consists of a 5-stage pipeline, separate instruction and data cache units, virtual memory management unit (MMU), three timers, debug, and interfaces to other functions.

1.4.1 Instruction and Data Cache Controllers

The PPC405CR processor core uses a 16KB instruction cache unit (ICU) and an 8KB data cache unit (DCU) to enable concurrent accesses and minimize pipeline stalls. Both cache units are two-way set-associative and use a 32-byte line size. The instruction set provides a rich assortment of cache control instructions, including instructions to read tag information and data arrays. See Chapter 4, “Cache Operations,” for detailed information about the ICU and DCU.

1.4.1.1 Instruction Cache Unit

The ICU provides one or two instructions per cycle to the execution unit (EXU) over a 64-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 words 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 PLB activity to increase PLB availability for other on-chip cores, such as the DMA controller.

1.4.1.2 Data Cache Unit

The DCU transfers 1, 2, 3, 4, or 8 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 word 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.

Additional DCU features enable the programmer to tailor performance for a given application. The DCU can function in write-back or write-through mode, as controlled by the Data Cache Write-through Register (DCWR) or the translation look-aside buffer (TLB). DCU performance can be tuned to balance performance and memory coherency. Store-without-allocate, controlled by the SWOA field of the Core Configuration Register 0 (CCR0), can inhibit line fills caused by store misses to further reduce potential pipeline stalls and unwanted external bus traffic. Similarly, load-without-allocate, controlled by CCR0[LWOA], can inhibit line fills caused by load misses.

1.4.2 Memory Management Unit

The 4GB address space of the PPC405CR is presented as a flat address space.

The 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 4GB 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 PPC405CR 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 page entry can be placed anywhere in the TLB. The translation function of the MMU occurs pre-cache for data accesses. Cache tags and indexing use physical addresses for data accesses; instruction fetches are virtually indexed and physically tagged.

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 for globally accessible static mappings. The instruction set provides several instructions 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 (1KB, 4KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB) are simultaneously supported, so that, at any given time, the TLB can contain entries for any combination of page sizes. For a logical to physical translation to occur, 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, 4 instruction-side and 8 data-side TLB entries are kept in shadow arrays. The shadow arrays prevent TLB contention. Hardware manages the replacement and invalidation of shadow-TLB entries; no system software action is required. The shadow arrays can be thought of as level 1 TLBs, with the main TLB serving as a level 2 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.

The PowerPC Architecture provides WIU0GE (write-back/write through, cachability, user-defined 0, guarded, endian) storage attributes that control memory accesses, using bits in the TLB or, when address translation is disabled, storage attribute control registers.

When address translation is enabled (MSR[IR, DR] = 1), storage attribute control bits in the TLB control the storage attributes associated with the current page. When address translation is disabled (MSR[IR, DR] = 0), bits in each storage attribute control register control the storage attributes associated with storage regions. Each storage attribute control register contains 32 fields. Each field sets the associated storage attribute for a 128MB memory region. See “Real-mode Storage Attribute Control” on page 5-17 for more information about the storage attribute control registers.

1.4.3 Timer Facilities

The processor core 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 a decrement occurs on a PIT count of 1, 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.

1.4.4 Debug

The processor core 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 hardware and software development tools.

1.4.4.1 Development Tool Support

The PPC405CR supports a wide range of hardware and software development tools.

An operating system debugger is an example of an operating system-aware debugger, implemented using software traps.

RISCWatch is 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.

The RISCTrace™ feature of RISCWatch is an example of a development tool that uses the real-time trace capability of the processor core.

1.4.4.2 Debug Modes

The internal, external, real-time-trace, and debug wait modes support a variety of debug tool used in embedded systems development. These debug modes are described in detail in “Debug Modes” on page 11-6.

1.4.5 Processor Core Interfaces

The processor core provides a range of I/O interfaces.

1.4.5.1 Processor Local Bus

The PLB-compliant interface provides separate 32-bit address and 64-bit data buses for the instruction and data sides.

1.4.5.2 Device Control Register Bus

The Device Control Register (DCR) bus interface provides access to on-chip registers for configuration and status of peripherals such as SDRAM, DMA and so on.

These registers are accessed using the **mfdcr** and **mtdcr** instructions.

1.4.5.3 Clock and Power Management

This interface supports several methods of clock distribution and power management.

1.4.5.4 JTAG

The 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 tool 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.

1.4.5.5 Interrupts

The processor core provides an interface to the UIC, an on-chip interrupt controller that is logically outside the processor core. The UIC combines asynchronous interrupt inputs from on-chip and off-chip sources and presents them to the processor core using a pair of interrupt signals: critical and non-critical.

1.5 Processor Core Programming Model

The programming model is described in detail in Chapter 3, “Programming Model.”

The PowerPC instruction set and Special Purpose Registers (SPRs) provide a high degree of user control over configuration and operation of the processor core functional units.

1.5.1 Data Types

Processor core operands are bytes, halfwords, and words. 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 multibyte operand is always the lowest memory address occupied by that operand. Byte ordering can be selected as big endian (the lowest memory address of an operand contains its most significant byte) or as little endian (the lowest memory address of an operand contains its least significant byte). See “Byte Ordering” on page 3-23 for more information about big and little endian operation.

1.5.2 Processor Core Register Set Summary

The processor core 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 device control registers (DCRs).

Chapter 21, “Register Summary,” provides a register diagram and a register field description table for each register.

1.5.2.1 General Purpose Registers

The processor core contains 32 GPRs; each register contains 32 bits. 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 instructions, can also receive instruction results and the contents of other registers.

1.5.2.2 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.

All SPRs are privileged (unavailable to user-mode programs), except the Count Register (CTR), the Link Register (LR), SPR General Purpose Registers (SPRG4–SPRG7, read-only), and the Fixed-point Exception Register (XER). Note that access to the Time Base Lower (TBL) and Time Base Upper (TBU) registers, when addressed as SPRs, is write-only and privileged. However, when addressed as Time Base Registers (TBRs), read access to these registers is not privileged. See “Time Base Registers” on page 21-3 for more information.

1.5.2.3 Machine State Register

The processor core 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 processor core.

1.5.2.4 Condition Register

The processor core 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.

1.5.2.5 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.

The **mtdcr** and **mfdcr** instructions are privileged, for all DCRs. Therefore, all accesses to DCRs are privileged. See “Privileged Mode Operation” on page 3-37.

1.5.3 Memory-Mapped I/O Registers

The memory-mapped I/O (MMIO) registers are accessed using load and store instructions. MMIO registers, which are outside processor core and which are not architected, are used to control, configure, and hold status for various functional units that are not part of the processor core.

1.5.4 Addressing Modes

The processor core supports the following addressing modes, which enable 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.

Chapter 2. On-Chip Buses

The on-chip bus architecture, which consists of the processor local bus (PLB), on-chip peripheral bus (OPB), and device control register (DCR) bus, provides a link between the cache units in the processor core and other PLB and OPB master and slave devices used in the PPC405CR. These devices include the SDRAM controller, DMA controller, and external bus controller.

The PLB is a high performance bus used to access memory through bus interface units. The PLB master and slave assignments for the PPC405CR are listed in “PLB Masters and Slaves” on page 2-2.

Lower performance peripherals (such as serial ports) are attached to the OPB. A bridge between the PLB and OPB enables data transfers between PLB masters and OPB slaves. DMA peripherals can also be OPB peripherals.

The DCR bus is used primarily to access status and control registers of the various PLB and OPB masters and slaves. The DCR bus offloads status and control read and write transfers from the PLB. The DCR bus is not described further in this chapter.

The following publications, which are available from your IBM representative and in the IBM Microelectronics technical library (www.chips.ibm.com), describe the on-chip bus architecture:

- *The CoreConnect™ Bus Architecture*
- *Processor Local Bus Architecture Specifications*
- *On-Chip Peripheral Bus Architecture Specifications*
- *Device Control Register Bus Architecture Specifications*

The PPC405CR block diagram (Figure 1-1 on page 1-2) illustrates the on-chip bus structure of the PPC405CR.

2.1 Processor Local Bus

The PLB is a high-performance on-chip bus. The PLB supports read and write data transfers between master and slave devices equipped with a PLB interface and connected through PLB signals.

Each PLB master is attached to the PLB through separate address, read data and write data buses, and transfer qualifier signals. PLB slaves are attached to the PLB through shared, but decoupled, address, read data and write data buses, and transfer control and status signals for each data bus.

Access to the PLB is granted through a central arbitration mechanism that enables masters to compete for bus ownership. This arbitration mechanism provides for fixed and fair priority schemes.

Timing for all PLB signals is provided by a clock source that is shared by all PLB masters and slaves.

2.1.1 PLB Features

- Overlapping of read and write transfers allows two data transfers per clock cycle for maximum bus utilization
- Decoupled address and data buses support split-bus transaction capability for improved bandwidth
- Address pipelining reduces overall bus latency by allowing the latency associated with a new request to be overlapped with an ongoing data transfer in the same direction
- Late master request abort capability reduces latency associated with aborted requests
- Four levels of request priority and selectable arbitration modes provide flexible arbitration policies.
- Support for 16-, 32-, and 64-byte line data transfers
- Sequential burst protocol allows byte, halfword, and word burst data transfers.
- DMA buffered peripheral-to-memory, memory-to-peripheral, and memory-to-memory operations are supported

2.1.2 PLB Masters and Slaves

Table 2-1 lists the PLB masters and slaves provided in the PPC405CR.

Table 2-1. PPC405CR PLB Agents as Masters and Slaves

PLB Agent	PLB Master/Slave
Processor core ICU	Master
Processor core DCU	Master
External bus master interface	Master
Static memory/peripherals	Slaves
DMA controller	Master
PLB to OPB bridge	Slave
SDRAM controller	Slave

The SDRAM and EBC slaves are connected in line with the decompression controller. When the decompression controller is disabled, a bypass mode that directly accesses SDRAM and the EBC is used. When the decompression controller is enabled, it intercepts reads from compressed regions before they reach SDRAM and the EBC.

2.1.3 PLB Master Assignments

Each PLB master can be programmed to use one of four priority levels during PLB transfers, enabling the system designer to tune PLB transfer priorities to the requirements of a particular application. For example, if an application always requires DMA to SDRAM transfers to have the lowest latency, the DMA master can be programmed to the highest PLB master priority. This causes the PLB arbiter to grant DMA access requests before granting the access requests of any other master.

> **Programming Note:** PLB master priority assignments, which are application-dependent, must be considered carefully to prevent potential lockouts of lower priority masters. For most applications, assigning a priority of 0b10 to each master is a useful starting point.

A register associated with each master controls the priority of that master. Table 2-2 lists the PLB masters and the register fields controlling the priority of the masters. Priorities range from 0b00 (lowest) to 0b11 (highest).

Table 2-2. Registers Controlling PLB Master Priority Assignments

Master ID	Description	Register Field	Comments
0	Processor core ICU	CCR0[IPP]	
1	Processor core data cache unit DCU	CCR0[DPP1]	The high-order bit of CCR0[DPP1] is controlled by the DCU logic, so only the low-order priority bit can be programmed.
2	External bus master	EBC0_CFG[EMPL] EBC0_CFG[EMPH]	Which field sets external master priority depends upon the setting of the HoldPri signal.
5	DMA controller	DMA0_CR0[CP]	Unique priorities can be assigned to each DMA channel.
		DMA0_CR1[CP]	
		DMA0_CR2[CP]	
		DMA0_CR3[CP]	

See “PLB Arbiter Control Register (PLB0_ACR)” on page 2-5 for information about programming the PLB0_ACR to control PLB priority mode and priority order, which determine how the PLB arbitrates simultaneous PLB bus access requests having equal priorities.

2.1.4 PLB Transfer Protocol

A PLB transaction is composed of an address cycle and a data cycle.

The address cycle has three phases: request, transfer, and address acknowledge. A PLB transaction begins when a master drives its address and transfer qualifier signals and requests ownership of the bus during the request phase of the address cycle. Once bus ownership has been granted by the PLB arbiter, the master’s address and transfer qualifiers are presented to the slave devices during the transfer phase.

During normal operation, the address cycle is terminated by a slave latching the master’s address and transfer qualifiers during the address acknowledge phase.

Each data beat in the data cycle has two phases: transfer and data acknowledge. During the transfer phase, the master drives the write data bus for a write transfer or samples the read data bus for a read transfer. Data acknowledge signals are required during the data acknowledge phase for each data beat in a data cycle.

Note: For single-beat transfers, data acknowledge signals also indicate the end of the data transfer. For line or burst transfers, the data acknowledge signals apply to each beat and indicate the end of the data cycle only after the final beat.

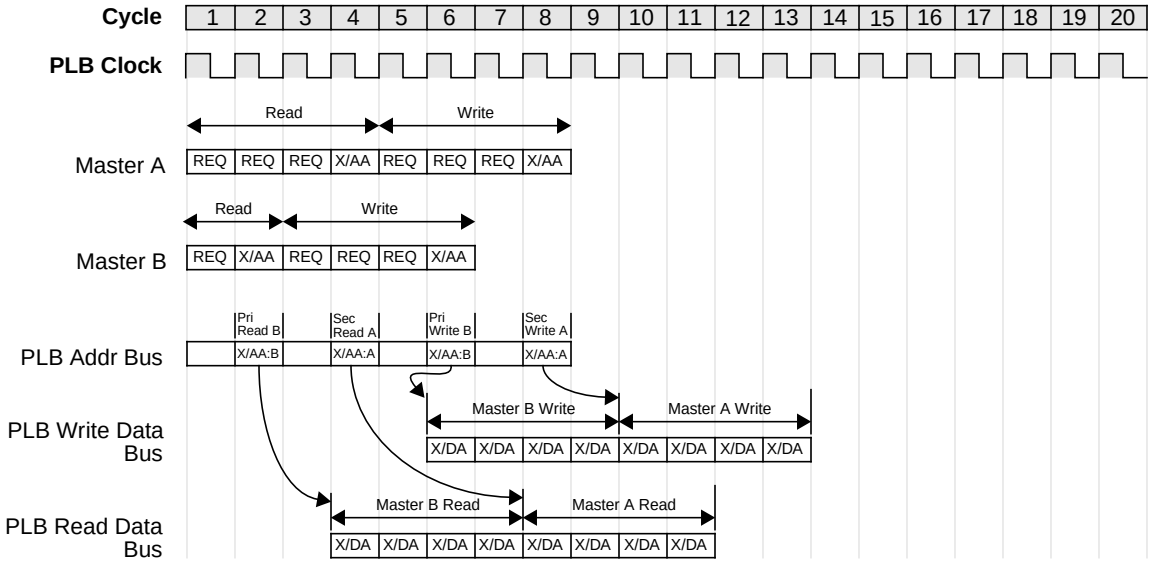
2.1.5 Overlapped PLB Transfers

Figure 2-1 shows an example of overlapped PLB transactions on the read/write data buses with pipelining. PLB address, read data, and write data buses are decoupled from one another, allowing

for address cycles to be overlapped with read or write data cycles, and for read data cycles to be overlapped with write data cycles. The PLB split-bus transaction capability allows the address and data buses to have different masters at the same time.

PLB address pipelining capability enables a new bus transfer to begin before an ongoing transfer finishes. Address pipelining reduces overall bus latency on the PLB by enabling the latency associated with a new transfer request to be overlapped with an ongoing data transfer in the same direction.

PLB masters A and B each present a read request followed by a write request. Master B gets the bus first, so its read is the primary read transaction. The master A address cycle begins as soon as the master B address cycle ends. The master A read is taken as a secondary transfer. Writes follow reads.



Note: X/AA = Xfer/AddrAck
X/DA = Xfer/DataAck
X/AA:A = X/AA Master A
X/AA:B = X/AA Master B

Figure 2-1. Overlapped PLB Transfers

Note: A master can begin to request ownership of the PLB in parallel with the address cycle or data cycle of another master bus transfer. Overlapped read and write data transfers and split-bus transactions enable the PLB to operate at a very high bandwidth.

2.1.6 PLB Arbiter Registers

PLB arbiter registers are DCRs accessed using the **mfdcr** and **mtdcr** instructions.

Table 2-3 summarizes the PLB arbiter DCRs.

Table 2-3. PLB Arbiter Registers

Mnemonic	Register Name	Address	Access	Page
PLB0_ACR	PLB Arbiter Control Register	0x087	R/W	2-5
PLB0_BEAR	PLB Error Address Register	0x086	R/O	2-6
PLB0_BESR	PLB Error Status Register	0x084	R/Clear	2-6

2.1.6.1 PLB Arbiter Control Register (PLB0_ACR)

The PLB0_ACR controls PLB arbitration priority, which is determined by PLB priority mode and PLB priority order.

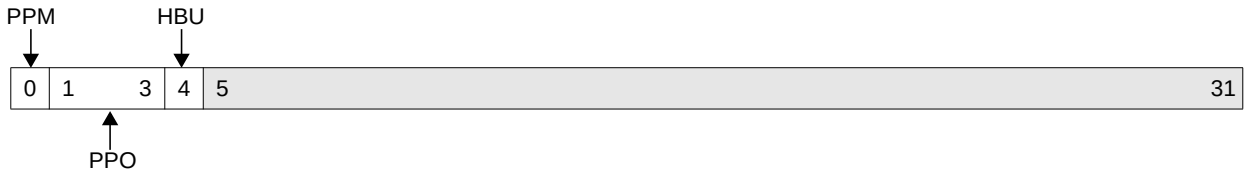


Figure 2-2. PLB Arbiter Control Register (PLB0_ACR)

0	PPM	PLB Priority Mode 0 Fixed 1 Fair
1:3	PPO	PLB Priority Order 000 Masters 0, 1, 2, 5 001 Masters 1, 2, 5, 0 010 Masters 2, 5, 0, 1 011 Masters 5,0, 1, 2 100 Reserved 101 Reserved 110 Reserved 111 Reserved
4	HBU	High Bus Utilization 0 Disabled 1 Enabled
5:31		Reserved

2.1.6.2 PLB Error Address Register (PLB0_BEAR)

The read-only PLB0_BEAR contains the address of the access on which a bus timeout error occurred.

The PLB0_BEAR can be locked by the master. Once locked, the PLB0_BEAR cannot be updated, if a subsequent error occurs, until all PLB0_BESR[FLCK*n*] fields are cleared (*n* is the master ID).

Figure 2-3. PLB Error Address Register (PLB0_BEAR)

0:31		Address of bus timeout error

2.1.6.3 PLB Error Status Register (PLB0_BESR)

The read/clear PLB0_BESR identifies timeout errors on PLB bus transfers, the master initiating the transfer, and the type of transfer.

Each PLB0_BESR[PTE*n*] field (*n* is the master ID) can be locked by the master. Once locked, PLB0_BESR [PTE*n*] fields cannot be updated if a subsequent error occurs until the corresponding PLB0_BESR [FLCK*n*] field is cleared. To clear a PLB0_BESR field, write 1 to the field. Writing 0 to a PLB0_BESR field does not affect the field.

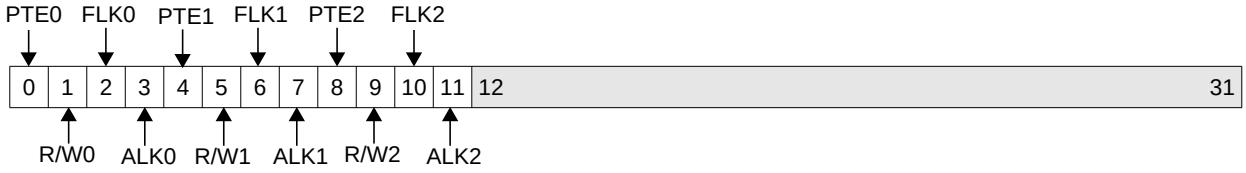


Figure 2-4. PLB Error Status Register (PLB0_BESR)

0	PTE0	Master 0 PLB Timeout Error Status 0 No master 0 timeout error 1 Master 0 timeout error	Master 0 is the processor core ICU.
1	R/W0	Master 0 Read/Write Status 0 Master 0 error operation was a write 1 Master 0 ICU error operation was a read	
2	FLK0	Master 0 PLB0_BESR Field Lock 0 Master 0 PLB0_BESR field is unlocked 1 Master 0 field is locked	
3	ALK0	Master 0 PLB0_BEAR Address Lock 0 Master 0 PLB0_BEAR is unlocked 1 Master 0 PLB0_BEAR is locked	
4	PTE1	Master 1 PLB Timeout Error Status 0 No master 1 timeout error 1 Master 1 timeout error	Master 1 is the processor core DCU.

5	R/W1	Master 1 Read/Write Status 0 Master 1 error operation was a write 1 Master 1 error operation was a read
6	FLK1	Master 1PLB0_BESR Field Lock 0 Master 1 PLB0_BESR field is unlocked 1 Master 1 PLB0_BESR field is locked
7	ALK1	Master 1 PLB0_BEAR Address Lock 0 Master 1 PLB0_BEAR is unlocked 1 Master 1 PLB0_BEAR is locked
8	PTE2	Master 2 PLB Timeout Error Status Master 2 is the external master. 0 No master 2 timeout error 1 Master 2 timeout error
9	R/W2	Master 2 Read/Write Status 0 Master 2 error operation was a write 1 Master 2 error operation was a read
10	FLK2	Master 2 PLB0_BESR Field Lock 0 Master 2 PLB0_BESR field is unlocked 1 Master 2 PLB0_BESR field is locked
11	AL2	Master 2 PLB0_BEAR Address Lock 0 Master 2 PLB0_BEAR is unlocked 1 Master 2 PLB0_BEAR is locked
12:31		Reserved

2.1.7 PLB to OPB Bridge Registers

The PLB to OPB bridge registers are DCRs accessed using the **mfdcr** and **mtdcr** instructions.

Table 2-4 lists the PLB to OPB bridge registers.

Table 2-4. PLB Arbiter Registers

Mnemonic	Register Name	Address	Access	Page
POB0_BEAR	Bridge Error Address Register	0x0A2	R/O	2-5
POB0_BESR0	Bridge Error Status Register 0 (Master IDs 0,1, 2)	0x0A0	R/Clear	2-6
POB0_BESR1	Bridge Error Status Register (Master ID 5)	0x0A4	R/Clear	2-6

2.1.7.1 Bridge Error Address Register (POB0_BEAR)

The read-only POB0_BEAR reports the address of a PLB to OPB transfer that results in an error. The PLB to OPB bridge writes the error address in the POB0_BEAR, unless the associated POB0_BESR*m*[ALCK*n*] field is set (*m* is either 0 or 1, depending on the master ID specified by *n*). Once locked, the PLB to OPB bridge cannot write POB0_BEAR until all POB0_BESR*m*[ALCK*n*] fields that are set are cleared.

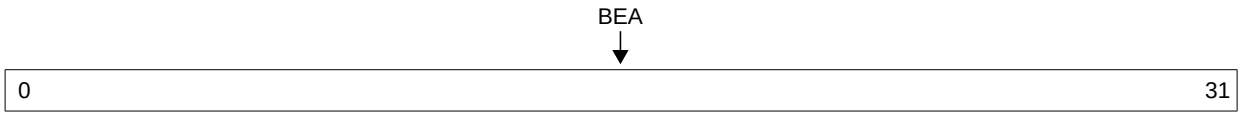


Figure 2-5. Bridge Error Address Register (POB0_BEAR)

0:31	BEA	Address of bus error

2.1.7.2 Bridge Error Status Registers (POB0_BESR0–POB0_BESR1)

The PLB to OPB bridge writes error information into the POB0_BESR. (For master IDs 0, 1, and 2, *m* = 0; for master ID 5, *m* = 1.)

POB0_BESR*m* fields can be locked using the POB0_BESR*m*[FLK*n*] and POB0_BESR*m*[ALK*n*] fields (*n* is the master ID). Once locked, the POB0_BESR*m* fields associated with a master cannot be overwritten if a subsequent error occurs until the locking fields are cleared. To clear a lock, write 1 to the POB0_BESR*m*[FLK*n*] and POB0_BESR*m*[ALK*n*] fields that are set. Writing 0 to a lock field does not affect the field.

Figure 2-6. Bridge Error Status Register 0 (POB0_BESR0)

0:1	PTE0	PLB Timeout Error Status Master 0 00 No master 0 error occurred 01 Master 0 timeout error occurred 10 Master 0 slave error occurred 11 Reserved	Master 0 is the processor core ICU.
2	R/W0	Read Write Status Master 0 0 Master 0 error operation is a read 1 Master 0 error operation is a write	
3	FLK0	POB0_BESR0 Field Lock Master 0 0 Master 0 POB0_BESR0 field is unlocked 1 Master 0 POB0_BESR0 field is locked	

4	ALK0	POB0_BEAR Address Lock Master 0 0 Master 0 POB0_BEAR address is unlocked 1 Master 0 POB0_BEAR address is locked	
5:6	PTE1	PLB Timeout Error Status Master 1 00 No master 1 error occurred 01 Master 1 timeout error occurred 10 Master 1 slave error occurred 11 Reserved	Master 1 is the processor core DCU.
7	R/W1	Read/Write Status Master 1 0 Master 1 error operation is a read 1 Master 1 error operation is a write	
8	FLK1	POB0_BESR0 Field Lock Master 1 0 Master 1 POB0_BESR0 field is unlocked 1 Master 1 POB0_BESR0 field is locked	
9	ALK1	POB0_BEAR Address Lock Master 1 0 Master 1 POB0_BEAR address is unlocked 1 Master 1 POB0_BEAR address is locked	
10:11	PTE2	PLB Timeout Error Status Master 2 00 No master 2 error occurred 01 Master 2 timeout error occurred 10 Master 2 slave error occurred 11 Reserved	Master 2 is the external master.
12	R/W2	Read/Write Status Master 2 0 Master 2 error operation is a read 1 Master 2 error operation is a write	
13	FLK2	POB0_BESR0 Field Lock Master 2 0 Master 2 POB0_BESR0 field is unlocked 1 Master 2 POB0_BESR0 field is locked	
14	ALK2	POB0_BEAR Address Lock Master 2 0 Master 2 POB0_BEAR address is unlocked 1 Master 2 POB0_BEAR address is locked	
15:31		Reserved	

Figure 2-7 illustrates POB0_BESR1.

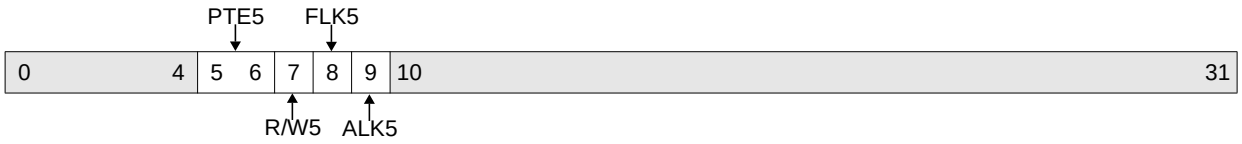


Figure 2-7. Bridge Error Status Register 1 (POB0_BESR1)

0:4		Reserved
5:6	PTE5	PLB Timeout Error Status Master 5 Master 5 is DMA. 00 No Master 5 error occurred 01 Master 5 timeout error occurred 10 Master 5 slave error occurred 11 Reserved
7	R/W5	Read/Write Status Master 5 0 Master 5 error operation is a read 1 Master 5 error operation is a write
8	FLK5	POB0_BESR1 Field Lock Master 5 0 Master 5 POB0_BESR1 field is unlocked 1 Master 5 POB0_BESR1 field is locked
9	ALK5	POB0_BEAR Address Lock Master 5 0 Master 5 POB0_BEAR address is unlocked 1 Master 5 POB0_BEAR address is locked
10:31		Reserved

2.1.7.3 On-Chip Peripheral Bus

The OPB is used to attach peripherals that do not require the bandwidth of the PLB. The OPB does not connect directly to the PPC405CR processor core, which accesses peripherals attached to the OPB through a PLB-to-OPB bridge.

2.1.8 OPB Features

The on-chip peripheral bus features:

- A 32-bit address bus and a 32-bit data bus
- Dynamic bus sizing; byte, halfword, and word transfers
- Byte and halfword duplication for byte and halfword transfers
- Single-cycle transfer of data between OPB bus master and OPB slaves
- Sequential address (burst) protocol support
- Devices on the OPB may be memory mapped, act as DMA peripherals, or support both transfer methods
- A 16-cycle fixed bus timeout provided by the OPB arbiter
- Bus parking for reduced latency

- Bus arbitration overlapped with last cycle of bus transfers

2.1.9 OPB Master Assignments

Table 2-1 lists the OPB masters. (The numbering reflects that the PPC405CR implements two masters; the OPB can support four masters.)

Table 2-5. PPC405CR OPB Master Assignments

OPB Agents	Description
DMA controller	DMA (master 0)
OPB to PLB bridge	OPB to PLB bridge (master 2)

2.1.10 OPB Arbiter Registers

The OPB arbiter contains the MMIO registers summarized in Table 2-6.

Table 2-6. PLB Arbiter Registers

Mnemonic	Register Name	Address	Access	Page
OPBA0_CR	OPB Arbiter Control Register	0xEF600601	R/W	2-5
OPBA0_PR	OPB Arbiter Priority Register	0xEF600600	R/W	2-6

2.1.10.1 OPB Arbiter Control Register (OPBA0_CR)

The OPBA0_CR fields controls updating of the OPBA0_PR (described in “OPB Arbiter Priority Register (OPBA0_PR)” on page 2-12). Because the PPC405CR provides two masters, master IDs 1 and 3 are ignored.

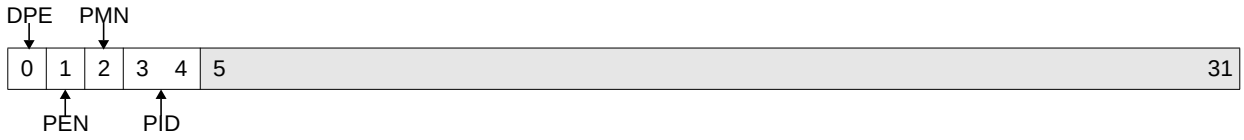


Figure 2-8. OPB Arbiter Control Register (OPBA0_CR)

0	DPE	Dynamic Priority Enable 0 Dynamic priority disabled 1 Dynamic priority enabled
1	PEN	Park Enable 0 Park disabled 1 Park enabled

2	PMN	Park on Master Not Last 0 Park on master last 1 Park on master not last	
3:4	PID	Parked Master ID 00 Master ID 0 01 Reserved 10 Master ID 2 11 Reserved	Master 0 is DMA; master 2 is the OPB to PLB bridge.
5:31		Reserved	

2.1.10.2 OPB Arbiter Priority Register (OPBA0_PR)

The OPBA0_PR assigns priorities to the OPB master IDs. Because the PPC405 provides two masters, master IDs 1 and 3 are ignored. At reset, master ID 0 (DMA) has a higher priority than master 2 (OPB to PLB bridge).

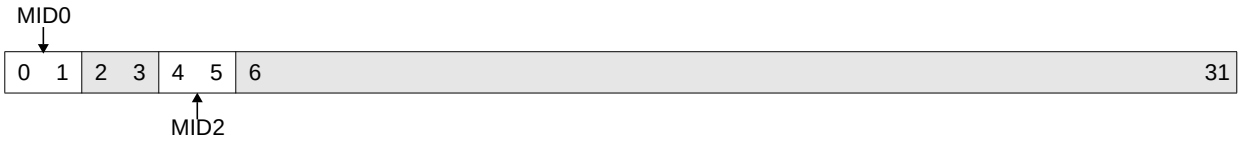


Figure 2-9. OPB Arbiter Priority Register (OPBA0_PR)

0:1	MID0	High Priority Master ID 00 Master ID 0 01 Reserved 10 Reserved 11 Reserved	At reset, this priority is assigned to DMA.
2:3		Reserved	
4:5	MID2	Low Priority master ID 00 Reserved 01 Reserved 10 Master ID 2 11 Reserved	At reset, this priority is assigned to the OPB to PLB bridge.
6:31		Reserved	

Part II. The PPC405CR RISC Processor

Chapter 3. Programming Model

The programming model of the PPC405CR embedded processor describes the following features and operations:

3.1 User and Privileged Programming Models

The PPC405CR executes programs in two modes, also referred to as states. Programs running in *privileged mode* (also referred to as the supervisor state) can access any register and execute any instruction. These instructions and registers comprise the privileged programming model. In *user mode*, certain registers and instructions are unavailable to programs. This is also called the problem state. Those registers and instructions that are available comprise the user programming model.

Privileged mode provides operating system software access to all processor resources. Because access to certain processor resources is denied in user mode, application software runs in user mode. Operating system software and other application software is protected from the effects of an errant application program.

Throughout this book, the terms user program and privileged programs are used to associate programs with one of the programming models. Registers and instructions are described as user or privileged. Privileged mode operation is described in detail in “Privileged Mode Operation” on page 3-37.

3.2 Memory Organization and Addressing

The PowerPC Architecture defines a 32-bit, 4-gigabyte (GB) address space for instructions and data.

3.2.1 Physical Address Map

Table 3-1 illustrates the physical address map. Addresses that are not shown are reserved.

Table 3-1. PPC405CR Address Space

Function	Start Address	End Address	Size
Local Memory/Peripherals[1]	0x00000000	0x7FFFFFFF	2GB
Internal Peripherals (Total)	0xEF600000	0xEFFFFFFF	10MB
UART0 Registers	0xEF600300	0xEF600307	8B
UART1 Registers	0xEF600400	0xEF600407	8B
IIC Registers	0xEF600500	0xEF600510	17B
OPB Arbiter Registers	0xEF600600	0xEF600601	2B
GPIO Controller Registers	0xEF600700	0xEF60077F	128B
Expansion ROM[2]	0xF0000000	0xFFDFFFFF	254MB
Boot ROM[2]	0xFFE00000	0xFFFFFFFF	2MB

1. The local memory/peripheral area of the memory map can be configured for SDRAM, ROM, or peripherals.

2. The boot ROM and expansion ROM areas of the memory map are intended for ROM or flash devices. The controller supports volatile memory devices such as SDRAM and SRAM in these areas.

3.2.2 Storage Attributes

The 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), cachability (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 U0 storage attribute) and byte ordering (the E storage attribute).

The PPC405CR provides two control mechanisms for the W, I, U0, G, and E attributes. Because the PPC405CR does not provide hardware support for multiprocessor environments, the M storage attribute, when present, has no effect.

When the PPC405CR operates in virtual mode (address translation is enabled), each storage attribute is controlled by the W, I, U0, G, and E fields in the translation lookaside buffer (TLB) entry for each memory page. The size of memory pages, and hence the size of storage attribute control regions, is variable. Multiple sizes can be in effect simultaneously on different pages.

When the PPC405CR operates in real mode (address translation is disabled), storage attribute control registers control the corresponding storage attributes. These registers are:

- Data Cache Write-through Register (DCWR)
- Data Cache Cachability Register (DCCR)
- Instruction Cache Cachability Register (ICCR)
- Storage Guarded Register (SGR)
- Storage Little-Endian Register (SLER)

- Storage User-defined 0 Register (SU0R)

Each storage attribute control register contains 32 bits; each bit controls one of thirty-two 128MB storage attribute control regions. Bit 0 of each register controls the lowest-order region, with ascending bits controlling ascending regions in memory. The storage attributes in each storage attribute region are set independently of each other and of the storage attributes for other regions.

3.3 Registers

All PPC405CR registers are listed in this section. Some of the frequently-used registers are described in detail. Other registers are covered in their respective topic chapters (for example, the cache registers are described in Chapter 4, “Cache Operations”). All registers are summarized in Chapter 21, “Register Summary.”

The registers are grouped into categories: General Purpose Registers (GPRs), Special Purpose Registers (SPRs), Time Base Registers (TBRs), the Machine State Register (MSR), the Condition Register (CR), Device Control Registers (DCRs), and memory-mapped I/O registers (MMIO). Different instructions are used to access each category of registers.

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.

> **Programming Note:** 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.

Figure 3-1 on page 3-4 illustrates the registers in the user and supervisor programming models.

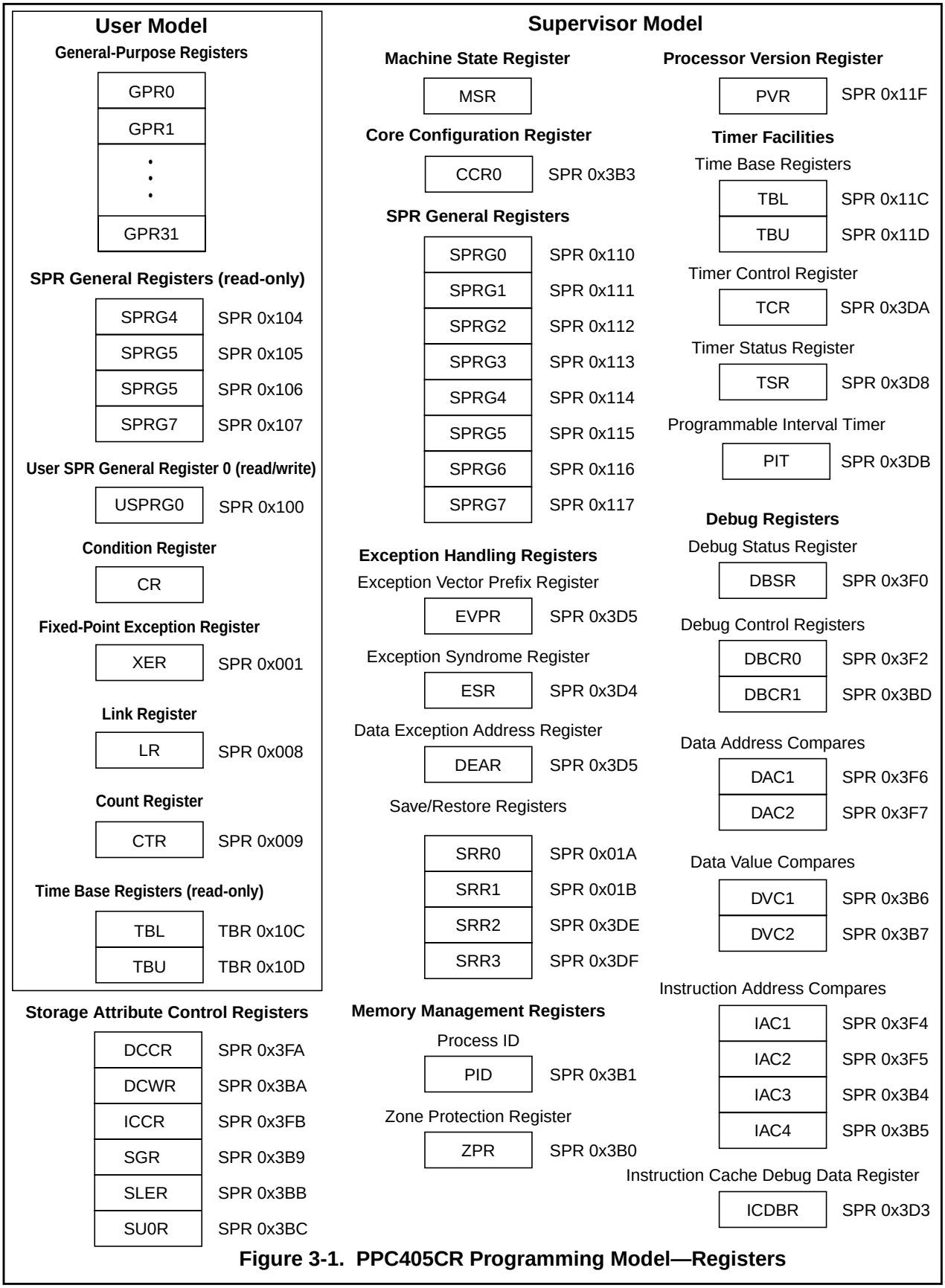


Figure 3-1. PPC405CR Programming Model—Registers

3.3.1 General Purpose Registers (R0-R31)

The PPC405CR contains thirty-two 32-bit general purpose registers (GPRs). Data from memory can be read into GPRs using load instructions and the contents of GPRs can be written to memory using store instructions. Most integer instructions use GPRs for source and destination operands. See Table 21.3, “General Purpose Registers,” on page 21-1 for the numbering of the GPRs.

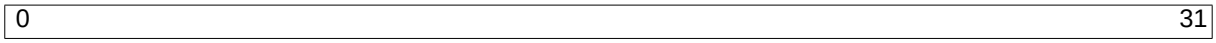


Figure 3-2. General Purpose Registers (R0-R31)

0:31		General Purpose Register data

3.3.2 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 operation of debug facilities, timers, interrupts, storage control attributes, and other architected processor resources. Table 21.5, “Special Purpose Registers,” on page 21-1 shows the mnemonic, name, and number for each SPR. Table 3-2, “PPC405CR SPRs,” on page 3-6 lists the PPC405CR SPRs by function and indicates the pages where the SPRs are described more fully.

Except for the Link Register (LR), the Count Register (CTR), the Fixed-point Exception Register (XER), User SPR General 0 (USPRG0, and read access to SPR General 4–7 (SPRG4–SPRG7), all SPRs are privileged. As SPRs, the registers TBL and TBU are privileged write-only; as TBRs, these registers can be read in user mode. Unless used to access non-privileged SPRs, attempts to execute **mfspr** and **mtspr** instructions while in user mode cause privileged violation program interrupts. See “Privileged SPRs” on page 3-38.

Table 3-2. PPC405CR SPRs

Function	Register				Access	Page
Configuration	CCR0				Privileged	4-11
Branch Control	CTR				User	3-6
	LR				User	3-7
Debug	DAC1	DAC2			Privileged	11-14
	DBCR0	DBCR1			Privileged	11-9
	DBSR				Privileged	11-12
	DVC1	DVC2			Privileged	11-15
	IAC1	IAC2	IAC3	IAC4	Privileged	11-14
	ICDBDR				Privileged	4-14
Fixed-point Exception	XER				User	3-7
General-Purpose SPR	SPRG0	SPRG1	SPRG2	SPRG3	Privileged	3-10
	SPRG4	SPRG5	SPRG6	SPRG7	User read, privileged write	3-10
	USPRG0				User	3-10
Interrupts and Exceptions	DEAR				Privileged	9-28
	ESR				Privileged	9-26
	EVPR				Privileged	9-26
	SRR0	SRR1			Privileged	9-24
	SRR2	SRR3			Privileged	9-25
Processor Version	PVR				Privileged, read-only	3-11
Storage Attribute Control	DCCR				Privileged	5-17
	DCWR				Privileged	5-17
	ICCR				Privileged	5-17
	SGR				Privileged	5-17
	SLER				Privileged	5-17
	SU0R				Privileged	5-17
Timer Facilities	TBL	TBU			Privileged, write-only	10-2
	PIT				Privileged	10-2
	TCR				Privileged	10-9
	TSR				Privileged	10-8
Zone Protection	ZPR				Privileged	5-13

3.3.2.1 Count Register (CTR)

The CTR is written from a GPR using **mtspr**. The CTR contents can be used as a loop count that is decremented and tested by some branch instructions. Alternatively, the CTR contents can specify a target address for the **bcctr** instruction, enabling branching to any address.

The CTR is in the user programming model.

Figure 3-3. Count Register (CTR)

0:31		Count	Used as count for branch conditional with decrement instructions, or as address for branch-to-counter instructions.

3.3.2.2 Link Register (LR)

The LR is written from a GPR using **mtspr**, and by 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. Thus, the LR contents can be used as the return address for a subroutine that was called using the branch.

The LR contents can be used as a target address for the **bclr** instruction. This allows branching to any address.

When the LR contents represent an instruction address, $LR_{30:31}$ are assumed to be 0, because all instructions must be word-aligned. However, when LR is read using **mfspr**, all 32 bits are returned as written.

The LR is in the user programming model.

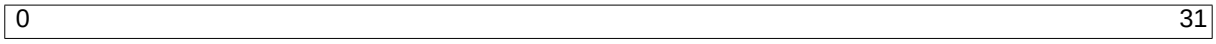


Figure 3-4. Link Register (LR)

0:31		Link Register contents	If (LR) represents an instruction address, $LR_{30:31}$ should be 0.

3.3.2.3 Fixed Point Exception Register (XER)

The XER records overflow and carry conditions generated by integer arithmetic instructions.

The Summary Overflow (SO) field is set to 1 when instructions cause the Overflow (OV) field to be set to 1. The SO field does not necessarily indicate that an overflow occurred on the most recent

arithmetic operation, but that an overflow occurred since the last clearing of XER[SO]. **mtspr**(XER) 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.

XER[OV] is set to indicate whether 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 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 in Chapter 20, "Instruction Set."

- Move instructions:

  mcrxr, **mtspr**(XER)

- Multiply and divide instructions:

  mullwo, **mullwo.**, **divwo**, **divwo.**, **divwuo, divwuo**

The Carry (CA) field is set to indicate whether 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 in Chapter 20, "Instruction Set."

- Move instructions

  mcrxr, **mtspr**(XER)

- Shift-algebraic operations

  sraw, srawi

The Transfer Byte Count (TBC) field is the byte count for load/store string instructions.

The XER is part of the user programming model.

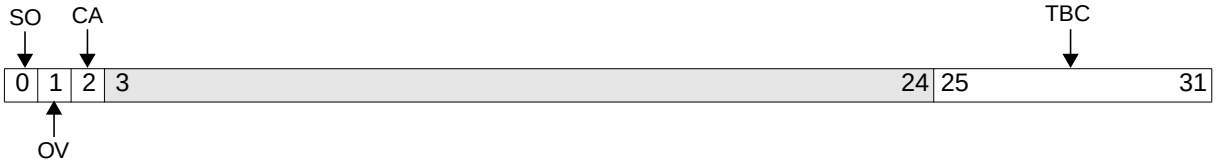


Figure 3-5. Fixed Point Exception Register (XER)

0	SO	Summary Overflow 0 No overflow has occurred. 1 Overflow has occurred.	Can be *set* by **mtspr** or by using “o” form instructions; 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 by using “o” form instructions; can be *reset* by **mtspr**, by **mcrxr**, or “o” form instructions.
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; 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**.

Table 3-3 and Table 3-4 list the PPC405CR instructions that update the XER. In the tables, the syntax “[**o**]” indicates that the instruction has an “o” form that updates XER[SO,OV], and a “non-o” form. The syntax “[**.**]” indicates that the instruction has a “record” form that updates CR[CR0] (see “Condition Register (CR)” on page 3-11), and a “non-record” form.

Table 3-3. XER[CA] Updating Instructions

Integer Arithmetic		**Integer Shift**	**Processor Control**
Add	**Subtract**	**Shift Right Algebraic**	**Register Management**
addc[**o**][**.**] **adde**[**o**][**.**] **addic**[**.**] **addme**[**o**][**.**] **addze**[**o**][**.**]	**subfc**[**o**][**.**] **subfe**[**o**][**.**] **subfic** **subfme**[**o**][**.**] **subfze**[**o**][**.**]	**sraw**[**.**] **srawi**[**.**]	**mtspr** **mcrxr**

Table 3-4. XER[SO,OV] Updating Instructions

Integer Arithmetic					Auxiliary Processor		Processor Control
Add	**Subtract**	**Multiply**	**Divide**	**Negate**	**Multiply-Accumulate**	**Negative Multiply-Accumulate**	**Register Management**
addo[.] **addco**[.] **addeo**[.] **addmeo**[.] **addzeo**[.]	**subfo**[.] **subfco**[.] **subfeo**[.] **subfmeo**[.] **subfzeo**[.]	**mullwo**[.]	**divwo**[.] **divwuo**[.]	**nego**[.]	**macchwo**[.] **macchwso**[.] **macchwsuo**[.] **macchwuo**[.] **machhwo**[.] **machhwso**[.] **machhwsuo**[.] **machhwuo**[.] **maclhwo**[.] **maclhwso**[.] **maclhwsuo**[.] **maclhwuo**[.]	**nmacchwo**[.] **nmacchwso**[.] **nmachhwo**[.] **nmachhwso**[.] **nmaclhwo**[.] **nmaclhwso**[.]	**mtspr** **mcrxr**

3.3.2.4 Special Purpose Register General (SPRG0–SPRG7)

USPRG0 and SPRG0–SPRG7 are provided for general purpose software use. For example, these registers are used as temporary storage locations. For example, an interrupt handler might save the contents of a GPR to an SPRG, and later restore the GPR from it. This is faster than a save/restore to a memory location. These registers are written using **mtspr** and read using **mfspr**.

Access to USPRG0 is non-privileged for both read and write.

SPRG0–SPRG7 provide temporary storage locations. For example, an interrupt handler might save the contents of a GPR to an SPRG, and later restore the GPR from it. This is faster than performing a save/restore to memory. These registers are written by **mtspr** and read by **mfspr**.

Access to SPRG0–SPRG7 is privileged, except for read access to SPRG4–SPRG7. See “Privileged SPRs” on page 3-38 for more information.

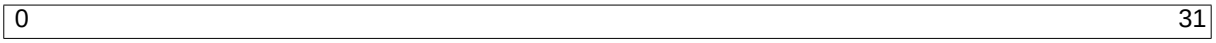


Figure 3-6. Special Purpose Register General (SPRG0–SPRG7)

0:31		General data	Software value; no hardware usage.

3.3.2.5 Processor Version Register (PVR)

The PVR is a read-only register that uniquely identifies a standard product or Core+ASIC implementation. Software can examine the PVR to recognize implementation-dependent features and determine available hardware resources.

Access to the PVR is privileged. See “Privileged SPRs” on page 3-38 for more information.

Figure 3-7. Processor Version Register (PVR)

0:31		Assigned PVR value

3.3.3 Condition Register (CR)

The CR contains eight 4-bit fields (CR0–CR7), as shown in Figure 3-8. The fields contain conditions detected during the execution of integer or logical compare instructions, as indicated in the instruction descriptions in Chapter 20, “Instruction Set.” The CR contents can be used in conditional branch instructions.

The CR can be modified in any of the following ways:

- **mtcrf** sets specified CR fields by writing to the CR from a GPR, under control of a mask specified as an instruction field.
- **mcrf** sets a specified CR field by copying another CR field to it.
- **mcrxr** copies certain bits of the XER into a designated CR field, and then clears the corresponding XER bits.
- The “with update” forms of integer instructions implicitly update CR[CR0].
- Integer compare instructions update a specified CR field.
- The CR-logical instructions update a specified CR bit with the result of a logical operation on a specified pair of CR bit fields.
- Conditional branch instructions can test a CR bit as one of the branch conditions.

If a CR field is set by a compare instruction, the bits are set as described in “CR Fields after Compare Instructions.”

The CR is part of the user programming model.

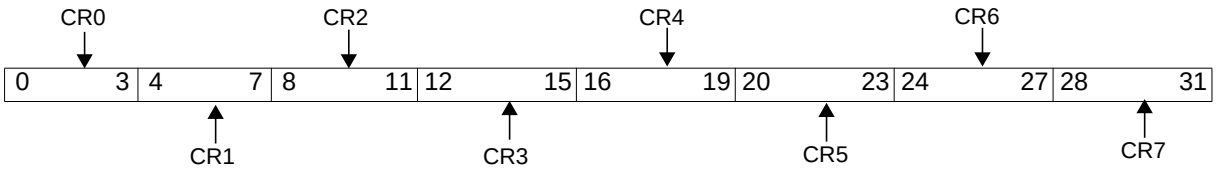


Figure 3-8. Condition Register (CR)

0:3	CR0	Condition Register Field 0
4:7	CR1	Condition Register Field 1
8:11	CR2	Condition Register Field 2
12:15	CR3	Condition Register Field 3
16:19	CR4	Condition Register Field 4
20:23	CR5	Condition Register Field 5
24:27	CR6	Condition Register Field 6
28:31	CR7	Condition Register Field 7

3.3.3.1 CR Fields after Compare Instructions

Compare instructions compare the values of two 32-bit registers. The two types of compare instructions, *arithmetic* and *logical*, are distinguished by the interpretation given to the 32-bit values. For *arithmetic* compares, the values are considered to be signed, where 31 bits represent the magnitude and the most-significant bit is a sign bit. For *logical* compares, the values are considered to be unsigned, so all 32 bits represent magnitude. There is no sign bit. As an example, consider the comparison of 0 with 0xFFFF FFFF. In an *arithmetic* compare, 0 is larger, because 0xFFFF FFFF represents –1; in a *logical* compare, 0xFFFF FFFF is larger.

A compare instruction can direct its CR update to any 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 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 (page 20-34 through page 20-37) 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].

3.3.3.2 The CR0 Field

After the execution of compare instructions that update CR[CR0], CR[CR0] is interpreted as described in “CR Fields after Compare Instructions” on page 3-12. The “dot” forms of arithmetic and logical instructions also alter CR[CR0]. After most 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 0; 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, either LT or GT is set, depending on the value of the most-significant bit of the 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 not actually an arithmetic result.

If an arithmetic overflow occurs, the “sign” of an instruction result indicated in $CR[CR0]_{LT, GT, EQ}$ 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].

3.3.4 The Time Base

The PowerPC Architecture provides a 64-bit time base. “Time Base” on page 10-2 describes the architected time base. Access to the time base is through two 32-bit time base registers (TBRs). The least-significant 32 bits of the time base are read from the Time Base Lower (TBL) register and the most-significant 32 bits are read from the Time Base Upper (TBU) register.

User-mode access to the time base is read-only, and there is no explicitly privileged read access to the time base.

The **mftb** instruction reads from TBL and TBU. Writing the time base is accomplished by moving the contents of a GPR to a pair of SPRs, which are also called TBL and TBU, using **mtspr**.

Table 3-5 shows the mnemonics and names of the TBRs.

Table 3-5. Time Base Registers

Mnemonic	Register Name	Access
TBL	Time Base Lower (Read-only)	Read-only
TBU	Time Base Upper (Read-only)	Read-only

3.3.5 Machine State Register (MSR)

The Machine State Register (MSR) controls processor core functions, such as the enabling or disabling of interrupts and address translation.

The MSR is written from a GPR using the **mtmsr** instruction. The contents of the MSR can be read into a GPR using the **mfmsr** instruction. MSR[EE] is set or cleared using the **wrtee** or **wrteei** instructions.

The MSR contents are automatically saved, altered, and restored by the interrupt-handling mechanism. See “Machine State Register (MSR)” on page 9-23.

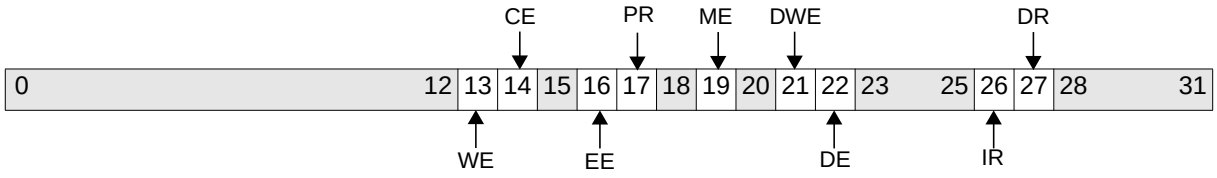


Figure 3-9. Machine State Register (MSR)

0:12		Reserved	
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 interrupt 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 time-out interrupts.
15		Reserved	

16	EE	External Interrupt Enable 0 Asynchronous interrupts (external to the processor core) are disabled. 1 Asynchronous interrupts are enabled.	Controls the non-critical external interrupt input, PIT, and FIT 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 interrupts are disabled. 1 Machine check interrupts are enabled.	
20		Reserved	
21	DWE	Debug Wait Enable 0 Debug wait mode is disabled. 1 Debug wait mode is enabled.	
22	DE	Debug Interrupts Enable 0 Debug interrupts are disabled. 1 Debug interrupts are enabled.	
23:25		Reserved	
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.	
27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.	
28:31		Reserved	

3.3.6 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).

DCRs are used to control the operations of on-chip buses, peripherals, and some processor behavior.

Table 3-6. Directly Accessed DCRs

Register	DCR Number	Access	Description
DCRs Used for Indirect Access			

Table 3-6. Directly Accessed DCRs

Register	DCR Number	Access	Description
SDRAM0_CFGADDR	0x010	R/W	Memory Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	Memory Controller Data Register
EBC0_CFGADDR	0x012	R/W	Peripheral Controller Address Register
EBC0_CFGDATA	0x013	R/W	Peripheral Controller Data Register
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register
On-Chip Buses			
PLB0_BESR	0x084	R/Clear	PLB Bus Error Status Register
PLB0_BEAR	0x086	R/W	PLB Bus Error Address Register
PLB0_ACR	0x087	R/W	PLB Arbiter Control Register
POB0_BESR0	0x0A0	R/Clear	PLB to OPB Bus Error Status Register 0
POB0_BEAR	0x0A2	R	PLB to OPB Bus Error Address Register
POB0_BESR1	0x0A4	R/Clear	PLB to OPB Bus Error Status Register 1
Clocking, Power Management, and Chip Control			
CPC0_PLLMR	0x0B0	R/W	PLL Mode Register
CPC0_CR0	0x0B1	R/W	Chip Control Register 0
CPC0_CR1	0x0B2	R/W	Chip Control Register 1
CPC0_PSR	0x0B4	R	Chip Pin Strapping Register
CPC0_JTAGID	0x0B5	R	JTAG ID Register
CPC0_SR	0x0B8	R	CPM Status Register
CPC0_ER	0x0B9	R/W	CPM Enable Register
CPC0_FR	0x0BA	R/W	CPM Force Register
Universal Interrupt Controllers			
UIC0_SR	0x0C0	R/Clear	UIC0 Status Register
UIC0_ER	0x0C2	R/W	UIC0 Enable Register
UIC0_CR	0x0C3	R/W	UIC0 Critical Register
UIC0_PR	0x0C4	R/W	UIC0 Polarity Register
UIC0_TR	0x0C5	R/W	UIC0 Triggering Register
UIC0_MSR	0x0C6	R	UIC0 Masked Status Register
UIC0_VR	0x0C7	R	UIC0 Vector Register
UIC0_VCR	0x0C8	W	UIC0 Vector Configuration Register
Direct Memory Access			
DMA0_CR0	0x100	R/W	DMA Channel Control Register 0
DMA0_CT0	0x101	R/W	DMA Count Register 0
DMA0_DA0	0x102	R/W	DMA Destination Address Register 0
DMA0_SA0	0x103	R/W	DMA Source Address Register 0
DMA0_SG0	0x104	R/W	DMA Scatter/Gather Descriptor Address Register 0
DMA0_CR1	0x108	R/W	DMA Channel Control Register 1
DMA0_CT1	0x109	R/W	DMA Count Register 1
DMA0_DA1	0x10A	R/W	DMA Destination Address Register 1
DMA0_SA1	0x10B	R/W	DMA Source Address Register 1

Table 3-6. Directly Accessed DCRs

Register	DCR Number	Access	Description
DMA0_SG1	0x10C	R/W	DMA Scatter/Gather Descriptor Address Register 1
DMA0_CR2	0x110	R/W	DMA Channel Control Register 2
DMA0_CT2	0x111	R/W	DMA Count Register 2
DMA0_DA2	0x112	R/W	DMA Destination Address Register 2
DMA0_SA2	0x113	R/W	DMA Source Address Register 2
DMA0_SG2	0x114	R/W	DMA Scatter/Gather Descriptor Address Register 2
DMA0_CR3	0x118	R/W	DMA Channel Control Register 3
DMA0_CT3	0x119	R/W	DMA Count Register 3
DMA0_DA3	0x11A	R/W	DMA Destination Address Register 3
DMA0_SA3	0x11B	R/W	DMA Source Address Register 3
DMA0_SG3	0x11C	R/W	DMA Scatter/Gather Descriptor Address
DMA0_SR	0x120	R/Clear	DMA Status Register
DMA0_SGC	0x123	R/W	DMA Scatter/Gather Command Register
DMA0_SLP	0x125	R/W	DMA Sleep Mode Register
DMA0_POL	0x126	R/W	DMA Polarity Configuration Register

3.3.6.1 Indirectly Accessed DCRs

The DCRs for the SDRAM controller, external bus controller (EBC), and decompression controller are indirectly accessed.

3.3.7 Indirect Access of SDRAM Controller DCRs

The following procedure accesses the SDRAM controller DCRs listed in Table 3-7.

1. Write the offset from Table 3-8 to the Memory Controller Address Register (SDRAM0_CFGADDR).
2. Read data from or write data to the Memory Controller Data Register (SDRAM0_CFGDATA).

Table 3-7. SDRAM Controller DCR Usage

Register	DCR Number	Access	Description
SDRAM0_CFGADDR	0x010	R/W	Memory Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	Memory Controller Data Register

Table 3-8. Offsets for SDRAM Controller Registers

Register	Offset	R/W	Description
SDRAM0_BESR0	0x00	R/Clear	Bus Error Syndrome Register 0
SDRAM0_BESR1	0x08	R/Clear	Bus Error Syndrome Register 1
SDRAM0_BEAR	0x10	R/W	Bus Error Address Register
SDRAM0_CFG	0x20	R/W	Memory Controller Options 1
SDRAM0_STATUS	0x24	R	SDRAM Controller Status

Table 3-8. Offsets for SDRAM Controller Registers

Register	Offset	R/W	Description
SDRAM0_RTR	0x30	R/W	Refresh Timer Register
SDRAM0_PMIT	0x34	R/W	Power Management Idle Timer
SDRAM0_B0CR	0x40	R/W	Memory Bank 0 Configuration Register
SDRAM0_B1CR	0x44	R/W	Memory Bank 1 Configuration Register
SDRAM0_B2CR	0x48	R/W	Memory Bank 2 Configuration Register
SDRAM0_B3CR	0x4C	R/W	Memory Bank 3 Configuration Register
SDRAM0_TR	0x80	R/W	SDRAM Timing Register 1
SDRAM0_ECCCFG	0x94	R/W	ECC Configuration
SDRAM0_ECCESR	0x98	R/Clear	ECC Error Status Register

3.3.8 Indirect Access of External Bus Controller DCRs

The following procedure accesses the EBC DCRs listed in Table 3-9.

1. Write the offset from Table 3-10 to the Peripheral Controller Address Register (EBC0_CFGADDR).
2. Read data from or write data to the Peripheral Controller Data Register (EBC0_CFGDATA).

Table 3-9. External Bus Controller DCR Usage

Register	DCR Number	Access	Description
EBC0_CFGADDR	0x012	R/W	Peripheral Controller Address Register
EBC0_CFGDATA	0x013	R/W	Peripheral Controller Data Register

Table 3-10. Offsets for External Bus Controller Registers

Register	Offset	Access	Description
EBC0_B0CR	0x00	R/W	Peripheral Bank 0 Configuration Register
EBC0_B1CR	0x01	R/W	Peripheral Bank 1 Configuration Register
EBC0_B2CR	0x02	R/W	Peripheral Bank 2 Configuration Register
EBC0_B3CR	0x03	R/W	Peripheral Bank 3 Configuration Register
EBC0_B4CR	0x04	R/W	Peripheral Bank 4 Configuration Register
EBC0_B5CR	0x05	R/W	Peripheral Bank 5 Configuration Register
EBC0_B6CR	0x06	R/W	Peripheral Bank 6 Configuration Register
EBC0_B7CR	0x07	R/W	Peripheral Bank 7 Configuration Register
EBC0_B0AP	0x10	R/W	Peripheral Bank 0 Access Parameters
EBC0_B1AP	0x11	R/W	Peripheral Bank 1 Access Parameters
EBC0_B2AP	0x12	R/W	Peripheral Bank 2 Access Parameters
EBC0_B3AP	0x13	R/W	Peripheral Bank 3 Access Parameters
EBC0_B4AP	0x14	R/W	Peripheral Bank 4 Access Parameters
EBC0_B5AP	0x15	R/W	Peripheral Bank 5 Access Parameters
EBC0_B6AP	0x16	R/W	Peripheral Bank 6 Access Parameters
EBC0_B7AP	0x17	R/W	Peripheral Bank 7 Access Parameters
EBC0_BEAR	0x20	R/W	Peripheral Bus Error Address Register

Table 3-10. Offsets for External Bus Controller Registers

Register	Offset	Access	Description
EBC0_BESR0	0x21	R/W	Peripheral Bus Error Status Register 0
EBC0_BESR1	0x22	R/W	Peripheral Bus Error Status Register 1
EBC0_CFG	0x23	R/W	External Peripheral Control Register

3.3.9 Indirect Access of Decompression Controller DCRs

The following procedure accesses the decompression controller DCRs listed in Table 3-11.

1. Write the offset from Table 3-12 to the Decompression Controller Address Register (DCP0_CFGADDR).
2. Read data from or write data to the Decompression Controller Data Register (DCP0_CFGDATA).

Table 3-11. Decompression Controller DCR Usage

Register	DCR Number	Access	Description
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register

Table 3-12. Offsets for Decompression Controller Registers

Register	Offset	R/W	Description
DCP0_ITOR0	0x00	R/W	Index Table Origin Register 0
DCP0_ITOR1	0x01	R/W	Index Table Origin Register 1
DCP0_ITOR2	0x02	R/W	Index Table Origin Register 2
DCP0_ITOR3	0x03	R/W	Index Table Origin Register 3
DCP0_ADDR0	0x04	R/W	Address Decode Definition Register 0
DCP0_ADDR1	0x05	R/W	Address Decode Definition Register 1
DCP0_CFG	0x40	R/W	Decompression Controller Configuration Register
DCP0_ID	0x41	R	Decompression Controller ID Register
DCP0_VER	0x42	R	Decompression Controller Version Number Register
DCP0_PLBBEAR	0x50	R	Bus Error Address Register (PLB address)
DCP0_MEMBEAR	0x51	R	Bus Error Address Register (DCP to EBC address)
DCP0_ESR	0x52	R/Clear	Bus Error Status Register 0 (masters 0-3)
DCP0_RAM0–DCP0_RAM3FF	0x400–0x7FF	R/W	Decode Tables

3.3.10 Memory-Mapped Input/Output Registers

Some registers associated with on-chip peripherals are memory-mapped input/output (MMIO) registers. Such registers are mapped into the system memory space and are accessed using load/store instructions.

3.3.10.1 Directly Accessed MMIO Registers

Directly-accessed MMIO registers are accessed using load/store instructions that contain the register addresses. Table 3-13 lists the directly-accessed MMIO registers.

Table 3-13. Directly Accessed MMIO Registers

Register	Address	Access	Description
Serial Ports			
UART0_RBR	0xEF600300	R	UART 0 Receiver Buffer Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_THR		W	UART 0 Transmitter Holding Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_DLL		R/W	UART 0 Baud-rate Divisor Latch LSB **Note:** Set UART0_LCR[DLAB] = 1 to access.
UART0_IER	0xEF600301	R/W	UART 0 Interrupt Enable Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_DLM		R/W	UART 0 Baud-rate Divisor Latch MSB **Note:** Set UART0_LCR[DLAB] = 1 to access.
UART0_IIR	0xEF600302	R	UART 0 Interrupt Identification Register
UART0_FCR	0xEF600302	W	UART 0 FIFO Control Register
UART0_LCR	0xEF600303	R/W	UART 0 Line Control Register
UART0_MCR	0xEF600304	R/W	UART 0 Modem Control Register
UART0_LSR	0xEF600305	R/W	UART 0 Line Status Register
UART0_MSR	0xEF600306	R/W	UART 0 Modem Status Register
UART0_SCR	0xEF600307	R/W	UART 0 Scratch Register
UART1_RBR	0xEF600400	R	UART 1 Receiver Buffer Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_THR		W	UART 1 Transmitter Holding Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_DLL		R/W	UART 1 Baud-rate Divisor Latch LSB **Note:** Set UART1_LCR[DLAB] = 1 to access.
UART1_IER	0xEF600401	R/W	UART 1 Interrupt Enable Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_DLM		R/W	UART 1 Baud-rate Divisor Latch MSB **Note:** Set UART1_LCR[DLAB] = 1 to access.
UART1_IIR	0xEF600402	R	UART 1 Interrupt Identification Register
UART1_FCR	0xEF600402	W	UART 1 FIFO Control Register
UART1_LCR	0xEF600403	R/W	UART 1 Line Control Register
UART1_MCR	0xEF600404	R/W	UART 1 Modem Control Register
UART1_LSR	0xEF600405	R/W	UART 1 Line Status Register
UART1_MSR	0xEF600406	R/W	UART 1 Modem Status Register
UART1_SCR	0xEF600407	R/W	UART 1 Scratch Register

Table 3-13. Directly Accessed MMIO Registers

Register	Address	Access	Description
Inter-Integrated Circuit			
IIC0_MDBUF	0xEF600500	R/W	IIC0 Master Data Buffer
IIC0_SDBUF	0xEF600502	R/W	IIC0 Slave Data Buffer
IIC0_LMADR	0xEF600504	R/W	IIC0 Low Master Address
IIC0_HMADR	0xEF600505	R/W	IIC0 High Master Address
IIC0_CNTL	0xEF600506	R/W	IIC0 Control
IIC0_MDCNTL	0xEF600507	R/W	IIC0 Mode Control
IIC0_STS	0xEF600508	R/W	IIC0 Status
IIC0_EXTSTS	0xEF600509	R/W	IIC0 Extended Status
IIC0_LSADR	0xEF60050A	R/W	IIC0 Low Slave Address
IIC0_HSADR	0xEF60050B	R/W	IIC0 High Slave Address
IIC0_CLKDIV	0xEF60050C	R/W	IIC0 Clock Divide
IIC0_INTRMSK	0xEF60050D	R/W	IIC0 Interrupt Mask
IIC0_XFRCNT	0xEF60050E	R/W	IIC0 Transfer Count
IIC0_XTCNTLSS	0xEF60050F	R/W	IIC0 Extended Control and Slave Status
IIC0_DIRECTCNTL	0xEF600510	R/W	IIC0 Direct Control
OPB Arbiter			
OPBA0_PR	0xEF600600	R/W	OPB Arbiter Priority Register
OPBA0_CR	0xEF600601	R/W	OPB Arbiter Control Register
General-Purpose I/O			
GPIO0_OR	0xEF600700	R/W	GPIO0_IRO Output Register
GPIO0_TCR	0xEF600704	R/W	GPIO0_IRO Three-State Control Register
GPIO0_ODR	0xEF600718	R/W	GPIO0_IRO Open Drain Register
GPIO0_IR	0xEF60071C	R	GPIO0_IRO Input Register

3.4 Data Types and Alignment

The data types consist of bytes (eight bits), halfwords (two bytes), words (four bytes), and strings (1 to 128 bytes). Figure 3-10 shows the byte, halfword, and word data types and their bit and byte definitions for big endian representations of values. Note that PowerPC bit numbering is reversed from industry conventions; bit 0 represents the most significant bit of a value.

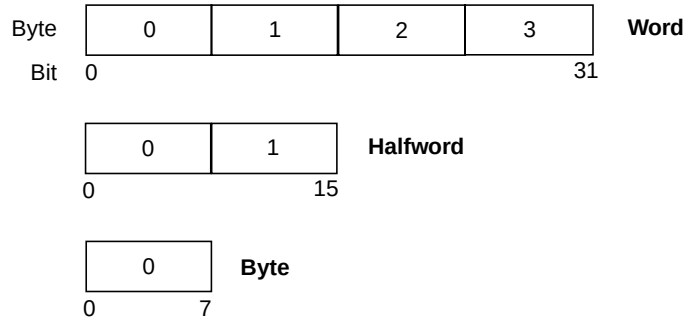


Figure 3-10. PPC405CR Data Types

Data is represented in either 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 words, and are word-aligned (the lowest byte address is divisible by 4).

3.4.1 Alignment for Storage Reference and Cache Control Instructions

The storage reference instructions (loads and stores; see Table 3-21, “Storage Reference Instructions,” on page 3-43) move data to and from storage. The data cache control instructions listed in Table 3-30, “Cache Management Instructions,” on page 3-47, 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 Chapter 20, “Instruction Set,” for more information.

Cache control instructions ignore the four least significant bits of the EA; no alignment restrictions exist in the DCU because of EAs. However, storage control attributes can cause alignment exceptions. When data address translation is disabled and a **dcbz** instruction references a storage region that is non-cachable, or for which write-through caching is the write strategy, an alignment exception is taken. Such exceptions result from the storage control attributes, not from EA alignment. The alignment exception enables system software to emulate the write-through function.

Alignment requirements for the storage reference instructions and the **dcread** instruction depend on the particular instruction. Table 3-14, “Alignment Exception Summary,” on page 3-23, 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 word instructions have word targets, word-aligned.
- Load/ store halfword instructions have halfword targets, halfword-aligned.
- Load/store byte instructions have byte targets, byte-aligned (that is, any alignment).

Misalignments are addresses that are not naturally aligned on data type boundaries. An address not divisible by four is misaligned with respect to word instructions. An address not divisible by two is misaligned with respect to halfword instructions. The PPC405CR implementation handles misalignments within and across word boundaries, but there is a performance penalty because additional bus cycles are required.

3.4.2 Alignment and Endian Operation

The endian storage control attribute does not affect alignment behavior. In little endian storage regions, the alignment of data is treated as it is in big endian storage regions; 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.

3.4.3 Summary of Instructions Causing Alignment Exceptions

Table 3-14 summarizes the instructions that cause alignment exceptions and the conditions under which the alignment exceptions occur.

Table 3-14. Alignment Exception Summary

Instructions Causing Alignment Exceptions	Conditions
dcbz	EA in non-cachable or write-through storage
dcread, **lwarx**, **stwcx**.	EA not word-aligned

3.5 Byte Ordering

The following discussion describes the “endianness” of the PPC405CR core, which, by default and in normal use is “big endian.” The PPC405CR also contains “little endian” peripherals and supports the attachment of external little endian peripherals.

If scalars (individual data items and instructions) were indivisible, “byte ordering” would not be a concern. It is meaningless to consider the order of bits or groups of bits within a byte, 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 byte order arise.

For a machine in which the smallest addressable unit of storage is the 32-bit word, there is no question of the ordering of bytes within words. All transfers of individual scalars between registers and storage are of words, and the address of the byte containing the high-order eight bits of a scalar is the same as the address of any other byte of the scalar.

For the PowerPC Architecture, as for most computer architectures currently implemented, the smallest addressable unit of storage is the 8-bit byte. Other scalars are halfwords, words, or doublewords, which consist of groups of bytes. When a word-length scalar is moved from a register to storage, the scalar is stored in four consecutive byte addresses. It thus becomes meaningful to discuss the order of the byte addresses with respect to the value of the scalar: that is, which byte contains the highest-order eight bits of the scalar, which byte contains the next-highest-order eight bits, and so on.

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 a word, but only two of these orderings are commonly used:

- 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 on.

  This ordering is called *big endian* because the “big end” of the scalar, considered as a binary number, comes first in storage.

- 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 on.

  This ordering is called *little endian* because the “little end” of the scalar, considered as a binary number, comes first in storage.

3.5.1 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 word */
        long long b;    /* 0x2122_2324_2526_2728 doubleword */
        char *c;        /* 0x3132_3334 word */
        char d[7];      /* 'A','B','C','D','E','F','G' array of bytes */
        short e;        /* 0x5152 halfword */
        int f;          /* 0x6162_6364 word */
} 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.

3.5.1.1 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

3.5.1.2 Little Endian Mapping

Structure *s* is shown mapped little endian.

14	**13**	**12**	**11**				
0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
28	**27**	**26**	**25**	**24**	**23**	**22**	**21**
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
34	**33**	**32**	**31**	**'A'**	**'B'**	**'C'**	**'D'**
0x10	0x11	0x12	0x13	0x14	0x15	0x16	0x17
'E'	**'F'**	**'G'**		**52**	**51**		
0x18	0x19	0x1A	0x1B	0x1C	0x1D	0x1E	0x1F
64	**63**	**62**	**61**				
0x20	0x21	0x22	0x23	0x24	0x25	0x26	0x27

3.5.2 Support for Little Endian Byte Ordering

Except as noted, this book describes the processor as if it operated only in a big endian fashion. In fact, the IBM PowerPC Embedded Environment also supports little endian operation.

The PowerPC little endian mode, defined in the PowerPC Architecture, is not implemented.

3.5.3 Endian (E) Storage Attribute

The endian (E) storage attribute supports direct connection of the PPC405CR to little endian peripherals and to memory containing little endian instructions and data. For every storage reference (instruction fetch or load/store access), an E storage attribute is associated with the storage region of the reference. The E attribute specifies whether that region is organized as big endian (E = 0) or little endian (E = 1).

When address translation is enabled (MSR[IR] = 1 or MSR[DR] = 1), the E field in the corresponding TLB entry controls the endianness of a memory region. When address translation is disabled (MSR[IR] = 0 or MSR[DR] = 0), the SLER controls the endianness of a memory region.

Bytes in storage that are accessed as little endian are arranged in true little endian format. The PPC405CR does not support the little endian mode defined in the PowerPC architecture and used in PPC401xx and PPC403xx processors. Furthermore, no address modification is performed when accessing storage regions programmed as little endian. Instead, the PPC405CR reorders the bytes as they are transferred between the processor and memory.

The on-the-fly reversal of bytes 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 access (load/store). The following sections describe byte reordering for the two kinds of storage accesses.

3.5.3.1 Fetching Instructions from Little Endian Storage Regions

Instructions are words (four bytes) that are aligned on word boundaries in memory. As such, instructions in a big endian memory region are arranged with the most significant byte (MSB) of the instruction word at the lowest 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 the little endian mapping instruction *p* is arranged with the least significant byte (LSB) of the instruction word 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 instruction queue in the proper order. The execution unit assumes that the MSB of an instruction word is at the lowest address. Therefore, when instructions are fetched from little endian storage regions, the four bytes of an instruction word are reversed before the instruction is decoded. In the PPC405CR, the byte reversal occurs between memory and the instruction cache unit (ICU). The ICU always stores instructions in big endian format, regardless of whether the memory region containing the instruction is 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 storage region must be reloaded with program and data structures in the appropriate endian format. If the endian format of instruction memory changes, the ICU must be made coherent with the updates. The ICU must be invalidated and the updated instruction memory using the new endian format must be fetched so that the proper byte ordering occurs before the new instructions are placed in the ICU.

3.5.3.2 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, halfword, or word) 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 what type of byte reversal is required. Therefore, byte reversal during load/store accesses is performed between the DCU and the GPR.

When accessing data in a little endian storage region:

- For byte loads/stores, no reordering occurs.
- For halfword loads/stores, bytes are reversed within the halfword.
- For word loads/stores, bytes are reversed within the word.

Note that this applies, regardless of data alignment.

The big endian and little endian mappings of the structure *s*, shown in “Structure Mapping Examples” on page 3-24, demonstrate how the size of an item determines its byte ordering. For example:

- The word *a* has its four bytes reversed within the word spanning addresses 0x00–0x03.
- The halfword *e* has its two bytes reversed within the halfword spanning addresses 0x1C–0x1D.

Note that 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.

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.

3.5.3.3 PowerPC Byte-Reverse Instructions

For big endian storage regions, normal load/store instructions move the more significant bytes of a 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.

As Figure 3-11 through Figure 3-14 illustrate, a normal store to a big endian storage region is the same as a byte-reverse store to a little endian storage region. Conversely, a normal store to a little endian storage region is the same as a byte-reverse store to a big endian storage region.

Figure 3-11 illustrates the contents of a GPR and memory (starting at address 00) after a normal load/store in a big endian storage region.

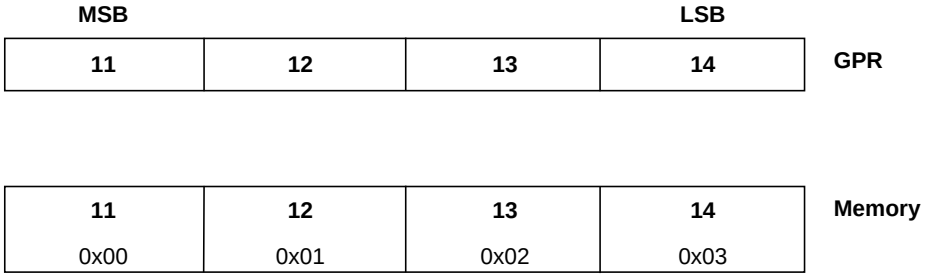


Figure 3-11. Normal Word Load or Store (Big Endian Storage Region)

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 3-12.

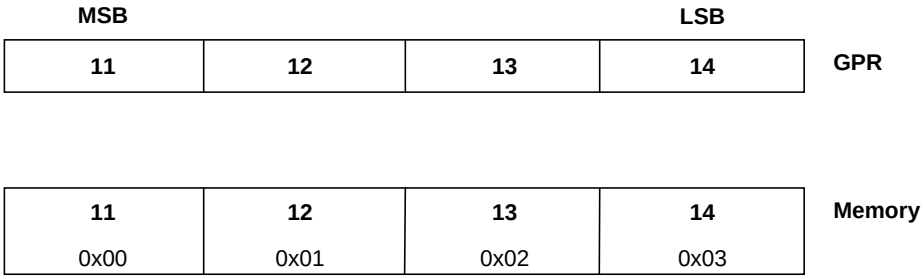


Figure 3-12. Byte-Reverse Word Load or Store (Little Endian Storage Region)

Figure 3-13 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.

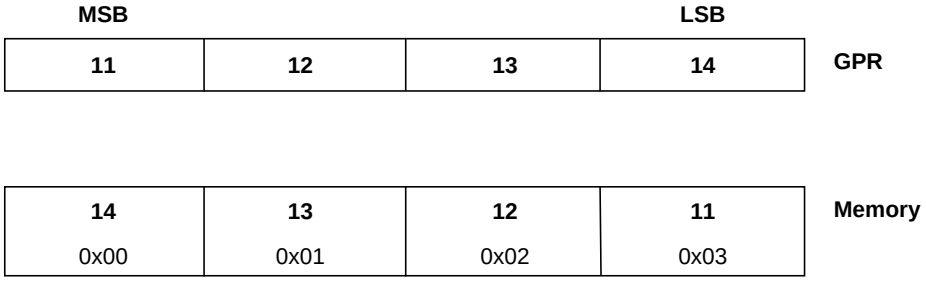


Figure 3-13. Byte-Reverse Word Load or Store (Big Endian Storage Region)

Note that the results are identical to the results of a normal load/store in a little endian storage region, as illustrated in Figure 3-14.

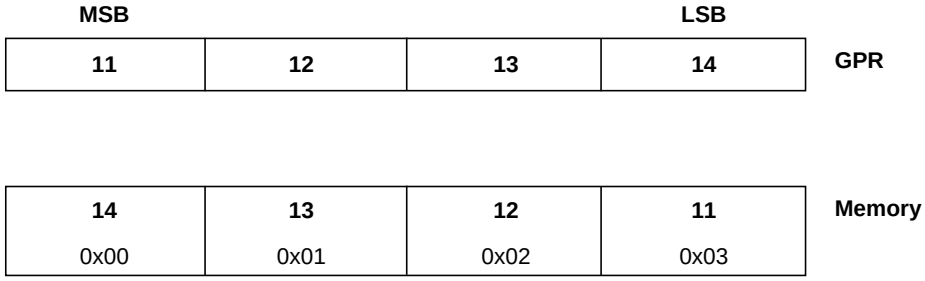


Figure 3-14. Normal Word Load or Store (Little Endian Storage Region)

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 storage region in little endian format.

  Only the endian storage attribute mechanism supports the fetching of 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 device drivers written in hand-coded assembler.

  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.

3.6 Instruction Processing

The instruction pipeline, illustrated in Figure 3-15, contains three queue locations: prefetch buffer 1 (PFB1), prefetch buffer 0 (PFB0), and decode (DCD). This queue implements a pipeline with the following functional stages: fetch, decode, execute, write-back and load write-back. Instructions are fetched from the instruction cache unit (ICU), placed in the instruction queue, and eventually dispatched to the execution unit (EXU).

Instructions are fetched from the ICU at the request of the EXU. Cachable instructions are forwarded directly to the instruction queue and stored in the ICU cache array. Non-cachable instructions are also forwarded directly to the instruction queue, but are not stored in the ICU cache array. Fetched instructions drop to the empty queue location closest to the EXU. When there is room in the queue, Instructions can be returned from the ICU two at a time. If the queue is empty and the ICU is returning two instructions, one instruction drops into DCD while the other drops into PFB0. PFB1 buffers instructions when the pipeline stalls.

Branch instructions are examined in DCD and PFB0 while all other instructions are decoded in DCD. All instructions must pass through DCD before entering the EXU. The EXU contains the execute, write-back and load write-back stages of the pipe. The results of most instructions are calculated during the execute stage and written to the GPR file during the write back stage. Load instructions write the GPR file during the load write-back stage.

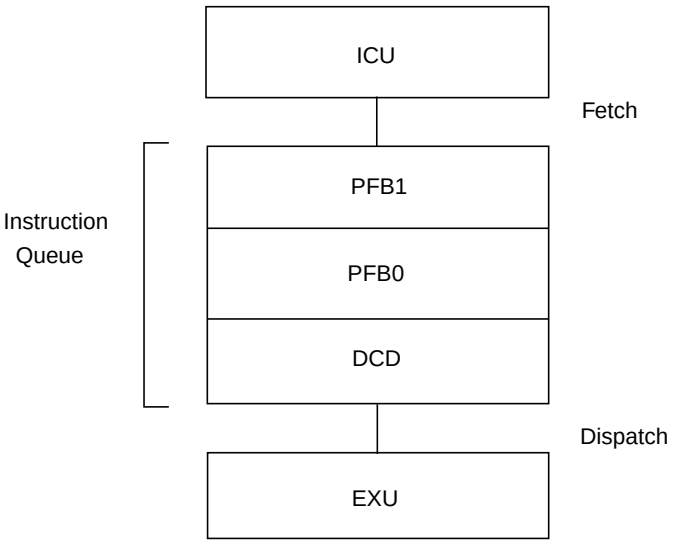


Figure 3-15. PPC405CR Instruction Pipeline

3.7 Branch Processing

The PPC405CR, which provides a variety of conditional and unconditional branching instructions, uses the branch prediction techniques described in “Branch Prediction” on page 3-35.

3.7.1 Unconditional Branch Target Addressing Options

The unconditional branches (**b**, **ba**, **bl**, **bla**) carry the displacement to the branch target address as a signed 26-bit value (the 24-bit LI field right-extended with 0b00). The displacement enables unconditional branches to cover an address range of ±32MB.

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 0 plus the signed displacement. If the sign bit (LI[0]) is 0, the displacement is the target address. If the sign bit is 1, the displacement is a negative value and wraps to the highest memory addresses. For example, if the displacement is 0x3FF FFFC (the 26-bit representation of –4), the target address is 0xFFFF FFFC (0 – 4B, or 4 bytes below the top of memory).

3.7.2 Conditional Branch Target Addressing Options

The conditional branches (**bc**, **bca**, **bcl**, **bcla**) carry the displacement to the branch target address as a signed 16-bit value (the 14-bit BD field right-extended with 0b00). The displacement enables conditional branches to cover an address range of ±32KB.

For the relative (AA = 0) forms (**bc**, **bcl**), the target address is the CIA plus the signed displacement.

For the absolute (AA = 1) forms (**bca**, **bcla**), the target address is 0 plus the signed displacement. If the sign bit (BD[0]) is 0, the displacement is the target address. If the sign bit is 1, the displacement is negative and wraps to the highest memory addresses. For example, if the displacement is 0xFFFC (the 16-bit representation of –4), the target address is 0xFFFF FFFC (0 – 4B, or 4 bytes from the top of memory).

3.7.3 Conditional Branch Condition Register Testing

Conditional branch instructions can test a CR bit. The value of the BI field specifies the bit to be tested (bit 0–31). The BO field controls whether the CR bit is tested, as described in the following section.

3.7.4 BO Field on Conditional Branches

The BO field of the conditional branch instruction specifies the conditions used to control branching, and specifies 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 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 $\neq$ 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 the CTR are left unchanged, and the CTR does not participate in the branch condition test.

Table 3-15 summarizes the usage of the bits of the BO field. BO[4] is further discussed in “Branch Prediction.”

Table 3-15. 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 Test for CR[BI] = 0. 1 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 Test for CTR $\neq$ 0. 1 Test for CTR = 0.
BO[4]	Branch Prediction Reversal 0 Apply standard branch prediction. 1 Reverse the standard branch prediction.

Table 3-16 lists specific BO field contents, and the resulting actions; *z* represents a mandatory value of 0, and *y* is a branch prediction option discussed in “Branch Prediction.”

Table 3-16. Conditional Branch BO Field

BO Value	Description
0000*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0 and CR[BI]=0.
0001*y*	Decrement the CTR, then branch if the decremented CTR = 0 and CR[BI] = 0.
001*zy*	Branch if CR[BI] = 0.
0100*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0 and CR[BI] = 1.
0101*y*	Decrement the CTR, then branch if the decremented CTR=0 and CR[BI] = 1.
011*zy*	Branch if CR[BI] = 1.
1*z*00*y*	Decrement the CTR, then branch if the decremented CTR ≠ 0.
1*z*01*y*	Decrement the CTR, then branch if the decremented CTR = 0.
1*z*1*zz*	Branch always.

3.7.5 Branch Prediction

Conditional branches present a problem to the instruction fetcher. A branch might be taken. The branch EXU attempts to predict whether or not 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 starting at the predicted branch target address. If the prediction is correct, time is saved because the branched-to instruction is available in the instruction queue. Otherwise, the instruction pipeline stalls while the correct instruction is fetched into the instruction queue. To be effective, branch prediction must be correct most of the time.

The PowerPC Architecture enables software to reverse the default 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 the sign bit of the displacement for conditional branch (**bc**) instructions, and 0 for **bclr** and **bcctr** instructions.

(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 default prediction for this case was decided by considering the relative form of **bc**, which is commonly used at the end of loops to control the number of times that a loop is executed. The branch is taken every 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.

If branch displacements are positive (*s* = 0), 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 “Unconditional Branch Target Addressing Options” on page 3-30, if the algebraic sign of the displacement is negative (*s* = 1), the branch target address is in high memory. If

the algebraic sign of the displacement is positive (s = 0), the branch target address 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 default prediction is taken, and for the low memory case the default prediction is not taken.

BO[4] is the *prediction reversal bit*. If BO[4] = 0, the default prediction is applied. If BO[4] = 1, the reverse of the standard prediction is applied. For the cases in Table 3-17 where BO[4] = *y*, software can reverse the default prediction. This should only be done when the default prediction is likely to be wrong. Note that for the “branch always” condition, reversal of the default prediction is not allowed.

The 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 set appropriately to force the branch to be predicted taken.

3.8 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.

Sometimes speculative accesses are inappropriate. 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 will be executed, the rest of the cache block can be speculatively accessed.)

3.8.1 Speculative Accesses in the PPC405CR

The PPC405CR does not perform speculative loads.

Two methods control speculative instruction fetching. If instruction address translation is enabled (MSR[IR] = 1), the G (guarded) field in the translation lookaside buffer (TLB) entries controls speculative accesses.

If instruction address translation is disabled (MSR[IR] = 0), the Storage Guarded Register (SGR) controls speculative accesses for regions of memory. When a region is guarded (speculative fetching is disallowed), instruction prefetching is disabled for that region. 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 that, following any reset, the PPC405CR operates with all of storage guarded.

Note that when address translation is enabled, attempts to access guarded storage result in instruction storage exceptions. Guarded memory is in most often needed with peripheral status registers that are cleared automatically after being read, because an unintended access resulting from a speculative fetch would cause the loss of status information. Because the MMU provides 64 pages with a wide range of page sizes as small as 1KB, fetching instructions from guarded storage should be unnecessary.

3.8.1.1 Prefetch Distance Down an Unresolved Branch Path

The fetcher will speculatively access 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 "Instruction Processing" on page 3-29 for a description of the instruction pipeline). 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 cachable.

If the address is non-cachable (as controlled by the I storage attribute), no more than two instructions are speculatively accessed.

3.8.1.2 Prefetch of Branches to the CTR and Branches to the LR

When the instruction fetcher predicts that a **bctr** or **blr** instruction will be taken, the fetcher does not attempt to fetch an instruction from the target address in the CTR or LR if an executing instruction updates the register ahead of the branch. (See "Instruction Processing" on page 3-29 for a description of the instruction pipeline). The fetcher recognizes that the CTR or LR contains data left from an earlier use and that such data is probably not valid.

In such cases, the fetcher does not fetch the instruction at the target address until the instruction that is updating the CTR or LR completes. Only then are the "correct" CTR or LR contents known. This prevents the fetcher from speculatively accessing a completely "random" address. After 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.

3.8.2 Preventing Inappropriate Speculative Accesses

A memory-mapped I/O peripheral, such as a serial port having a status register that is automatically reset when read provides a simple example of storage that should not be speculatively accessed. If code is in memory at an address adjacent to the peripheral (for example, code goes from 0x0000 0000 to 0x0000 0FFF, and the peripheral is at 0x0000 1000), prefetching past the end of the code will read the peripheral.

Guarding storage also prevents prefetching past the end of memory. If the highest memory address is left unguarded, the fetcher could attempt to fetch past the last valid address, potentially causing machine checks on the fetches from invalid addresses. While the machine checks do not actually cause an exception until the processor attempts to execute an instruction at an invalid address, some systems could suffer from the attempt to access such an invalid address. For example, an external memory controller might log an error.

System designers can avoid problems from speculative fetching without using the guarded storage attributes. The rest of this section describes ways to prevent speculative instruction fetches to sensitive addresses in unguarded memory regions.

3.8.2.1 Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction

Suppose a **bctr** or **blr** instruction closely 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 must insert an unconditional branch hang (**b $**) just after the **rfi**. This prevents the hardware from prefetching the invalid target address used by **bctr**.

Consider also the above code sequence, with the **rfi** instruction replaced by an **sc** instruction used 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 non-sensitive address into the CTR. This address will be 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.

3.8.2.2 Fetching Past tw or twi Instructions

The interrupt-causing instructions, **tw** and **twi**, do not require the special handling described in "Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction" on page 3-35. 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.

3.8.2.3 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 will not be speculatively fetched.

3.8.2.4 Suggested Locations of Memory-Mapped Hardware

The preferred method of protecting memory-mapped hardware from inadvertent access is to use address translation, with hardware isolated to guarded pages (the G storage attribute in the associated TLB entry is set to 1.) The pages can be as small as 1KB. Code should never be stored in such pages.

If address translation is disabled, the preferred protection method is to isolate memory-mapped hardware into regions guarded using the SGR. Code should never be stored in such regions. The disadvantage of this method, compared to the preferred method, is that each region guarded by the SGR consumes 128MB of the address space.

Table 3-17 shows two address regions of the PPC405CR. Suppose a system designer can map all I/O devices and all ROM and SRAM devices into any location in either region. The choices made by the designer can prevent speculative accesses to the memory-mapped I/O devices.

Table 3-17. Example Memory Mapping

0x7800 0000 – 0x7FFF FFFF (SGR bit 15)	128MB Region 2
0x7000 0000 – 0x77FF FFFF (SGR bit 14)	128MB Region 1

A simple way to avoid the problem of speculative reads to peripherals is to map all storage containing code into Region 2, and all I/O devices into Region 1. Thus, accesses to Region 2 would only be for code and program data. Speculative fetches occuring in Region 2 would never access addresses in Region 1. Note that this hardware organization eliminates the need to use of the G storage attribute to protect Region 1. However, Region 1 could be set as guarded with no performance penalty, because there is no code to execute or variable data to access in Region 1.

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. Prefetching from the highest addresses of Region 1 could cause an attempt to speculatively access the bottom of Region 2, but guarding prevents this from occurring. The performance penalty is slight, under the assumption that code infrequently executes the instructions in the highest addresses of Region 1.

3.8.3 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 “Fetching Past an Interrupt-Causing or Interrupt-Returning Instruction” on page 3-35.
- 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 “Fetching Past an Unconditional Branch” on page 3-35.
- Recognize that a maximum of five words (20 bytes) can be prefetched past an unresolved conditional branch, either down the target path or the sequential path. See “Prefetch Distance Down an Unresolved Branch Path” on page 3-34.

Of course, software should not code branches with known unsafe targets (either relative to the instruction counter, or to addresses contained in the LR or CTR), on the assumption that the targets

are “protected” by code guaranteeing that the unsafe direction is not taken. The fetcher assumes that if a branch is predicted to be taken, it is safe to fetch down the target path.

3.9 Privileged Mode Operation

In the PowerPC Architecture, several terms describe two operating modes that have different instruction execution privileges. When a processor is in “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:

Privileged	Non-privileged
Privileged Mode	User Mode
Supervisor State	Problem State

The architecture uses MSR[PR] to control 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).

After a reset, MSR[PR] = 0.

3.9.1 MSR Bits and Exception Handling

The current value of MSR[PR] is saved, along with all other MSR bits, in the SRR1 (for non-critical interrupts) or SRR3 (for critical interrupts) upon any interrupt, and MSR[PR] is set to 0. Therefore, all exception handlers operate in privileged mode.

Attempting to execute a privileged instruction while in user mode causes a privileged violation program exception (see “Program Interrupt” on page 9-35). The PPC405CR 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 PRR field of the Exception Syndrome Register (ESR) is set when an interrupt was caused by a privileged instruction program exception. Software is not required to clear ESR[PRR].

3.9.2 Privileged Instructions

The instructions listed in Table 3-18 are privileged and cannot be executed while in user mode (MSR[PR] = 1).

Table 3-18. Privileged Instructions

dcbi	
dccci	
dcread	
iccci	
icread	
mfdcr	
mfmsr	

Table 3-18. Privileged Instructions (continued)

mfspr	For all SPRs except CTR, LR, SPRG4–SPRG7, and XER. See “Privileged SPRs” on page 3-38
mtdcr	
mtmsr	
mtspr	For all SPRs except CTR, LR, XER. See “Privileged SPRs” on page 3-38
rfci	
rfi	
tlbia	
tlbre	
tlbsx	
tlbsync	
tlbwe	
wrtee	
wrteei	

3.9.3 Privileged SPRs

All SPRs are privileged, except for the LR, the CTR, the XER, USPRG0, and read access to SPRG4–SPRG7. Reading from the time base registers Time Base Lower (TBL) and Time Base Upper (TBU) is not privileged. These registers are read using the **mftb** instruction, rather than the **mfspr** instruction. TBL and TBU are written (with different addresses) using **mtspr**, which is privileged for these registers. 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 (0b0000011010), the most-significant bit is 0. However, the machine code for the instruction reverses the subfields, resulting in the following SPRF field: 0b1101000000. 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 non-privileged.

3.9.4 Privileged DCRs

The **mtdcr** and **mfdcr** instructions themselves are privileged, in all cases. All DCRs are privileged.

3.10 Synchronization

The PPC405CR supports the synchronization operations of the 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”

3.10.1 Context Synchronization

The context of a program is the environment (for example, privilege and address translation) 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 PPC405CR, these include any interrupt, except a non-recoverable instruction machine check, 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 PPC405CR, one must use either an **eieio** instruction or a **sync** instruction. Note that for the PPC405CR, the **eieio** and **sync** instructions are implemented identically. See “Storage Synchronization” on page 3-42.

The contents of DCRs are not considered as part of the processor “context” managed by a context synchronizing operation. DCRs are not part of the processor core, 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 pseudocode examples to illustrate these limitations of context synchronization. Subsequent text explains how software can further guarantee “storage ordering.”

1. Consider the following instruction sequence:

   STORE non-cachable 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 will execute; either the old version or the new (stored) version might.

2. Consider the following instruction sequence, which assumes that the PPC405CR uses DCRs to provide bus region control:

   STORE non-cachable to address XYZ
   isync
   MTDCR to change a bus region containing XYZ

   In this sequence, there is no guarantee that the STORE will occur before the **mtdcr** changing the bus region control DCR. The STORE could fail because of a configuration error.

Consider an interrupt that changes privileged mode. An interrupt is a context synchronizing operation, because interrupts cause the MSR to be updated. The MSR is part of the processor context; the context synchronizing operation guarantees that all instructions that precede the interrupt complete using the preinterrupt value of MSR[PR], and that all instructions that follow the interrupt complete using the postinterrupt value.

Consider, on the other hand, some code that uses **mtmsr** to change the value of MSR[PR], which changes the privileged mode. In this case, the MSR is changed, changing the context. It is possible, for example, that prefetched privileged instructions expect to execute after the **mtmsr** has changed the operating mode from privileged mode to user mode. To prevent privileged instruction program exceptions, the code must execute a context synchronization operation, such as **isync**, immediately after the **mtmsr** instruction to prevent further instruction execution until the **mtmsr** completes.

eieio or **sync** can ensure that the contents of memory and DCRs are synchronized in the instruction stream. These instructions guarantee storage ordering because all memory accesses that precede **eieio** or **sync** are completed before subsequent memory accesses. Neither **eieio** nor **sync** guarantee that instruction prefetching is delayed until the **eieio** or **sync** completes. The instructions do not cause the prefetch queues to be purged and instructions to be refetched. See “Storage Synchronization” on page 3-42 for more information.

Instruction cache state is part of context. A context synchronization operation is required to guarantee instruction cache access ordering.

3. Consider the following instruction sequence, which is required for creating 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**.

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.

isync guarantees that all subsequent instructions are fetched and executed using the context established by all previous instructions. **isync** is a context synchronizing operation; **isync** causes all subsequently prefetched instructions to be discarded and refetched.

The following example illustrates the use of **isync** with debug exceptions:

mtdbcr0	Enable an instruction address compare (IAC) event
isync	Wait for the new Debug Control Register 0 (DBCR0) context to be established
XYZ	This instruction is at the IAC address; an **isync** was necessary to guarantee that the IAC event occurs at the execution of this instruction

3.10.2 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 PPC405CR 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. (For example, 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**.

Note that the PowerPC Architecture requires MSR[EE] (the external interrupt bit) to be, in effect, execution synchronizing: if a **mtmsr** sets MSR[EE] = 1, 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 PPC405CR 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**.

3.10.3 Storage Synchronization

The **sync** instruction guarantees that all previous storage references complete with respect to the PPC405CR 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 peripheral
sync	Wait for store to actually complete
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 peripheral, address X; this resets a status bit in the device
eieio	Guarantee **stb** X completes before next instruction
lbz Y	Load from peripheral, address Y; this is the status register updated by **stb** X. **eieio** was necessary, because the read and write addresses are different, but affect each other

The PPC405CR 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 PPC405CR does not provide hardware support for multiprocessor memory coherency, so **sync** does not guarantee memory ordering across multiple processors.

3.11 Instruction Set

The PPC405CR instruction set contains instructions defined in the PowerPC Architecture and instructions specific to the IBM PowerPC 400 family of embedded processors.

Chapter 20, “Instruction Set,” contains detailed descriptions of each instruction.

Appendix A, “Instruction Summary,” alphabetically lists each instruction and extended mnemonic and provides a short-form description. Appendix B, “Instructions by Category,” provides short-form descriptions of instructions, grouped by the instruction categories listed in Table 3-19, “PPC405CR Instruction Set Summary,” on page 3-42.

Table 3-19 summarizes the PPC405CR instruction set functions by categories. Instructions within each category are described in subsequent sections.

Table 3-19. PPC405CR Instruction Set Summary

Storage Reference	load, store
Arithmetic	add, subtract, negate, multiply, multiply-accumulate, multiply halfword, divide
Logical	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	rotate and insert, rotate and mask, shift left, shift right

Table 3-19. PPC405CR Instruction Set Summary

Shift	shift left, shift right, shift right algebraic
Cache Management	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

3.11.1 Instructions Specific to IBM PowerPC Embedded Processors

To support functions required in embedded real-time applications, the IBM PowerPC 400 family of embedded processors defines instructions that are not defined in the PowerPC Architecture.

Table 3-20 lists the instructions specific to IBM PowerPC embedded processors. Programs using these instructions are not portable to PowerPC implementations that are not part of the IBM PowerPC 400 family of embedded processors.

In the table, the syntax [**s**] indicates that the instruction has a signed form. The syntax [**u**] indicates that the instruction has an unsigned form. The syntax "[**.**]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-20. Implementation-specific Instructions

dccci **dcread** **iccci** **icread**	**macchw**[**s**][**u**] **machhw**[**s**][**u**] **maclhw**[**s**][**u**] **nmacchw**[**s**] **nmachhw**[**s**] **nmaclhw**[**s**]	**mulchw**[**u**] **mulhhw**[**u**] **mullhw**[**u**]	**mfdcr** **mtdcr** **rfci** **tlbre** **tlbsx**[**.**] **tlbwe** **wrtee** **wrteei**

3.11.2 Storage Reference Instructions

Table 3-21 lists the PPC405CR storage reference instructions. Load/store instructions transfer data between memory and the GPRs. These instructions operate on bytes, halfwords, and words. Storage reference instructions also support loading or storing multiple registers, character strings, and byte-reversed data.

In the table, the syntax "[**u**]" indicates that an instruction has an "update" form that updates the RA addressing register with the calculated address, and a "non-update" form. The syntax "[**x**]" indicates that an instruction has an "indexed" form, which forms the address by adding the contents of the RA and RB GPRs and a "base + displacement" form, in which the address is formed by adding a 16-bit signed immediate value (included as part of the instruction word) to the contents of RA GPR.

Table 3-21. Storage Reference Instructions

Loads				**Stores**			
Byte	**Halfword**	**Word**	**Multiple/String**	**Byte**	**Halfword**	**Word**	**Multiple/String**
lbz[**u**][**x**]	**lha**[**u**][**x**] **lhbrx** **lhz**[**u**][**x**]	**lwarx** **lwbrx** **lwz**[**u**][**x**]	**lmw** **lswi** **lswx**	**stb**[**u**][**x**]	**sth**[**u**][**x**] **sthbrx**	**stw**[**u**][**x**] **stwbrx** **stwcx.**	**stmw** **stswi** **stswx**

3.11.3 Arithmetic Instructions

Arithmetic operations are performed on integer operands stored in GPRs. 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 GPRs. The result is placed in a third, operand, which is stored in a GPR. Instructions that perform operations on one operand are defined using a two-operand format; the operation is performed on the operand in a GPR and the result is placed in another GPR. Several instructions also have immediate formats in which an operand is contained in a field in the instruction word.

Most arithmetic instructions have versions that can update CR[CR0] and XER[SO, OV], based on the result of the instruction. Some arithmetic instructions also update XER[CA] implicitly. See “Condition Register (CR)” on page 3-11 and “Fixed Point Exception Register (XER)” on page 3-7 for more information.

Table 3-22 lists the PPC405CR arithmetic instructions. In the table, the syntax “[**o**]” indicates that an instruction has an “o” form that updates XER[SO,OV], and a “non-o” form. The syntax “[**.**]” indicates that the instruction has a “record” form that updates CR[CR0], and a “non-record” form.

Table 3-22. Arithmetic Instructions

Add	Subtract	Multiply	Divide	Negate
add[**o**][**.**] **addc**[**o**][**.**] **adde**[**o**][**.**] **addi** **addic**[.] **addis** **addme**[**o**][**.**] **addze**[**o**][**.**]	**subf**[**o**][**.**] **subfc**[**o**][**.**] **subfe**[**o**][**.**] **subfic** **subfme**[**o**][**.**] **subfze**[**o**][**.**]	**mulhw**[**.**] **mulhwu**[**.**] **mulli** **mullw**[**o**][**.**]	**divw**[**o**][**.**] **divwu**[**o**][**.**]	**neg**[**o**][**.**]

Table 3-23 lists additional arithmetic instructions for multiply-accumulate and multiply halfword operations. In the table, the syntax "**[o]**" indicates that an instruction has an "o" form that updates XER[SO,OV], and a "non-o" form. The syntax "**[.]**" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-23. Multiply-Accumulate and Multiply Halfword Instructions

Multiply-Accumulate	Negative-Multiply-Accumulate	Multiply Halfword
macchw[**o**][**.**] **macchws**[**o**][**.**] **macchwsu**[**o**][**.**] **macchwu**[**o**][**.**] **machhw**[**o**][**.**] **machhws**[**o**][**.**] **machhwsu**[**o**][**.**] **machhwu**[**o**][**.**] **maclhw**[**o**][**.**] **maclhws**[**o**][**.**] **maclhwsu**[**o**][**.**] **maclhwu**[**o**][**.**]	**nmacchw**[**o**][**.**] **nmacchws**[**o**][**.**] **nmachhw**[**o**][**.**] **nmachhws**[**o**][**.**] **nmaclhw**[**o**][**.**] **nmaclhws**[**o**][**.**]	**mulchw**[**.**] **mulchwu**[**.**] **mulhhw**[**.**] **mulhhwu**[**.**] **mullhw**[**.**] **mullhwu**[**.**]

3.11.4 Logical Instructions

Table 3-24 lists the PPC405CR logical instructions. In the table, the syntax "**[.]**" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-24. Logical Instructions

And	And with complement	Nand	Or	Or with complement	Nor	Xor	Equivalence	Extend sign	Count leading zeros
and[**.**] **andi.** **andis.**	**andc**[**.**]	**nand**[**.**]	**or**[**.**] **ori** **oris**	**orc**[**.**]	**nor**[**.**]	**xor**[**.**] **xori** **xoris**	**eqv**[**.**]	**extsb**[**.**] **extsh**[**.**]	**cntlzw**[**.**]

3.11.5 Compare Instructions

These instructions perform arithmetic or logical comparisons between two operands and update the CR with the result of the comparison.

Table 3-25 lists the PPC405CR compare instructions.

Table 3-25. Compare Instructions

Arithmetic	Logical
cmp **cmpi**	**cmpl** **cmpli**

3.11.6 Branch Instructions

These instructions unconditionally or conditionally branch to an 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 CTR as part of branch determination, and can save the return address in the LR.The target address for a branch can be a displacement from the current instruction address (a relative address), an absolute address, or contained in the CTR or LR.

See “Branch Processing” on page 3-30 for more information on branch operations.

Table 3-26 lists the PPC405CR branch instructions. In the table, the syntax “[**l**]” indicates that the instruction has a “link update” form that updates LR with the address of the instruction after the branch, and a “non-link update” form. The syntax “[**a**]” indicates that the instruction has an “absolute address” form, in which the target address is formed directly using the immediate field specified as part of the instruction, and a “relative” form, in which the target address is formed by adding the immediate field to the address of the branch instruction).

Table 3-26. Branch Instructions

Branch
b[**l**][**a**] **bc**[**l**][**a**] **bcctr**[**l**] **bclr**[**l**]

3.11.6.1 CR Logical Instructions

These instructions perform logical operations on a specified pair of bits in the CR, placing the result in another specified bit. These instructions can logically combine the results of several comparisons without incurring the overhead of conditional branch instructions. Software performance can significantly improve if multiple conditions are tested at once as part of a branch decision.

Table 3-27 lists the PPC405CR condition register logical instructions.

Table 3-27. CR Logical Instructions

crand	**crnor**
crandc	**cror**
creqv	**crorc**
crnand	**crxor**
	mcrf

3.11.6.2 Rotate Instructions

These instructions rotate operands stored in the GPRs. Rotate instructions can also mask rotated operands.

Table 3-28 lists the PPC405CR rotate instructions. In the table, the syntax “[**.**]” indicates that the instruction has a “record” form that updates CR[CR0], and a “non-record” form.

Table 3-28. Rotate Instructions

Rotate and Insert	Rotate and Mask
rlwimi[**.**]	**rlwinm**[**.**] **rlwnm**[**.**]

3.11.6.3 Shift Instructions

These instructions rotate operands stored in the GPRs.

Table 3-29 lists the PPC405CR shift instructions. Shift right algebraic instructions implicitly update XER[CA]. In the table, the syntax "**[.]**" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-29. Shift Instructions

Shift Left	Shift Right	Shift Right Algebraic
slw[**.**]	**srw**[**.**]	**sraw**[**.**] **srawi**[**.**]

3.11.6.4 Cache Management Instructions

These instructions control the operation of the ICU and DCU. Instructions are provided to fill or invalidate instruction cache blocks. Instructions are also provided to fill, flush, invalidate, or zero data cache blocks, where a block is defined as a 32-byte cache line.

Table 3-30 lists the PPC405CR cache management instructions.

Table 3-30. Cache Management Instructions

DCU	ICU
dcba **dcbf** **dcbi** **dcbst** **dcbt** **dcbtst** **dcbz** **dccci** **dcread**	**icbi** **icbt** **iccci** **icread**

3.11.7 Interrupt Control Instructions

mfmsr and **mtmsr** read and write data between the MSR and a GPR to enable and disable interrupts. **wrtee** and **wrteei** enable and disable external interrupts. **rfi** and **rfci** return from interrupt handlers. Table 3-31 lists the PPC405CR interrupt control instructions.

Table 3-31. Interrupt Control Instructions

mfmsr
mtmsr
rfi
rfci
wrtee
wrteei

3.11.8 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 will translate a given address, and invalidate all TLB entries. There is also an instruction for synchronizing TLB updates with other processors, but because the PPC405CR is for use in uniprocessor environments, this instruction performs no operation.

Table 3-32 lists the TLB management instructions. In the table, the syntax "[**.**]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table 3-32. TLB Management Instructions

tlbia
tlbre
tlbsx[**.**]
tlbsync
tlbwe

3.11.9 Processor Management Instructions

These instructions move data between the GPRs and SPRs, the CR, and DCRs in the PPC405CR, and provide traps, system calls, and synchronization controls.

Table 3-33 lists the processor management instructions in the PPC405CR.

Table 3-33. Processor Management Instructions

eieio	**mcrxr**	**mtcrf**
isync	**mfcr**	**mtdcr**
sync	**mfdcr**	**mtspr**
	mfspr	**sc**
		tw
		twi

3.11.10 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 word immediate" instruction, because the "rotate left word 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.

Refer to the following chapter and appendixes for lists of the extended mnemonics:

- Chapter 20, “Instruction Set,” lists extended mnemonics under the associated hardware instruction mnemonics.
- Appendix A, “Instruction Summary,” lists extended mnemonics alphabetically, along with the hardware instruction mnemonics.
- Table B-5 in Appendix B, “Instructions by Category,” lists all extended mnemonics.

Chapter 4. Cache Operations

The PPC405CR incorporates two internal caches, a16KB instruction cache and an 8KB data cache. Instructions and data can be accessed in the caches much faster than in main memory.

The instruction cache unit (ICU) controls instruction accesses to main memory and stores frequently used instructions to reduce the overhead of instruction transfers between the instruction pipeline and external memory. Using the instruction cache minimizes access latency for frequently executed instructions.

The data cache unit (DCU) controls data accesses to main memory and stores frequently used data to reduce the overhead of data transfers between the GPRs and external memory. Using the data cache minimizes access latency for frequently used data.

The ICU features:

- Programmable address pipelining and prefetching for cache misses and non-cachable lines
- Support for non-cachable hits from lines contained in fill buffer
- Programmable non-cachable requests to memory as 4 or 8 words (line or half line)
- Bypass path for critical words
- Non-blocking cache for hits during fills
- Flash invalidate (one instruction invalidates entire cache)
- Programmable allocation for fetch fills, enabling program control of cache contents using the **icbt** instruction
- Virtually indexed, physically tagged cache arrays
- Support for 64-bit and 32-bit PLB slaves
- A rich set of cache control instructions

The DCU features:

- Address pipelining for line fills
- Support for load hits from non-cachable and non-allocated lines contained in fill buffer
- Bypass path for critical words
- Non-blocking cache for hits during fills
- Write-back and write-through write strategies controlled by storage attributes
- Programmable non-cachable load requests to memory as lines or words.
- Handling of up to two pending line flushes.
- Holding of up to three stores before stalling the core pipeline
- Physically indexed, physically tagged cache arrays
- Support for 64-bit and 32-bit PLB slaves
- A rich set of cache control instructions

“ICU Organization” on page 4-2 and “DCU Organization” on page 4-5 describe the organization and provide overviews of the ICU and the DCU.

4.1 ICU Organization

The ICU manages instruction transfers between external cachable memory and the instruction queue in the execution unit.

The ICU contains a two-way set-associative 16KB cache memory. Each way is organized in 256 lines of eight words (eight instructions) each.

As shown in Table 4-1, tag ways A and B store instruction address bits $A_{0:21}$ for each line in cache ways A and B. Instruction address bits $A_{19:26}$ serve as the index to the cache array. The two cache lines that correspond to the same line index (one in each way) are called a congruence class.

Table 4-1. Instruction Cache Organization

Tags (Two-way Set)		Instructions (Two-way Set)	
Way A	**Way B**	**Way A**	**Way B**
$A_{0:21}$ Line 0 A	$A_{0:21}$ Line 0 B	Line 0 A	Line 0 B
$A_{0:21}$ Line 1 A	$A_{0:21}$ Line 1 B	Line 1 A	Line 1 B
• • •	• • •	• • •	• • •
$A_{0:21}$ Line 254 A	$A_{0:21}$ Line 254 B	Line 254 A	Line 254 B
$A_{0:21}$ Line 255 A	$A_{0:21}$ Line 255 B	Line 255 A	Line 255 B

When a cache line is to be loaded, the cache way to receive the line is determined by using an least-recently-used (LRU) policy. The index, determined by the instruction address, selects a congruence class. Within a congruence class, the line which was accessed most recently is retained, and the other line is marked as LRU, using an LRU bit in the tag array. The line to receive the incoming data is the LRU line. After the cache line fill, the LRU bit is then set to identify as least-recently-used the line opposite the line just filled.

Figure 4-1 shows the relationships between the ICU and the instruction pipeline.

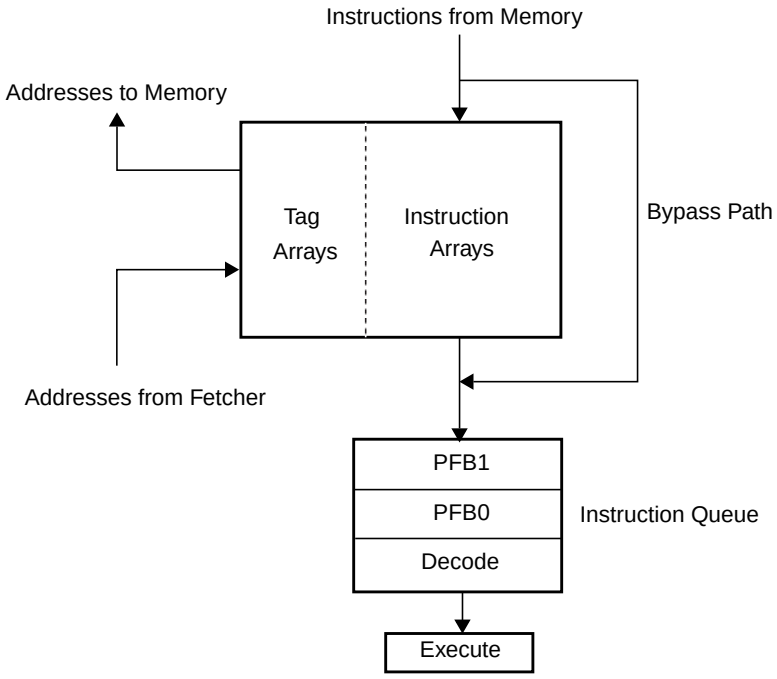


Figure 4-1. Instruction Flow

4.1.1 ICU Operations

Instructions from cachable memory regions are copied into the instruction cache array. The fetcher can access instructions much more quickly from a cache array than from memory. Cache lines are loaded either target-word-first or sequentially. Target-word-first fills start at the requested word, continue to the end of the line, and then wrap to fill the remaining words at the beginning of the line. Sequential fills start at the first word of the cache line and proceed sequentially to the last word of the line.

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 word (the word 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-word-first.

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 from the fill register 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 not written to the cache. During a clock cycle, the ICU can send two instruction to the fetcher.

4.1.2 Instruction Cachability Control

When instruction address translation is enabled (MSR[IR] = 1), instruction cachability 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. Cachability is controlled separately for each page, which can range in size from 1KB to 16MB. “Translation Lookaside Buffer (TLB)” on page 5-2 describes the TLB.

When instruction address translation is disabled (MSR[IR] = 0), instruction cachability is controlled by the Instruction Cache Cachability Register (ICCR). Each field in the ICCR (ICCR[S0:S31]) controls the cachability of a 128MB region (see “Real-mode Storage Attribute Control” on page 5-17). If ICCR[S*n*] = 1, caching is enabled for the specified region; otherwise, caching is inhibited.

The performance of the PPC405CR is significantly lower while fetching instructions from cache-inhibited regions.

Following system reset, address translation is disabled and all ICCR bits are reset to 0 so that no memory regions are cachable. Before regions can be designated as cachable, the ICU cache array must be invalidated. The **iccci** instruction must execute before the cache is enabled. Address translation can then be enabled, if required, and the TLB or the ICCR can then be configured for the required cachability.

4.1.3 Instruction Cache Synonyms

The following information applies only if instruction address translation is enabled (MSR[IR] = 1) and 1KB or 4KB page sizes are used. See Chapter 5, “Memory Management,” for information about address translation and page sizes.

An instruction cache synonym occurs when the instruction cache array contains multiple cache lines from the same real address. Such synonyms result from combinations of:

- Cache array size
- Cache associativity
- Page size
- The use of effective addresses (EAs) to index the cache array

For example, the instruction cache array has a "way size" of 8KB (16KB array/2 ways). Thus, 11 bits ($EA_{19:29}$) are needed to select a word (instruction) in each way. For the minimum page size of 1KB, the low order 8 bits ($EA_{22:29}$) address a word in a page. The high order address bits ($EA_{0:21}$) are translated to form a real address (RA), which the ICU uses to perform the cache tag match. Cache synonyms could occur because the index bits ($EA_{19:29}$) overlap the translated RA bits. For 1KB pages, overlap in $EA_{19:21}$ and $RA_{19:21}$ could result in as many as 8 synomyms. In other words, data from the same RA could occur as as manyas 8 llocations in the cache array. Similarly, for 4KB pages, $EA_{0:19}$ are translated. Differences in EA_{19} and RA_{19} could result in as many as 2 synonyms. For the next largest page size (16KB), only $EA_{0:17}$ are translated. Because there is no overlap with index bits $EA_{19:21}$, synonyms do not occur.

In practice, cache synonyms occur when a real instruction page having multiple virtual mappings exists in multiple cache lines. For 1KB pages, all EAs differing in $EA_{19:21}$ must be cast out of cache, using an **icbi** instruction for each such EA (up to 8 per cache line in the page). For 4KB pages, all EAs differing in EA_{19} must be cast out in the same manner (up to 2 per cache line in the page). For larger pages, cache synonyms do not occur, and casting out any of the multiple EAs removes the physical information from the cache.

Programming Note: To prevent the occurrence of cache synonyms, use only page sizes greater than the cache way size (8KB), if possible. For the PPC405CR, the minimum such page size is 16KB.

4.1.4 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 cachable.

```
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
```

4.2 DCU Organization

The DCU manages data transfers between external cachable memory and the general-purpose registers in the execution unit.

The DCU contains a two-way set-associative 8KB1KB cache memory. Each way is organized in 128 lines of eight words (32 bytes) each.

As shown in Table 4-2, tag ways A and B store data address bits $A_{0:19}$ for each line in cache ways A and B. Data address bits $A_{20:26}$ serve as the index to the cache array. The two cache lines that correspond to the same line index (one in each way) are called a congruence class.

Table 4-2. Data Cache Organization

Tags (Two-way Set)		Data (Two-way Set)	
Way A	**Way B**	**Way A**	**Way B**
$A_{0:19}$ Line 0 A	$A_{0:19}$ Line 0 B	Line 0 A	Line 0 B
$A_{0:19}$ Line 1 A	$A_{0:19}$ Line 1 B	Line 1 A	Line 1 B
• • •	• • •	• • •	• • •
$A_{0:19}$ Line 126 A	$A_{0:19}$ Line 126 B	Line 126 A	Line 126 B
$A_{0:19}$ Line 127 A	$A_{0:19}$ Line 127 B	Line 127 A	Line 127 B

When a cache line is to be loaded, the cache way to receive the line is determined by using an LRU policy. The index, determined by the data address, selects a congruence class. Within a congruence class, the line which was accessed most recently is retained, and the other line is marked as LRU, using an LRU bit in the tag array. The line to receive the incoming data is the LRU line. After the cache line fill, the LRU bit is then set to identify as least-recently-used the line opposite the line just filled.

A bypass path handles data operations in cache-inhibited memory and improves performance during line fill operations.

4.2.1 DCU Operations

Data from cachable memory regions are copied from external memory into lines in the data cache array so that subsequent cache operations result in cache hits. Loads and stores that hit in the DCU are completed in one cycle. For loads, GPRs receive the requested byte, halfword, or word of data from the data cache array. The DCU supports byte-writeability to improve the performance of byte and halfword store operations.

Cache operations require a line fill when they require data from cachable memory regions that are not currently in the DCU. A line fill is the movement of a cache line (eight words) from external memory to the data cache array. Eight words are copied from external memory into the fill buffer, either target-word-first or sequentially. Loading order is controlled by the PLB slave. Target-word-first fills start at the requested word, continue to the end of the line, and then wrap to fill the remaining words at the beginning of the line. Sequential fills start at the first word of the cache line and proceed sequentially to the last word of the line. In both types of fills, the fill buffer, when full, is transferred to the data cache array. The cache line is marked valid when it is filled.

Loads that result in a line fill, and loads from non-cachable memory, are sent to a GPR. The requested byte, halfword, or word is sent from the DCU to the GPR from the fill buffer, using a cache bypass mechanism. Additional loads for data in the fill buffer can be bypassed to the GPR until the data is moved into the data array.

Stores that result in a line fill have their data held in the fill buffer until the line fill completes. Additional stores to the line being filled will also have their data placed in the fill buffer before being transferred into the data cache array.

To complete a line fill, the DCU must access the tag and data arrays. The tag array is read to determine the tag addresses, the LRU line, and whether the LRU line is dirty. A dirty cache line is one that was accessed by a store instruction after the line was established, and can be inconsistent with external memory. If the line being replaced is dirty, the address and the cache line must be saved so that external memory can be updated. During the cache line fill, the LRU bit is set to identify the line opposite the line just filled as LRU.

When a line fill completes and replaces a dirty line, a line flush begins. A flush copies updated data in the data cache array to main storage. Cache flushes are always sequential, starting at the first word of the cache line and proceeding sequentially to the end of the line.

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.

4.2.2 DCU Write Strategies

DCU operations can use write-back or write-through strategies to maintain coherency with external cachable 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. 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.

When data address translation is enabled (MSR[DR] = 1), the W storage attribute in the TLB entry for the memory page controls the write strategy for the page. If TLB_entry[W] = 0, write-back is selected; otherwise, write-through is selected. The write strategy is controlled separately for each page. “Translation Lookaside Buffer (TLB)” on page 5-2 describes the TLB.

When data address translation is disabled (MSR[DR] = 0), the Data Cache Write-through Register (DCWR) sets the storage attribute. Each bit in the DCWR (DCWR[W0:W31]) controls the write strategy of a 128MB storage region (see “Real-mode Storage Attribute Control” on page 5-17). If DCWR[W*n*] = 0, write-back is enabled for the specified region; otherwise, write-through is enabled.

> **Programming Note:** The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

4.2.3 DCU Load and Store Strategies

The DCU can control whether a load receives one word or one line of data from main memory.

For cachable memory, the load without allocate (LWOA) field of the CCR0 controls the type of load resulting from a load miss. If CCR0[LWOA] = 0, a load miss causes a line fill. If CCR0[LWOA] = 1, load misses do not result in a line fill, but in a word load from external memory. For infrequent reads of non-contiguous memory, setting CCR0[LWOA] = 1 may provide a small performance improvement.

For non-cachable memory and for loads misses when CCR0[LWOA] = 1, the load word as line (LWL) field in the CCR0 affects whether load misses are satisfied with a word, or with eight words (the equivalent of a cache line) of data. If CCR0[LWL] = 0, only the target word is bypassed to the core. If CCR0[LWL] = 1, the DCU saves eight words (one of which is the target word) in the fill buffer and bypasses the target data to the core to satisfy the load word request. The fill buffer is not written to the data cache array.

Setting CCR0[LWL] = 1 provides the fastest accesses to sequential non-cachable memory. Subsequent loads from the same line are bypassed to the core from the fill buffer and do not result in additional external memory accesses. The load data remains valid in the fill buffer until one of the following occurs: the beginning of a subsequent load that requires the fill buffer, a store to the target address, a **dcbi** or **dccci** instruction issued to the target address, or the execution of a **sync** instruction. Non-cachable loads to guarded storage never cause a line transfer on the PLB even if CCR0[LWL] = 1. Subsequent loads to the same non-cachable storage are always requested again from the PLB.

For cachable memory, the store without allocate (SWOA) field of the CCR0 controls the type of store resulting from a store miss. If CCR0[SWOA] = 0, a store miss causes a line fill. If CCR0[SWOA] = 1, store misses do not result in a line fill, but in a single word store to external memory.

4.2.4 Data Cachability Control

When data address translation is disabled (MSR[DR] = 0), data cachability is controlled by the Data Cache Cachability Register (DCCR). Each bit in the DCCR (DCCR[S0:S31]) controls the cachability of a 128MB region (see “Real-mode Storage Attribute Control” on page 5-17). If DCCR[S*n*] = 1, caching is enabled for the specified region; otherwise, caching is inhibited.

When data address translation is enabled (MSR[DR] = 1), data cachability 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. Cachability is controlled separately for each page, which can range in size from 1KB to 16MB. “Translation Lookaside Buffer (TLB)” on page 5-2 describes the TLB.

> **Programming Note:** The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

The performance of the PPC405CR is significantly lower while accessing memory in cache-inhibited regions.

Following system reset, address translation is disabled and all DCCR bits are reset to 0 so that no memory regions are cachable. The **dccci** instruction must execute 128before regions can be designated as cachable. This invalidates all congruence classes before enabling the cache. Address translation can then be enabled, if required, and the TLB or the DCCR can then be configured for the desired cachability.

> **Programming Note:** If a data block corresponding to the effective address (EA) exists in the cache, but the EA is non-cachable, 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**.

4.2.5 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.

4.3 Cache Instructions

For detailed descriptions of the instructions described in the following sections, see Chapter 20, “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.

4.3.1 ICU Instructions

The following instructions control instruction cache operations:

icbi Instruction Cache Block Invalidate

Invalidates a cache block.

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 instruction.

iccci Instruction Cache Congruence Class Invalidate

Invalidates the instruction cache array.

This is a privileged instruction.

icread Instruction Cache Read

Reads either an instruction cache tag entry or an instruction word from an instruction cache line, typically for debugging. Fields in CCR0 control instruction behavior (see “Cache Control and Debugging Features” on page 4-11).

This is a privileged instruction.

4.3.2 DCU 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 dcba references a non-cachable address, **dcba** is treated as a no-op.

If dcba references a cachable address, write-through required (which would otherwise cause an alignment 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 cachable.

dcbi Data Cache Block Invalidate

Invalidates a block, if found in the cache, regardless of whether the address is marked cachable. Any modified data is not flushed to memory.

This is a privileged 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 cachable.

dcbt Data Cache Block Touch

Fills a block with data, if the address is cachable and the data is not already in the cache. If the address is non-cachable, this instruction is a no-op.

dcbtst Data Cache Block Touch for Store

Implemented identically to the **dcbt** instruction 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 cachable 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-cachable or write-through, an alignment exception results.

dccci Data Cache Congruence Class Invalidate

Invalidates a congruence class (both cache ways).

This is a privileged instruction.

dcread Data Cache Read

Reads either a data cache tag entry or a data word from a data cache line, typically for debugging. Bits in CCR0 control instruction behavior (see “Cache Control and Debugging Features” on page 4-11).

This is a privileged instruction.

4.4 Cache Control and Debugging Features

Registers and instructions are provided to control cache operation and help debug cache problems. For ICU debug, the **icread** instruction and the Instruction Cache Debug Data Register (ICDBDR) are provided. See “ICU Debugging” on page 4-14 for more information. For DCU debug, the **dcread** instruction is provided. See “DCU Debugging” on page 4-15 for more information.

CCR0 controls the behavior of the **icread** and the **dcread** instructions.

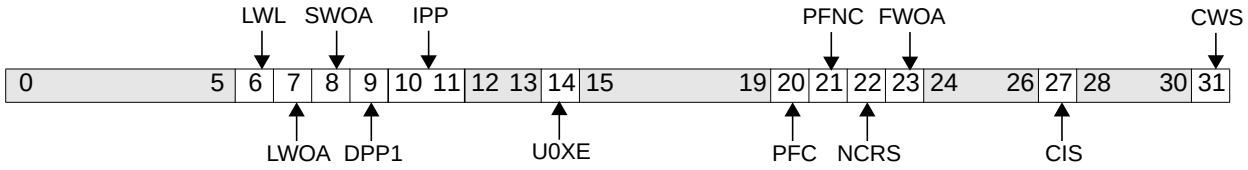


Figure 4-2. Core Configuration Register 0 (CCR0)

0:5		Reserved
6	LWL	Load Word as Line 0 The DCU performs load misses or non-cachable loads as words, halfwords, or bytes, as requested 1 For load misses or non-cachable loads, the DCU moves eight words (including the target word) into the line buffer

7	LWOA	Load Without Allocate 0 Load misses result in line fills 1 Load misses do not result in a line fill, but in non-cachable loads
8	SWOA	Store Without Allocate 0 Store misses result in line fills 1 Store misses do not result in line fills, but in non-cachable stores
9	DPP1	DCU PLB Priority Bit 1 0 DCU PLB priority 0 on bit 1 1 DCU PLB priority 1 on bit 1 **Note:** DCU logic dynamically controls DCU priority bit 0.
10:11	IPP	ICU PLB Priority Bits 0:1 00 Lowest ICU PLB priority 01 Next to lowest ICU PLB priority 10 Next to highest ICU PLB priority 11 Highest ICU PLB priority
12:13		Reserved
14	U0XE	Enable U0 Exception 0 Enables the U0 exception 1 Disables the U0 exception
15:19		Reserved
20	PFC	ICU Prefetching for Cachable Regions 0 Disables prefetching for cachable regions 1 Enables prefetching for cachable regions
21	PFNC	ICU Prefetching for Non-Cachable Regions 0 Disables prefetching for non-cachable regions 1 Enables prefetching for non-cachable regions
22	NCRS	Non-cachable ICU request size 0 Requests are for four-word lines 1 Requests are for eight-word lines
23	FWOA	Fetch Without Allocate 0 An ICU miss results in a line fill. 1 An ICU miss does not cause a line fill, but results in a non-cachable fetch.
24:26		Reserved
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.

4.4.1 CCR0 Programming Guidelines

Several fields in CCR0 affect ICU and DCU operation. Altering these fields while the cache units are involved in PLB transfers can cause errant operation, including a processor hang.

To guarantee correct ICU and DCU operation, specific code sequences must be followed when altering CCR0 fields.

CCR0[IPP, FWOA] affect ICU operation. When these fields are altered, execution of the following code sequence (Sequence 1) is required.

```
    ! SEQUENCE 1 Altering CCR0[IPP, FWOA]
    ! Turn off interrupts
    mfmsr       RM
    addis       RZ,r0,0x0002  ! CE bit
    ori         RZ,RZ,0x8000  ! EE bit
    andc        RZ,RM,RZ      ! Turn off MSR[CE,EE]
    mtmsr       RZ
    ! sync
    sync
    ! Touch code sequence into i-cache
    addis       RX,r0,seq1@h
    ori         RX,RX,seq1@l
    icbt         r0,RX
! Call function to alter CCR0 bits
    b seq1
back:
! Restore MSR to original value
    mtmsr       RM
                •
                •
                •
! The following function must be in cacheable memory
    .align 5      ! Align CCR0 altering code on a cache line boundary.
    seq1:
    icbt        r0,RX             ! Repeat ICBT and execute an ISYNC to guarantee CCR0
    isync                         ! altering code has been completely fetched across the PLB.
    mfspr       RN,CCR0       ! Read CCR0.
    andi/ori RN,RN,0xXXXX    ! Execute and/or function to change any CCR0 bits.
                                  ! Can use two instructions before having to touch
                                  ! in two cache lines.
    mtspr       CCR0, RN ! Update CCR0.
    isync             ! Refetch instructions under new processor context.
    b    back         ! Branch back to initialization code.
```

CCR0[DPP1, U0XE] affect DCU operation. When these fields are altered, execution of the following code sequence (Sequence 2) is required. Note that Sequence 1 includes Sequence 2, so Sequence 1 can be used to alter any CCR0 fields.

In the following sample code, registers RN, RM, RX, and RZ are any available GPRs.

```
! SEQUENCE 2 Alter CCR0[DPP1, U0XE)
! Turn off interrupts
    mfmsr     RM
    addis     RZ,r0,0x0002  ! CE bit
    ori       RZ,RZ,0x8000  ! EE bit
    andc      RZ,RM,RZ      ! Turn off MSR[CE,EE]
    mtmsr     RZ
! sync
    sync
! Alter CCR0 bits
    mfspr     RN,CCR0       ! Read CCR0.
    andi/ori  RN,RN,0xXXXX    ! Execute and/or function to change any CCR0 bits.
    mtspr     CCR0, RN      ! Update CCR0.
    isync           ! Refetch instructions under new processor context.
! Restore MSR to original value
    mtmsr     RM
```

CCR0[CIS, CWS] do not require special programming.

4.4.2 ICU Debugging

The **icread** instruction enables the reading of the instruction cache entries for the congruence class specified by $EA_{19:26}$. The cache information is read into the ICDBDR; from there it can subsequently be moved, using a **mfspr** instruction, into a GPR.

Figure 4-3. Instruction Cache Debug Data Register (ICDBDR)

0:31		Instruction cache information	See **icread**, page 20-68.

ICU tag information is placed into the ICDBDR as shown:

0:21	TAG	Cache Tag
22:26		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 CCR0[CIS] = 0, the data is a word of ICU data from the addressed line, specified by $EA_{27:29}$. If CCR0[CWS] = 0, the data is from the A-way; otherwise; the data from the B-way.

If CCR0[CIS] = 1, the cache information is the cache tag. If CCR0[CWS] = 0, the tag is from the A-way; otherwise, the tag is from the B-way.

Programming Note: The instruction pipeline does not wait for data from an **icread** instruction to arrive before attempting to use the contents the ICDBDR. The following code sequence ensures proper results:

```
icread r5,r6# read cache information
isync       # ensure completion of icread
mficdbdr r7# move information to GPR
```

4.4.3 DCU Debugging

The **dcread** instruction provides a debugging tool for reading the data cache entries for the congruence class specified by $EA_{20:26}$. The cache information is read into a GPR.

If CCR0[CIS] = 0, the data is a word of DCU data from the addressed line, specified by $EA_{27:29}$. If CCR0[CWS] = 0, the data is from the A-way; otherwise; the data is from the B-way.

If CCR0[CIS] = 1, the cache information is the cache tag. If CCR0[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:

0:19	TAG	Cache Tag
20:25		Reserved
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.

4.5 DCU Performance

DCU performance depends upon the application, but, in general, cache hits complete in one cycle 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

4.5.1 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.

In general, the DCU, when hitting in the cache array, can execute a load/store every cycle. If a cache miss occurs, the DCU must retrieve the line from main memory. For cache misses, the DCU stores the cache line in a line buffer until the entire cache line is received. The DCU can accept new DCU commands while the fill progresses. If the instruction causing the line fill is a load, the target word is bypassed to the GPR during the cycle after it becomes available in the fill buffer. When the fill buffer is full, it must be moved into the tag and data arrays. During this time, the DCU cannot begin a new cache operation and stalls the pipeline if new DCU commands are presented. Storing a line in the line buffer takes 3 cycles, unless the line being replaced has been modified. In that case, the operation takes 4 cycles.

The DCU can accept up to two load commands. If the data for the first load command is not immediately available, the DCU can still accept the second load command. If the load data is not required by subsequent instructions, those instructions will continue to execute. If data is required from either load command, the CPU pipeline will stall until the load data has been delivered. The pipeline will also stall until the second load has read the data array if a subsequent data cache command is issued.

In general, if the fill buffer is being used and the next load or store command requires the fill buffer, only one additional command can be accepted before causing additional DCU commands to stall the pipeline.

The DCU can accept up to three outstanding store commands before stalling the CPU pipeline for additional data cache commands.

The DCU can have two flushes pending before stalling the CPU pipeline.

DCU cache operations other than loads and stores stall the CPU pipeline until all prior data cache operations complete. Any subsequent data cache command will stall the pipeline until the prior operation is complete.

4.5.2 Cache Operation Priorities

The DCU uses a priority signal 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 PLB. The priority signal tells the external bus that the DCU requires immediate service, and is valid only when the data cache is requesting access to the PLB. The priority signal is asserted for all loads that require external data, or when the data cache is requesting the PLB and stalling an operation that is being presented to the data cache.

Table 4-3 provides examples of when the priority is asserted and deasserted.

Table 4-3. Priority Changes With Different Data Cache Operations

Instruction Requesting PLB	Priority	Next Instruction	Priority
Any load from external memory	1	N/A	N/A
Any store	0	Any other cache operation not being accepted by the DCU.	1
dcbf	0	Any cache hit.	0
dcbf/dcbst	0	Load non-cache.	1
dcbf/dcbst	0	Another command that requires a line flush.	1
dcbt	0	Any cache hit.	0
dcbi/dccci/dcbz	0	N/A	N/A

4.5.3 Simultaneous Cache Operations

Some cache operations can occur simultaneously to improve DCU performance. For example, combinations of line fills, line flushes, word load/stores, and operations that hit in the cache can occur

simultaneously. Cache operations other than loads/stores cannot begin until the PLB completes all previous operations.

4.5.4 Sequential Cache Operations

Some common cache operations, when performed sequentially, can limit DCU performance: sequential loads/stores to non-cachable 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 cycle. In such cases, the DCU does not limit performance.

However, when a load from a non-cachable storage region is followed by multiple loads from non-cachable regions, the loads can complete no faster than every four cycles, assuming that the addresses are accepted during the same cycle in which it is requested, and that the data is returned during the cycle after the load is accepted.

Similarly, when a store to a non-cachable storage region is followed by multiple stores to non-cachable regions the fastest that the stores can complete is every other cycle. The DCU can have accepted up to three stores before additional DCU commands will stall waiting for the prior stores to complete.

Sequential line fills can limit DCU performance. Line fills occur when a load/store or **dcbt** instruction misses in the cache, and can be pipelined on the PLB interface such that up to two requests can be accepted before stalling subsequent requests. The subsequent operations will wait in the DCU until the first line fill completes. The line fills must complete in the order that they are accepted.

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 flush begins.

Chapter 5. Memory Management

The PPC405CR memory management unit (MMU) performs address translation and protection functions. With appropriate system software, the MMU supports:

- Translation of effective addresses to real addresses
- Independent enabling of instruction and data address 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

5.1 MMU Overview

The instruction and integer units generate 32-bit effective addresses (EAs) for instruction fetches and data accesses, respectively. Instruction EAs are generated for sequential instruction fetches, and for instruction fetches causing changes in program flow (branches and interrupts). Data EAs are generated 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 PPC405CR MMU supports demand-paged virtual memory and other memory management schemes that depend on precise control of effective to real address mapping and flexible memory protection. Translation misses and protection faults cause precise interrupts. Sufficient information is available to correct the fault and restart the faulting instruction.

The MMU divides storage into pages. A page represents the granularity of EA translation and protection controls. Eight page sizes (1KB, 4KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB) are simultaneously supported. A valid entry for a page containing the EA to be translated must be in the translation lookaside buffer (TLB) for address translation to be performed. EAs for which no valid TLB entry exists cause TLB-miss interrupts.

5.2 Address Translation

Fields in the Machine State Register (MSR) control the use of the MMU for address translation. The instruction relocate (IR) field of the MSR controls translation for instruction accesses. The data relocate (DR) field of the MSR controls the translation mechanism for data accesses. These fields, specified independently, can be changed at any time by a program in supervisor state. Note that all interrupts 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, enabling address translation.

The processor references memory when it fetches an instruction, and when it executes load/store, branch, and cache control instructions. Processor accesses to memory use EAs to references a memory location. When translation is enabled, the EA is translated into a real address, as illustrated in Figure 5-1 on page 5-2. 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 supplies the real address for the storage reference. Figure 5-1 illustrates the process.

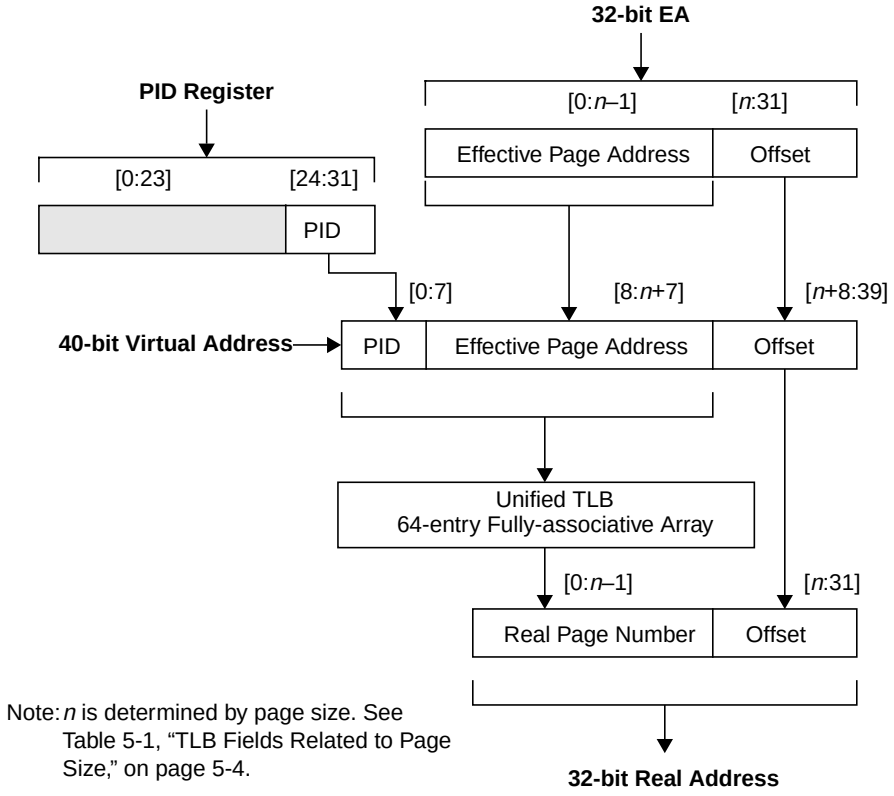


Figure 5-1. Effective to Real Address Translation Flow

5.3 Translation Lookaside Buffer (TLB)

The TLB is hardware that controls translation, protection, and storage attributes. The instruction and data units share a unified fully-associative TLB, in which any page entry (TLB entry) can be placed anywhere in the TLB. TLB entries are maintained under program 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.

5.3.1 Unified TLB

The unified TLB (UTLB) contains 64 entries; each has a TLBHI (tag) portion and a TLBLO (data) portion, as described in Figure 5-2 on page 5-3. 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 $SIZE_{0:2}$. All 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, U0).

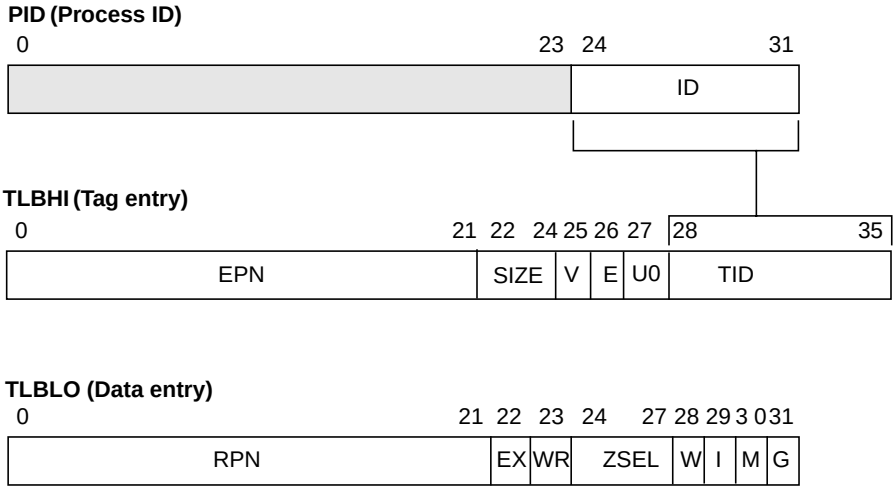


Figure 5-2. TLB Entries

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 used as a process or thread ID. TLBHI[TID] is compared to the PID during a 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.

5.3.2 TLB Fields

Each TLB entry describes a page that is enabled 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

5.3.2.1 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 result of a TLB lookup for such an EA is undefined. The following fields in the TLB entry identify the page. Except as noted, all comparisons must succeed to validate an entry for subsequent use.

EPN (effective page number, 22 bits)

Compared to some number of the $EA_{0:21}$ bits presented to the MMU. The number of bits corresponds to the page size.

The exact comparison depends on the page size, as shown in Table 5-1.

Table 5-1. TLB Fields Related to Page Size

Page Size	SIZE Field	*n* Bits Compared	EPN to EA Comparison	RPN Bits Set to 0
1KB	000	22	$EPN_{0:21} \leftrightarrow EA_{0:21}$	—
4KB	001	20	$EPN_{0:19} \leftrightarrow EA_{0:19}$	$RPN_{20:21}$
16KB	010	18	$EPN_{0:17} \leftrightarrow EA_{0:17}$	$RPN_{18:21}$
64KB	011	16	$EPN_{0:15} \leftrightarrow EA_{0:15}$	$RPN_{16:21}$
256KB	100	14	$EPN_{0:13} \leftrightarrow EA_{0:13}$	$RPN_{14:21}$
1MB	101	12	$EPN_{0:11} \leftrightarrow EA_{0:11}$	$RPN_{12:21}$
4MB	110	10	$EPN_{0:9} \leftrightarrow EA_{0:9}$	$RPN_{10:21}$
16MB	111	8	$EPN_{0:7} \leftrightarrow EA_{0:7}$	$RPN_{8:21}$

SIZE (page size, 3 bits)

Selects one of the eight page sizes, 1KB–16MB, listed in Table 5-1.

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 is compared with the PID value during a TLB access. The TID provides a convenient way to associate a translation with one of 255 unique software entities, typically a process or thread ID maintained by operating system software. Setting TLBHI[TID] = 0x00 disables TID-PID comparison and identifies a TLB entry as valid for all processes; the value of the PID register is then irrelevant.

5.3.2.2 Translation Field

When a TLB entry is identified as matching an EA (and possibly the PID), TLBLO[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 16KB page uses $EA_{0:17}$ for comparison. The translation mechanism replaces $EA_{0:17}$ with $TLBLO[RPN]_{0:17}$ to form the physical address, and $EA_{18:31}$ becomes the real page offset, as illustrated in Figure 5-1.

> **Programming Note:** Software must set all unused bits of RPN (as determined by page size) to 0. See Table 5-1.

5.3.2.3 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 “Zone Protection” on page 5-13.

EX (execute enable, 1 bit)

When set (TLBLO[EX] = 1), enables instruction execution at addresses within a page. ZPR settings can override TLBLO[EX]; see “Zone Protection” on page 5-13, for more information.

WR (write-enable 1 bit)

When set (TLBLO[WR] = 1), enables store operations to addresses in a page. ZPR settings can override TLBLO[WR]; see “Zone Protection” on page 5-13.

5.3.2.4 Storage Attribute Fields

TLB entries contain bits that control 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 attribute is defined in the IBM PowerPC Embedded Environment. The U0 attribute is implementation-specific. .

W (write-through,1 bit)

When set (TLBLO[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-through Register (DCWR) controls the write strategy.

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 undefined.

I (caching inhibited,1 bit)

When set (TLBLO[I] = 1), a memory access is completed by using 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 Cachability Register (ICCR) and Data Cache Cachability Register (DCCR) control cachability. In these registers, the setting of the bit is reversed; 1 indicates that a storage control region is cachable, rather than caching inhibited.

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 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 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 implementations that support multiprocessing, the M storage attribute improves the performance of memory coherency management. Because the PPC405CR 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 (TLBLO[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 data storage address while instruction address translation is enabled results in an instruction storage interrupt because data storage and memory mapped I/O (MMIO) addresses are not used to contain instructions.

An instruction fetch from a guarded region does not occur until the execution pipeline is empty, thus 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 guarding.

U0 (user-defined attribute, 1 bit)

When set (TLBLO[U0] = 1), indicates the user-defined attribute applies to the data in the associated page.

In real mode, the Storage User-defined 0 Register (SU0R) controls the setting of the U0 storage attribute.

E (endian, 1 bit)

When set (TLBLO[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.

5.3.3 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 (see “Shadow TLB Consistency” on page 5-8 for details).

The ITLB can be considered a level-1 instruction-side TLB; the UTLB serves as the level-2 instruction-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.

5.3.3.1 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 address 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 four-cycle latency occurs at each ITLB miss that is also a UTLB hit; the latency is longer 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.

5.3.4 Shadow Data TLB

To enhance performance, eight data-side TLB entries are kept in a eight-entry fully-associative shadow array. This array, called the data TLB (DTLB), helps to avoid TLB contention between instruction accesses to the TLB and load/store operations. Replacement and invalidation of the DTLB entries is managed by hardware. See “Shadow TLB Consistency” on page 5-8 for details.

The DTLB can be considered a level-1 data-side TLB; the UTLB serves as the level-2 data-side TLB. The DTLB is used only by instructions in execute for storing data address translations. Each DTLB entry contains the translation information for a page. The processor uses the DTLB for address translation of data accesses when MSR[DR] = 1.

5.3.4.1 DTLB Accesses

The execute unit accesses the DTLB independently of the rest of the MMU. DTLB accesses are transparent to the executing program, except that DTLB hits contribute to higher overall instruction throughput by allowing instruction address translations to occur in parallel. Therefore, when data accesses hit in the DTLB, the address translation mechanisms in the UTLB are available for use by instruction accesses simultaneously.

The DTLB requests a new entry from the UTLB when a DTLB miss occurs. A three-cycle latency occurs at each DTLB miss that is also a UTLB hit; the latency is longer if it is also a UTLB miss. If there is contention for the UTLB from the instruction side, the data side has priority. A round-robin replacement algorithm replaces existing entries with new entries.

Figure 5-3 illustrates the relationship of the shadow TLBs and UTLB in address translation:

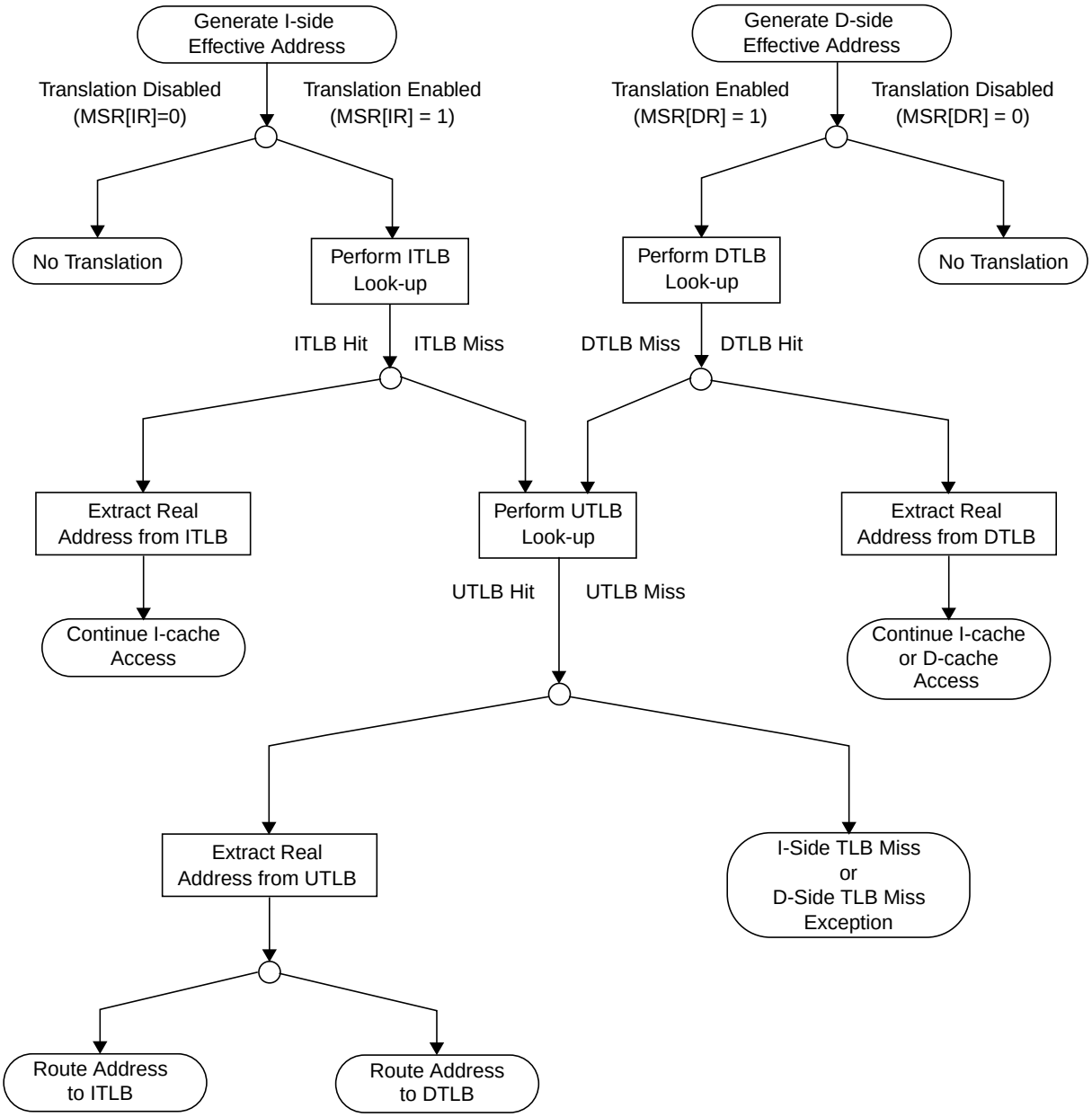


Figure 5-3. ITLB/DTLB/UTLB Address Resolution

5.3.4.2 Shadow TLB 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.

> **Programming Note:** As a rule of thumb, follow software manipulation of an translation mechanism (if performed while translation is active) with a context-synchronizing operation (usually **isync**).

5.4 TLB-Related Interrupts

The processor relies on interrupt handling software to implement paged virtual memory, and to enforce protection of specified memory pages.

When an interrupt occurs, the processor clears MSR[IR, DR]. Therefore, at the start of all interrupt handlers, the processor operates in real mode for instruction accesses and data accesses. Note that when address translation is disabled for an instruction fetch or load/store, the EA is equal to the real address and is passed directly to the memory subsystem (including cache units). Such untranslated addresses bypass all memory protection checks that would otherwise be performed by the MMU.

When translation is enabled, MMU accesses can result in the following interrupts:

- Data storage interrupt
- Instruction storage interrupt
- Data TLB miss interrupt
- Instruction TLB miss interrupt

5.4.1 Data Storage Interrupt

A data storage interrupt is generated when data address translation is active, and the desired access to the EA is not permitted for one of the following reasons:

- In the problem state
  - **icbi**, load/store, **dcbz**, or **dcbf** with an EA whose zone field is set to no access (ZPR[Z*n*] = 00). In this case, **dcbt** and **dcbtst** no-op, rather than cause an interrupt. Privileged instructions cannot cause data storage interrupts.
  - Stores, or **dcbz**, to an EA having TLB_entry[WR] = 0 (write access disabled) and ZPR[Z*n*] $\neq$ 11. (The privileged instructions **dcbi** and **dccci** are treated as “stores”, but cause program interrupts, rather than data storage interrupts.)

- In supervisor state
  - Data store, **dcbi**, **dcbz**, or **dccci** to an EA having TLB_entry[WR] = 0 and ZPR[Z*n*] other than 11 or 10.

dcba does not cause data storage exceptions. If conditions occur that would otherwise cause such an exception, **dcba** is treated as a no-op.

"Zone Protection" on page 5-13 describes zone protection in detail. See "Data Storage Interrupt" on page 9-31 for a detailed discussion of the data storage interrupt.

5.4.2 Instruction Storage Interrupt

An instruction storage interrupt is generated when instruction address 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:

- In the problem state
  - Instruction fetch from an EA with ZPR[Z*n*] = 00.
  - Instruction fetch from an EA having TLB_entry[EX] = 0 and ZPR[Z*n*] ≠ 11.
  - Instruction fetch from an EA having TLB_entry[G] = 1.
- In the supervisor state
  - Instruction fetch from an EA having TLB_entry[EX] = 0 and ZPR[Z*n*] other than 11 or 10.
  - Instruction fetch from an EA having TLB_entry[G] = 1.

See "Zone Protection" on page 5-13 for a detailed discussion of zone protection. See "Instruction Storage Interrupt" on page 9-32 for a detailed discussion of the instruction storage interrupt.

5.4.3 Data TLB Miss Interrupt

A data TLB miss interrupt is generated if data address translation is enabled and a valid TLB entry matching the EA and PID is not present. The interrupt applies to data access instructions and cache operations (excluding cache touch instructions).

See "Data TLB Miss Interrupt" on page 9-38 for a detailed discussion.

5.4.4 Instruction TLB Miss Interrupt

The instruction TLB miss interrupt is generated if instruction address 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 "Instruction TLB Miss Interrupt" on page 9-39 for a detailed discussion.

5.5 TLB Management

The processor does not imply any format for the page tables or the page table entries because there is no hardware support for page table management. Software has complete flexibility in implementing a replacement strategy, because software does the replacing. For example, software can "lock" TLB

entries that correspond to frequently used storage by electing to never replace them, so that those entries are never cast out of the TLB.

TLB management is performed by software with some hardware assist, consisting of:

- Storage of the missed EA in the Save/Restore Register 0 (SRR0) 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 Chapter 20, “Instruction Set,” for detailed instruction descriptions.

5.5.1 TLB Search Instructions (tlbsx/tlbsx.)

tlbsx locates entries in the TLB, to find the TLB entry associated with an interrupt, 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.

5.5.2 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.

5.5.3 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.

5.5.4 TLB Sync Instruction (tlbsync)

tlbsync guarantees that all TLB operations have completed for all processors in a multi-processor system. PPC405CR provides no multiprocessor support, so this instruction performs no function. The instruction is included to facilitate code portability.

5.6 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 out, 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 PPC405CR does not provide hardware reference or change bits, but TLB miss interrupts and data storage interrupts enable system software to maintain reference information for TLB entries and their associated pages, respectively.

A possible algorithm follows. 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 interrupt, 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 data structure, 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 interrupt because write access was turned off. The TLB miss handler records the write to the page in a data structure, for use as a “changed” flag, then sets TLB_entry[WR] = 1 to enable write access. (Note that TLB_entry[WR] can be considered a change bit for the page.) Subsequent write accesses to the page proceed normally.

5.7 Access Protection

The PPC405CR 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.

5.7.1 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 set to one to enable address translation. MSR[IR] enables protection on instruction fetches, which are inherently read-only. MSR[DR] enables protection on data accesses (loads/stores).

5.7.1.1 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]. These fields must match 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 starting a problem state program that is subject to access control.

If TLB_entry[TID] = 0x00, 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. Figure 5-4 illustrates the PID.

Figure 5-4. Process ID (PID)

0:23		Reserved
24:31		Process ID

5.7.1.2 Execute Permissions

If instruction address translation is enabled, instruction fetches are subject to MMU translation and have 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 only 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 interrupt results. This interrupt is precise with respect to the attempted execution. If the fetcher discards the instruction without attempting to execute it, no interrupt will result.

Zone protection can alter execution protection.

5.7.1.3 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 those caused by **dcbz**, **dcbi**, or **dccci**) is made to an EA having TLB_entry[WR] = 0, a data storage interrupt results. This interrupt is precise.

Zone protection can alter write protection (see “Zone Protection” on page 5-13). In addition, only zone protection can prevent read access of a page defined by a TLB entry.

5.7.1.4 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 as 16 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 alter 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.

The fields are defined within the ZPR as follows:

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 the ZPR illustration. 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.

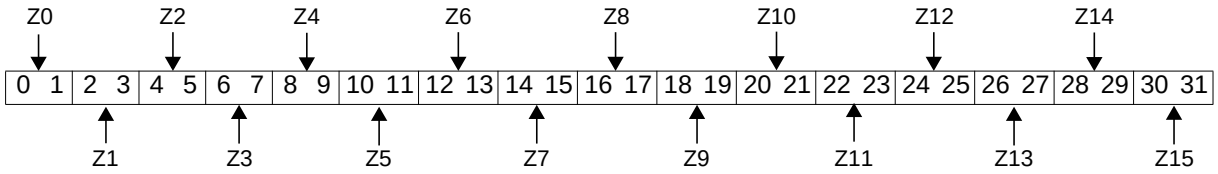


Figure 5-5. Zone Protection Register (ZPR)

0:1	Z0	TLB page access control for all pages in this zone.	
		In the problem state (MSR[PR] = 1): 00 No access 01 Access controlled by applicable TLB_entry[EX, WR] 10 Access controlled by applicable TLB_entry[EX, WR] 11 Accessed as if execute and write permissions (TLB_entry[EX, WR]) are granted.	In the supervisor state (MSR[PR] = 0): 00 Access controlled by applicable TLB_entry[EX, WR] 01 Access controlled by applicable TLB_entry[EX, WR] 10 Access controlled by applicable TLB_entry[EX, WR] 11 Accessed as if execute and write permissions (TLB_entry[EX, WR]) are granted.
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.

Setting ZPR[Z*n*] = 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 interrupt if MSR[DR] = 1 and ZPR[Z*n*] = 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.

5.7.2 Access Protection for Cache Control Instructions

Architecturally the instructions **dcba**, **dcbi**, and **dcbz** are treated as “stores” because they can change data, or cause loss of data by invalidating a dirty line (a modified cache block).

Table 5-2 summarizes the conditions under which the cache control instructions can cause data storage interrupts.

Table 5-2. Protection Applied to Cache Control Instructions

Instruction	Possible Data Storage interrupt	
	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

If data address translation is enabled, and write permission is denied (TLB_entry[WR] = 0), **dcbi** and **dcbz** can cause data storage interrupts. **dcbz** can cause a data storage interrupt when executed in the problem state and all access is denied (ZPR[Z*n*] = 00); **dcbi** cannot cause a data storage interrupt 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, interrupts that would otherwise result when ZPR[Z*n*] = 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” 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). To restrict possible damage from an instruction which can change data and yet avoids the protection mechanism, the **dccci** instruction is privileged.

If data address translation is enabled, **dccci** can cause data storage interrupts when TLB_entry[WR] = 0; the operand is treated as if it were address-specific. **dccci** cannot cause a data storage interrupt when ZPR[Z*n*] = 00, because it is a privileged instruction.

Because **dccci** can cause data storage and TLB -miss interrupts, use of **dccci** is not recommended when MSR[DR] = 1; if **dccci** is used. Note that the specific operand address can cause an interrupt.

Architecturally, **dcbt** and **dcbtst** are treated as “loads” because they do not change data; they cannot cause data storage interrupts when TLB_entry[WR] = 0.

The cache block touch instructions **dcbt** and **dcbtst** are considered “speculative” loads; therefore, if a data storage interrupt would otherwise result from the execution of **dcbt** or **dcbtst** when ZPR[Z*n*] = 00, the instruction is treated as a no-op and the interrupt does not occur. Similarly, TLB miss interrupts 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 interrupts when TLB_entry[WR] = 0. Because neither instruction is privileged, they can cause data storage interrupts when ZPR[Z*n*] = 00 and data address translation is enabled.

dcread is a “load” from a non-specific address, and is privileged. Therefore, it cannot cause data storage interrupts when ZPR[Z*n*] = 00 or TLB_entry[WR] = 0.

icbi and **icbt** are considered “loads” and cannot cause data storage interrupts when TLB_entry[WR] = 0. **icbi** can cause data storage interrupts when ZPR[Z*n*] = 00. Because **icbt** is privileged, it cannot cause data storage interrupts when ZPR[Z*n*] = 00.

The **iccci** instruction cannot change data; an instruction cache line cannot be dirty. The **iccci** instruction is privileged and is considered a load. It does not cause data storage interrupts when ZPR[Z*n*] = 00 or TLB_entry[WR] = 0.

Because **iccci** can cause a TLB miss interrupt, using **iccci** is not recommended when data address translation is enabled; if it is used, note that the specific operand address can cause an interrupt.

icread is considered a “load” from a non-specific address, and is privileged. Therefore, it cannot cause data storage interrupts when ZPR[Z*n*] = 00 or TLB_entry[WR] = 0.

5.7.3 Access Protection for String Instructions

The **stswx** instruction with string length equal to 0(XER[TBC] = 0) is a no-op.

When data address translation is enabled and the Transfer Byte Count (TBC) field of the Fixed Point Exception Register (XER) is 0, neither **lswx** nor **stswx** can cause TLB miss interrupts, or data storage interrupts when ZPR[Z*n*] = 0 or TLB_entry[WR] = 0.

5.8 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, G,U0, and E. Note that the U0 and E attributes are not defined in the PowerPC Architecture. The E attribute is defined in the IBM PowerPC Embedded Environment, and the U0 attribute is implementation-specific. No storage attribute control register is implemented for the M storage attribute because the PPC405CR does not provide multi-processor support or hardware support for data coherency.

These 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 “TLB Fields” on page 5-3).

The storage attribute control registers divide the 4GB real address space into thirty-two 128MB regions. In a storage attribute control register, bit 0 controls the lowest addressed 128MB region, bit 1 the next higher-addressed 128MB region, and so on. $EA_{0:4}$ specify a storage control region.

For detailed information on the function of the storage attributes, see “Storage Attribute Fields” on page 5-5.

5.8.1 Storage Attribute Control Registers

Figure 5-6 shows a generic storage attribute control register. The storage attribute control registers have the same bit numbering and address ranges.

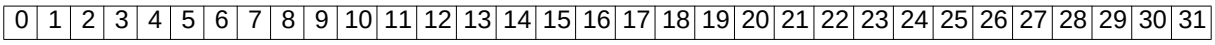


Figure 5-6. Generic Storage Attribute Control Register

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

5.8.1.1 Data Cache Write-through Register (DCWR)

The DCWR controls write-through policy (the W storage attribute) for the data cache unit (DCU). Write-through is not applicable to the instruction cache unit (ICU).

After any reset, all DCWR bits are set to 0, which establishes a write-back write strategy for all regions.

The PowerPC Architecture does not support memory models in which write-through is enabled and caching is inhibited.

5.8.1.2 Data Cache Cachability Register (DCCR)

The 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 in the TLB; DCCR[S*n*] = 1 enables caching, while TLB_entry[I] = 1 inhibits caching.

After any reset, all DCCR bits are set to 0. No memory regions are cachable. Before memory regions can be designated as cachable 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.

5.8.1.3 Instruction Cache Cachability Register (ICCR)

The ICCR controls the I storage attribute for instruction fetches. Note that the polarity of the bits in this register is opposite of that of the I attribute (ICCR[S*n*] = 1 enables caching, while TLB_entry[I] = 1 inhibits caching).

After any reset, all ICCR bits are set to 0. No memory regions are cachable. Before memory regions can be designated as cachable in the ICCR, it is necessary to execute the **iccci** instruction. This invalidates all congruence classes. The ICCR can then be reconfigured, and the ICU can begin normal operation.

5.8.1.4 Storage Guarded Register (SGR)

The SGR controls the G storage attribute for instruction and data accesses.

This attribute does not affect data accesses; the PPC405CR does not perform speculative loads or stores.

After any reset, all SGR bits are set to 1, marking all storage as guarded. For best performance, system software should clear the guarded attribute of appropriate regions as soon as possible. If MSR[IR] = 1, the G attribute comes from the TLB entry. Attempting to execute from a guarded region in translate mode causes an instruction storage interrupt. See “Instruction Storage Interrupt” on page 9-32 for more information.

5.8.1.5 Storage User-defined 0 Register (SU0R)

The Storage User-defined 0 Register (SU0R) controls the user-defined (U0) storage attribute for instruction and data accesses.

After any reset, all SU0R bits are set to 0.

5.8.1.6 Storage Little-Endian Register (SLER)

The SLER controls the E storage attribute for instruction and data accesses.

This attribute determines the byte ordering of storage. “Byte Ordering” on page 3-23 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).

Part III. PPC405CR System Operations

Chapter 6. Clocking

Clocking in the PPC405CR is highly configurable and supports a wide range of clock ratios on the internal and external buses.

illustrates the clocking options for the PPC405CR. A phase-locked loop (PLL) is the source for the CPU clock. Generated clock frequencies are integer ratios of this reference clock. Optional external clock inputs are shown where appropriate. The clocking inputs, which are highlighted in the figure (IICSCL is I/O), are described in more detail in Chapter 22, “Signal Summary.”

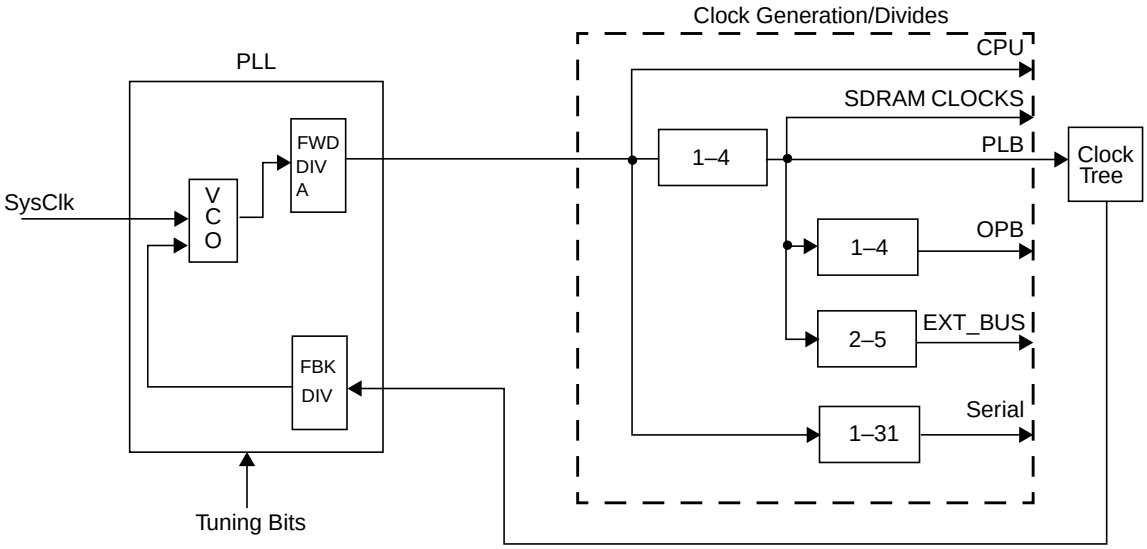


Figure 6-1. PPC405CR Clocking

6.1 PLL Overview

The PLL $V_{DD}A$ input to the PPC405CR requires 2.5 V ± 0.2 V, which should be supplied from a 2.5 V filtered board voltage.

The PLL operating range is controlled by forward divide and feedback divide ratios, tuning bits, and SysClk. The voltage controlled oscillator (VCO) in the PLL must operate at a range of 400–800 MHz. The value of the forward divide and feedback divide ratios affects the CPU frequency and the VCO operating frequency.

Several strapping values are required to correctly configure the PPC405CR. When setting the values, pay close attention to the following information to avoid accidentally configuring the controller in an unusable state.

For any acceptable SysClk input, the VCO frequency is set by the feedback and forward dividers, and the CPU to PLB divider, as shown in the following equation:

VCO = Reference Clock × *M*

where *M* = Feedback Divide × Forward Divide × CPU to PLB Divide

For example, with an input reference frequency of 33.33 MHz, feedback divider of 3, CPU to PLB divider of 2, and forward divider of 3, the VCO frequency at which the PLL stabilizes is 600 MHz.

Strap selection for PLL tuning depends upon the value of *M*, which can range from 6 to 32. The upper limit is based on a minimum input reference clock frequency of 25 MHz and a maximum VCO frequency of 800 MHz. The lower limit is based on a maximum input reference clock of 66 MHz and a minimum VCO frequency of 400MHz. *M* decreases as the reference clock frequency increases.

6.2 Input Reference Clock (SysClk)

The input reference clock, SysClk, must be between 25 MHz and 66 MHz for the PLL to achieve a stable lock. *Input clocks outside this range are not supported.*

Clock generation logic also ensures that the internally generated peripheral clock (PerClk) is phase-aligned with SysClk, as long as both run at the same clock frequency. This is useful when an external device, such as an external master, cannot be synchronized with the PPC405CR PerClk.

6.3 External Clock Strapping Setup

The PPC405CR configures its clocking based on strapping resistors connected to module I/O pins. The state of the strapping pins is stored in the Chip Pin Strapping Register (CPC0_PSR) upon power-on or system reset. Software can use this read-only register to determine clock strapping values. Table 6-1 indicates which CPC0_PSR fields are assigned to clocking functions and how they are encoded. See Chapter 8, “Pin Strapping and Sharing” for a more information and a complete list of strapping values.

Table 6-1. Clock Strapping Values

Strap Option	Description
PLL Forward Divide Ratio CPC0_PSR[PFWD] 00 Bypass mode 01 Divide by 3 10 Divide by 4 11 Divide by 6	These bits indicate bypass mode or one of three valid PLL forward divide ratios. Bypass mode is useful when the PPC405CR is run with a low-frequency reference clock input that is too low for the PLL to lock. For example, if the PPC405CR is clocked at 1 MHz in an emulation system, the PLL cannot be used and should be placed in bypass mode. The other three strapping combinations select one of three valid PLL forward divide ratios: 3, 4, or 6. These ratios are the only acceptable ratios.
PLL Feedback Divide Ratio CPC0_PSR[PFBD] 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4	These bits indicate one of four valid PLL feedback divide ratios: 1, 2, 3, or 4.

Table 6-1. Clock Strapping Values (continued)

Strap Option	Description
PLL Tuning CPC0_PSR[PT] 000 Choice 1; TUNE[5:0] = 010001 001 Choice 2; TUNE[5:0] = 010010 010 Choice 3; TUNE[5:0] = 010011 011 Choice 4; TUNE[5:0] = 010100 100 Choice 5; TUNE[5:0] = 010101 101 Choice 6; TUNE[5:0] = 010110 110 Choice 7; TUNE[5:0] = 010111 111 Choice 8; TUNE[5:0] = 100100	**Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL implemented in the PPC405CR. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs) with the PPC405CR, visit the technical documents area of the IBM PowerPC web site.
PLB Divide Ratio from CPU CPC0_PSR[PDC] 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4	These bits indicate one of four PLB divide ratios from the CPU: 1, 2, 3, or 4.
OPB Divide Ratio from PLB CPC0_PSR[ODP] 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4	These bits indicate one of four OPB divide ratios from the PLB: 1, 2, 3, or 4.
Peripheral Bus Divide Ratio from PLB CPC0_PSR[EBDP] 00 Divide by 2 01 Divide by 3 10 Divide by 4 11 Divide by 5	These bits indicate one of four peripheral bus divide ratios from the PLB: 2, 3, 4, or 5.

Table 6-2 lists PLL tuning settings, which are controlled by the pin strappings reported by CPC0_PSR[PT]. The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL implemented in the PPC405CR. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs) with the PPC405CR, visit the technical documents area of the IBM PowerPC web site.

Table 6-2. PLL Tuning Settings

M Range	Recommended Choice	CPC0_PSR[PT] Value	Equivalent TUNE[5:0]
$6 \leq M \leq 7$	3	010	010011
$7 \leq M \leq 12$	5	100	010101
$12 \leq M \leq 32$	6	101	010110

6.4 Sample Clock Ratios

Table 6-3, Table 6-4, and Table 6-5 list all usable clocking ratio combinations that can be achieved using the strapping inputs for three different reference clock frequencies. VCO frequencies and *M* multipliers are calculated for each valid combination of forward divide (FWD), CPU to PLB divide

(CPU/PLB), and feedback divide (FBK). The resulting clock frequencies for CPU and PLB are also shown. To simplify the table, OPB, external bus, and serial clock frequencies are not shown.

Ratios that result in unusable configurations are not listed. In some cases, the table indicates identical CPU and PLB frequencies. However, in such cases, the VCO frequency varies. When the table indicates a choice of VCO frequencies, select the highest frequency in the range of 400–800 MHz for best PLL performance.

Table 6-3. Possible Clocking Ratios for Reference Clock of 33.3 MHz

Strapping Options				Calculated Values (MHz)		
FWD	**CPU/PLB**	**FBK**	**M**	**VCO**	**CPU**	**PLB**
By-pass[2]	1	x	—	—	refclk	refclk
	2	x	—	—	refclk	refclk/2
	3	x	—	—	refclk	refclk/3
	4	x	—	—	refclk	refclk/4
3	1	4	12	400	133	133
3	2	2	12	400	133	66
3	2	3	18	600	200	100
3	2	4	24	800	266	133
3	3	2	18	600	200	66
3	4	1	12	400	133	33
3	4	2	24	800	266	66
4	1	3	12	400	100	100
4	1	4	16	533	133	133
4	2	2	16	533	133	66
4	2	3	24	800	200	100
4	3	1	12	400	100	33
4	3	2	24	800	200	66
4	4	1	16	533	133	33
6	1	2	12	400	66	66
6	1	3	18	600	100	100
6	1	4	24	800	133	133
6	2	1	12	400	66	33
6	2	2	24	800	133	66
6	3	1	18	600	100	33
6	4	1	24	800	133	33

Note 1: x = don't care.

Note 2: By-pass mode is for hardware emulator use only. Contact your IBM representative for more information.

Note 3: Not all PPC405CR parts support the frequencies shown. Check *PPC405CR Datasheet* for supported CPU and PLB frequencies for a specific part number.

Table 6-4. Possible Clocking Ratios for Reference Clock of 25 MHz

Strapping Options				Calculated Values (MHz)		
FWD	**CPU/PLB**	**FBK**	**M**	**VCO**	**CPU**	**PLB**
By-pass[2]	1	x	—	—	refclk	refclk
	2	x	—	—	refclk	refclk/2
	3	x	—	—	refclk	refclk/3
	4	x	—	—	refclk	refclk/4
3	2	3	18	450	150	75
3	2	4	24	600	200	100
3	3	2	18	450	150	50
3	3	3	27	675	225	75
3	4	2	24	600	200	50
4	1	4	16	400	100	100
4	2	2	16	400	100	50
4	2	3	24	600	150	75
4	2	4	32	800	200	100
4	3	2	24	600	150	50
4	4	1	16	400	100	25
4	4	2	32	800	200	50
6	1	3	18	450	75	75
6	1	4	24	600	100	100
6	2	2	24	600	100	50
6	3	1	18	450	75	25
6	4	1	24	600	100	25

Note 1: x = don't care.

Note 2: By-pass mode is designed for hardware emulator use only. Contact your IBM representative for more information.

Note 3: Not all PPC405CR parts support the frequencies shown. Check *PPC405CR Datasheet* for supported CPU and PLB frequencies for a specific part number.

Table 6-5. Possible Clocking Ratios for Reference Clock of 41.6 MHz

Strapping Options				Calculated Values (MHz)		
FWD	**CPU/PLB**	**FBK**	**M**	**VCO**	**CPU**	**PLB**
By-pass[2]	1	x	—	—	refclk	refclk
	2	x	—	—	refclk	refclk/2
	3	x	—	—	refclk	refclk/3
	4	x	—	—	refclk	refclk/4
3	2	2	12	500	166	83
3	2	3	18	750	250	125
3	3	2	18	750	250	83
3	4	1	12	500	166	41
4	1	3	12	500	125	125
4	2	2	16	666	166	83
4	3	1	12	500	125	41
4	4	1	16	666	166	41
6	1	2	12	500	83	83
6	1	3	18	750	125	125
6	2	1	12	500	83	41
6	3	1	18	750	125	41

Note 1: x = don't care.

Note 2: By-pass mode is designed for hardware emulator use only. Contact your IBM representative for more information.

Note 3: Not all PPC405CR parts support the frequencies shown. Check *PPC405CR Datasheet* for supported CPU and PLB frequencies for a specific part number.

6.5 Serial Port Clocking

The two PPC405CR UARTs (serial ports) can be clocked individually, either from an external serial clock or from an internally generated serial clock. The internally generated clock is derived from the CPU clock, and is CPU/n, where n ranges from 1 to 32.). Subsequently, the serial clock input to the UART is further divided in the UART to generate the desired serial data rate (baud rate).

Refer to Chapter 17, "Serial Port Operations," for more information about choosing CPU clock and serial input clock divisors.

6.6 Clocking Registers

Table 6-6 summarizes the Device Control Registers (DCRs) that control clocking in the PPC405CR.

Table 6-6. Clocking Control Registers

Register	Address	R/W	Description
CPC0_PLLMR	0x0B0	R/O	PLL Mode Register
CPC0_CR0	0x0B1	R/W	Chip Control Register 0

CPC0_CR0 selects the internal serial clock source. The clocking registers are read and written using the **mtdcr** and **mfdcr** instructions.

6.6.1 PLL Mode Register (CPC0_PLLMR)

The read-only CPC0_PLLMR contains a number of PLL and clock divisor values.

CPC0_PLLMR fields are based on the external strapping options.

The CPC0_PLLMR is big endian. However, values returned for the CPC0_PLLMR[FWDV, FBDV, TUN] fields are little endian to enable easier comparison of these bits with the PLL documentation in *ASIC SA-12E Databook* (available from your IBM representative), which uses little endian formats.

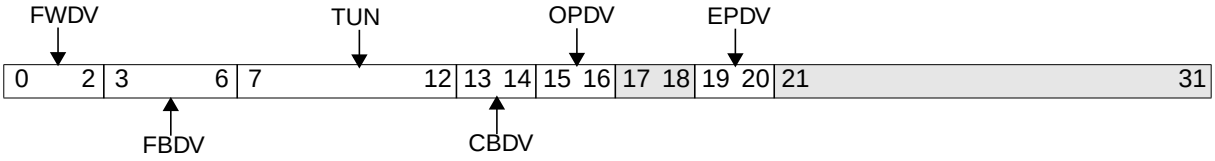


Figure 6-2. PLL Mode Register (CPC0_PLLMR)

0:2	FWDV	Forward Divisor 000 Forward divisor is 8. 001 Forward divisor is 7. 010 Forward divisor is 6. 011 Forward divisor is 5. 100 Forward divisor is 4. 101 Forward divisor is 3. 110 Forward divisor is 2. 111 Forward divisor is 1.	PLLOUTA Value: 50 MHz–100 MHz 58 MHz–114 MHz 66 MHz–134 MHz 80 MHz–160 MHz 100 MHz–200 MHz 133 MHz–267 MHz 200 MHz–400 MHz 400 MHz–800 MHz **Note:** PLLOUTA is the CPU clock.

3:6	FBDV	Feedback Divisor 0000 Feedback divisor is 16. 0001 Feedback divisor is 1. 0010 Feedback divisor is 2. 0011 Feedback divisor is 3. 0100 Feedback divisor is 4. 0101 Feedback divisor is 5. 0110 Feedback divisor is 6. 0111 Feedback divisor is 7. 1000 Feedback divisor is 8. 1001 Feedback divisor is 9. 1010 Feedback divisor is 10. 1011 Feedback divisor is 11. 1100 Feedback divisor is 12. 1101 Feedback divisor is 13. 1110 Feedback divisor is 14. 1111 Feedback divisor is 15.
7:12	TUN	TUNE[5:0] Field **Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs), visit the technical documents area of the IBM PowerPC web site.
13:14	CBDV	CPU:PLB Frequency Divisor 00 CPU:PLB divisor is 1 01 CPU:PLB divisor is 2 10 CPU:PLB divisor is 3 11 CPU:PLB divisor is 4
15:16	OPDV	OPB–PLB Frequency Divisor 00 OPB–PLB divisor is 1 01 OPB–PLB divisor is 2 10 OPB–PLB divisor is 3 11 OPB–PLB divisor is 4
17:18		Reserved
19:20	EPDV	External Bus–PLB Frequency Divisor 00 External bus–PLB divisor ratio is 2:1 01 External bus–PLB divisor ratio is 3:1 10 External bus–PLB divisor ratio is 4:1 11 External bus–PLB divisor ratio is 5:1
21:31		Reserved

6.6.2 Chip Control Register 0 (CPC0_CR0)

Only CPC0_CR0 fields related to clocking are shown in Figure 6-3. CPC0_CR0$_{10:23,}$ which control GPIO and UART functions, are summarized in “CPC0_CR0” on page 21-62.

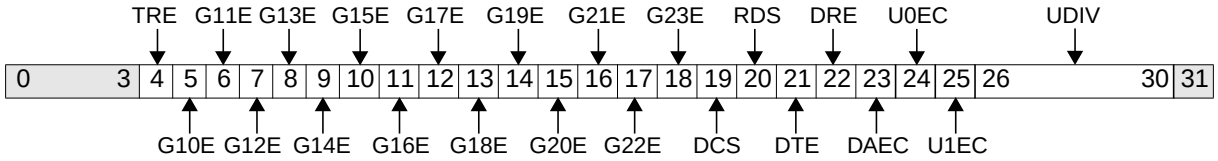


Figure 6-3. Chip Control Register 0 (CPC0_CR0)

0:3		Reserved	
4	TRE	CPU Trace Enable 0 GPIO1-9 are enabled 1 GPIO1-9 are disabled	Trace interface cannot be used when GPIO is enabled.
5	G10E	GPIO 10 Enable 0 Enable $\overline{\text{PerCS1}}$ as a chip select 1 Enable $\overline{\text{PerCS1}}$ as GPIO10	
6	G11E	GPIO 11 Enable 0 Enable $\overline{\text{PerCS2}}$ as a chip select 1 Enable $\overline{\text{PerCS2}}$ as GPIO11	
7	G12E	GPIO 12 Enable 0 Enable $\overline{\text{PerCS3}}$ as a chip select 1 Enable $\overline{\text{PerCS3}}$ as GPIO12	
8	G13E	GPIO 13 Enable 0 Enable $\overline{\text{PerCS4}}$ as a chip select 1 Enable $\overline{\text{PerCS4}}$ as GPIO13	
9	G14E	GPIO 14 Enable 0 Enable $\overline{\text{PerCS5}}$ as a chip select 1 Enable $\overline{\text{PerCS5}}$ as GPIO14	
10	G15E	GPIO 15 Enable 0 Enable $\overline{\text{PerCS6}}$ as a chip select 1 Enable $\overline{\text{PerCS6}}$ as GPIO15	
11	G16E	GPIO 16 Enable 0 Enable $\overline{\text{PerCS7}}$ as a chip select 1 Enable $\overline{\text{PerCS7}}$ as GPIO16	

12	G17E	GPIO 17 Enable 0 Enable interrupt IRQ0 as an interrupt 1 Enable interrupt IRQ0 as GPIO17	The purpose of GPIO_17_EN through GPIO_23_EN is to isolate the interrupt controller from activity on a shared pin when that pin is being used as a GPIO. For instance, when G17E is set to a 1, IRQ0 at the UIC will always be forced to a zero. Note: Setting G17E to a 0 will not prevent GPIO channel 17 (if configured as an output) from creating contention with the off-chip source of the IRQ input. Therefore, be sure to leave the shared GPIO channel disabled when using the pin as an interrupt input.
13	G18E	GPIO 18 Enable 0 Enable interrupt IRQ1 as an interrupt 1 Enable interrupt IRQ1 as GPIO18	
14	G19E	GPIO 19 Enable 0 Enable interrupt IRQ2 as an interrupt 1 Enable interrupt IRQ2 as GPIO19	
15	G20E	GPIO 20 Enable 0 Enable interrupt IRQ3 as an interrupt 1 Enable interrupt IRQ3 as GPIO20	
16	G21E	GPIO 21 Enable 0 Enable interrupt IRQ4 as an interrupt 1 Enable interrupt IRQ4 as GPIO21	
17	G22E	GPIO 22 Enable 0 Enable interrupt IRQ5 as an interrupt 1 Enable interrupt IRQ5 as GPIO22	
18	G23E	GPIO 23 Enable 0 Enable interrupt IRQ6 as an interrupt 1 Enable interrupt IRQ6 as GPIO23	
19	DCS	DSR/CTS select 0 DSR is selected. 1 CTS is selected.	
20	RDS	RTS/DTR select 0 RTS is selected. 1 DTR is selected.	
21	DTE	DMA Transmit Enable for UART0 0 DMA transmit channel is disabled. 1 DMA transmit channel is enabled.	
22	DRE	DMA Receive Enable for UART0 0 DMA receive channel is disabled. 1 DMA receive channel is enabled.	
23	DAEC	DMA Allow Enable Clear for UART0 0 DTE and DRE for UART0 are not cleared when the UART receives a corresponding terminal count. 1 DTE and DRE for UART0 are cleared when the UART receives a corresponding terminal count.	

24	U0EC	Select External Clock for UART0 0 UART0 uses the internally derived serial clock 1 UART0 uses the external serial clock input	
25	U1EC	Select External Clock for UART1 0 UART1 uses the internally derived serial clock 1 UART1 uses the external serial clock input	
26:30	UDIV	UART Internal Clock Divisor 00000 Divide by 1 00001 Divide by 2 00010 Divide by 3 . . . 11110 Divide by 31 11111 Divide by 32	UDIV specifies the divisor of the CPU clock frequency used to derive a UART serial clock frequency. For example, if the CPU is running at 200MHz, a divider value of 20 sets the serial clock frequency to 10MHz. **Note:** The maximum serial clock frequency is less than 1/2 × OPB frequency
31		Reserved	

6.6.3 Chip Control Register 1 (CPC0_CR1)

CPC0_CR1 determines whether CPU timers increment at the CPU clock frquency or at the frequency of the TmrClk input. See Chapter 10, “Timer Facilities,” for more information about the CPU timers.

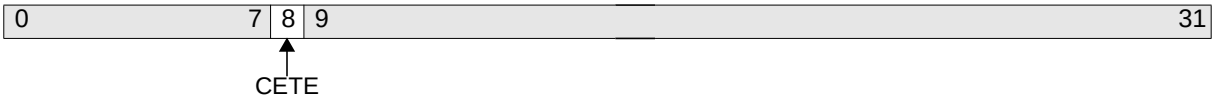


Figure 6-4. Chip Control Register 1 (CPC0_CR1)

0:7		Reserved
8	CETE	CPU External Timer Enable 0 CPU timers increment at CPU clock frequency. 1 CPU timers increment at frequency of TmrClk input.
9:31		Reserved

Chapter 7. Reset and Initialization

This chapter describes reset operations, the initial state of the PPC405CR processor core after a reset, and an example of the initialization code required to begin executing application code. Initialization of external system components or system-specific chip facilities must also be performed, in addition to the basic initialization described in this chapter.

Reset operations affect the PPC405CR at power on time as well as during normal operation, if programmed to do so. To understand how these operations work it is necessary to first understand the signal pins involved as well as the terminology of core, chip and system resets.

7.1 Reset Signals

The PPC405CR provides two reset signals, $\overline{\text{SysReset}}$ and $\overline{\text{ExtReset}}$. $\overline{\text{SysReset}}$ is bidirectional and $\overline{\text{ExtReset}}$ is an output.

When the $\overline{\text{SysReset}}$ signal is an input (asserted by an off-chip device), such as during power on reset (POR), the chip responds by performing a system reset as described in a following section. An external assertion of $\overline{\text{SysReset}}$ is not extended by an assertion of the open drain bidirectional $\overline{\text{SysReset}}$ driver.

As an output, the PPC405CR asserts the $\overline{\text{SysReset}}$ signal when a system reset request is detected. The $\overline{\text{SysReset}}$ open drain driver is activated and the signal is driven low for 8192 SysClk periods. This enables the PPC405CR to reset itself and other devices attached to the same reset network.

The $\overline{\text{ExtReset}}$ signal is used by synchronous peripheral devices served by the PerClk external bus clock, such as ROM and external bus masters. During chip and system resets, $\overline{\text{ExtReset}}$ is asserted until the PerClk signal is stable and all internal resets are released.

7.2 Reset Types

Three types of reset, each with different scope, are possible in the PPC405CR. A core reset affects only the processor core. Chip resets affect the processor core and all on-chip peripherals. System resets affect the processor core, all on-chip peripherals, and any off-chip devices connected to the PPC405CR reset net. Refer to chapters describing the on-chip peripherals for detailed information about their reset behavior.

7.2.1 Core Reset

A core reset results in a reset of the processor core. No other on-chip logic is affected. Clocking logic, outside the processor core, detects the core reset request and asserts the reset input to the processor core for a period of 16 processor core clock cycles.

7.2.2 Chip Reset

A chip reset results in the reset of the processor core and on-chip peripherals. Clocking logic detects the request for a chip reset and asserts the reset input to the processor core for a period of 16 processor core clock cycles. Subsequently, PLL locking is performed as described for a system reset.

During chip reset, the $\overline{\text{ExtReset}}$ signal is driven low to ensure the reset of synchronous devices that use the external bus clock signal, PerClk.

7.2.3 System Reset

A system reset results in a reset of all PPC405CR logic, including the processor core, internal phase-locked loop (PLL), and on-chip peripherals. A system reset can be initiated externally or internally. External system resets are initiated by the assertion of the $\overline{\text{SysReset}}$ signal for at least 16 SysClk cycles. Internal system resets are initiated by the processor core.

When a system reset is requested internally, the bidirectional open drain $\overline{\text{SysReset}}$ signal is asserted to enable other chips to be reset at the same time. In this case, the $\overline{\text{SysReset}}$ signal is driven low for 8192 SysClk cycles, resulting in a System reset of the PPC405CR chip and all other devices attached to the reset network connected to $\overline{\text{SysReset}}$.

After the $\overline{\text{SysReset}}$ signal is deasserted, the PLL begins its locking process, which also requires 8192 SysClk cycles. When 64 SysClk cycles remain, the internal PPC405CR clocks (OPB, EXT, and Serial) begin toggling. The PLB and CPU clocks toggle while $\overline{\text{SysReset}}$ is asserted, and during the PLL locking process. When the PLL lock timer expires, internal resets are released, and the processor core begins its initial instruction fetch.

During system reset, the $\overline{\text{ExtReset}}$ signal is driven low to ensure the reset of synchronous devices that use the external bus clock signal, PerClk.

7.3 Processor Initiated Resets

The processor core in the PPC405CR can request three types of processor resets: core, chip, and system. Each type of reset can be generated by a JTAG debug tool, by the second expiration of the watchdog timer, or by writing a non-zero value to the Reset (RST) field of the Debug Control Register 0 (DBCR0).

The effects of core and chip resets on the processor core are identical. To determine which reset type occurred, the most-recent reset (MRR) field of the Debug Status Register (DBSR) can be examined.

7.4 Processor State After Reset

After a reset, the contents of the Machine State Register (MSR) and the Special Purpose Registers (SPRs) control the initial processor state. The contents of Device Control Registers (DCRs) control the initial states of on-chip devices. Chapter 21, “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. “Contents of Special Purpose Registers after Reset” on page 7-4 describes these initial values. System software fully configures the processor.

“Machine State Register Contents after Reset” on page 7-3 describes the MSR contents.

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 Debug Status Register (DBSR) contains the most recent reset type. The Timer Status Register (TSR) contains the most recent watchdog reset.

7.5 Processor Register Contents After Reset

After a reset, the contents of the SPRs control the initial processor state. The initial register contents vary with the type of reset that occurred.

Chapter 21, “Register Summary,” contains descriptions of the registers referred to in Table 7-1 through Table 7-3.

7.5.1 Machine State Register Contents after Reset

After all resets, all fields of the Machine State Register (MSR) contain zeros. Table 7-1 shows how this affects chip operation.

Table 7-1. MSR Contents after Reset

Register	Field	Core Reset	Chip Reset	System Reset	Comment
MSR	WE	0	0	0	Wait state disabled
	CE	0	0	0	Critical interrupts disabled
	EE	0	0	0	External interrupts disabled
	PR	0	0	0	Supervisor mode
	ME	0	0	0	Machine check exceptions disabled
	DWE	0	0	0	Debug wait mode disabled
	DE	0	0	0	Debug interrupts disabled
	DR	0	0	0	Instruction translation disabled
	IR	0	0	0	Data translation disabled

7.5.2 Contents of Special Purpose Registers after Reset

In general, the contents of Special Purpose Registers (SPRs) are undefined after a core, chip, or system reset. Some SPRs retain the contents they had before a reset occurred.

Table 7-2 shows the contents of SPRs that are defined or unchanged after core, chip, and system resets.

Table 7-2. SPR Contents After Reset

Register	Bits/Fields	Core Reset	Chip Reset	System Reset	Comment
DBCR0	EDM	0	0	0	External debug mode disabled
	RST	00	00	00	No reset action.
DBCR1	0:31	0x00000000	0x00000000	0x00000000	Instruction, data compares disabled
DBSR	MRR	01	10	11	Most recent reset
DCCR	S0:S31	0x00000000	0x00000000	0x00000000	Data cache disabled
ESR	0:31	0x00000000	0x00000000	0x00000000	No exception syndromes
ICCR	S0:S31	0x00000000	0x00000000	0x00000000	Instruction cache disabled
PVR	0:31				Processor version
SGR	G0:G31	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	Storage is guarded
SLER	S0:S31	0x00000000	0x00000000	0x00000000	Storage is big endian
SU0R	K0:K31	0x00000000	0x00000000	0x00000000	Storage is uncompressed

7.6 DCR Contents after Reset

DCR reset values are unaffected by core resets and are generally identical for chip and system resets.

Table 7-3. DCR Contents After Reset

Register	Bits	Reset Value	Comment
Chip Control			
CP0_PSR	0:31		At POR, CPC0_PSR fields are set to the strapping values of the corresponding pins. CPC0_PSR is read-only.
CPC0_CR0	0:31	0x0000003C	
CPC0_CR1	0:31	0x2B0DB800	
CPC0_JTAGID	0:31		Refer to *PPC405CR Embedded Processor Data Sheet* for the value of this read-only register.
CPC0_PLLMR	0:31		At POR, CPC0_PLLMR fields are set to strapping values of the corresponding pins. CPC0_PLLMR is read-only.
Clock and Power Management (CPM)			
CPC0_ER	0:31	0xFFFF8000	$CPC0\_ER_{0:16}$ return 1, $CPC0\_ER_{17:31}$ return 0.

Table 7-3. DCR Contents After Reset

Register	Bits	Reset Value	Comment
CPC0_FR	0:31	0x00000000	
CPC0_SR	0:31	0x00000000	
Decompression Controller			
DCP0_ADDR0	0:31	0x00000000	
DCP0_ADDR1	0:31	0x00000000	
DCP0_MEMBEAR	0:31	0x00000000	
DCP0_CFG	0:31	0x00000001	
DCP0_ESR	0:31	0x00000000	
DCP0_ID	0:31	0x0000504D	
DCP0_ITOR0	0:31	0x00000000	
DCP0_ITOR1	0:31	0x00000000	
DCP0_ITOR2	0:31	0x00000000	
DCP0_ITOR3	0:31	0x00000000	
DCP0_PLBBEAR	0:31	0x00000000	
DCP0_RAM0– DCP0_RAM3FF	1KB	Undefined	
DCP0_VER	0:31	0x00000200	
Direct Memory Access (DMA)			
DMA0_CR0	0:31	0x00000000	
DMA0_CR1	0:31	0x00000000	
DMA0_CR2	0:31	0x00000000	
DMA0_CR3	0:31	0x00000000	
DMA0_CT0	0:31	0x00000000	
DMA0_CT1	0:31	0x00000000	
DMA0_CT2	0:31	0x00000000	
DMA0_CT3	0:31	0x00000000	
DMA0_DA0	0:31	0x00000000	
DMA0_DA1	0:31	0x00000000	
DMA0_DA2	0:31	0x00000000	
DMA0_DA3	0:31	0x00000000	
DMA0_POL	0:31	0x00000000	
DMA0_SA0	0:31	0x00000000	
DMA0_SA1	0:31	0x00000000	
DMA0_SA2	0:31	0x00000000	

Table 7-3. DCR Contents After Reset

Register	Bits	Reset Value	Comment
DMA0_SA3	0:31	0x00000000	
DMA0_SG0	0:31	0x00000000	
DMA0_SG1	0:31	0x00000000	
DMA0_SG2	0:31	0x00000000	
DMA0_SG3	0:31	0x00000000	
DMA0_SGC	0:31	0x00000000	
DMA0_SLP	0:31	0x7C000000	
DMA0_SR	0:31	0x00000000	
External Bus Controller (EBC)			
EBC0_B0AP	0:31	0x7F8FFE80	Slowest possible bus timings.
EBC0_B0CR	0:31	0xFFE28000	2MB read-only bank.
EBC0_B1AP	0:31	0x00000000	
EBC0_B1CR	0:31	0x00000000	
EBC0_B2AP	0:31	0x00000000	
EBC0_B2CR	0:31	0x00000000	
EBC0_B3AP	0:31	0x00000000	
EBC0_B3CR	0:31	0x00000000	
EBC0_B4AP	0:31	0x00000000	
EBC0_B4CR	0:31	0x00000000	
EBC0_B5AP	0:31	0x00000000	
EBC0_B5CR	0:31	0x00000000	
EBC0_B6AP	0:31	0x00000000	
EBC0_B6CR	0:31	0x00000000	
EBC0_B7AP	0:31	0x00000000	
EBC0_B7CR	0:31	0x00000000	
EBC0_BEAR	0:31	0x00000000	
EBC0_BESR0	0:31	0x00000000	
EBC0_BESR1	0:31	0x00000000	
EBC0_CFG	0:31	0x00000000	
Indirect Addressing Registers			
DCP0_CFGADDR		0x00000000	
DCP0_CFGDATA		0x00000000	
EBC0_CFGADDR		0x00000000	

Table 7-3. DCR Contents After Reset

Register	Bits	Reset Value	Comment
EBC0_CFGDATA		0x00000000	
SDRAM0_CFGADDR		0x00000000	
SDRAM0_CFGDATA		0x00000000	
On-Chip Buses			
PLB0_ACR	0:31	0x00000000	
PLB0_BEAR	0:31	Undefined	
POB0_BEAR	0:31	Undefined	
POB0_BESR0	0:31	0x00000000	
POB0_BESR1	0:31	0x00000000	
SDRAM Controller			
SDRAM0_B0CR	0:31	0x00000000	
SDRAM0_B1CR	0:31	0x00000000	
SDRAM0_B2CR	0:31	0x00000000	
SDRAM0_B3CR	0:31	0x00000000	
SDRAM0_BEAR	0:31	0x00000000	
SDRAM0_BESR0	0:31	0x00000000	
SDRAM0_BESR1	0:31	0x00000000	
SDRAM0_CFG	0:31	0x00000000	
SDRAM0_ECCCFG	0:31	0x00000000	
SDRAM0_ECCESR	0:31	0x00000000	
SDRAM0_PMIT	0:31	0x07C00000	
SDRAM0_RTR	0:31	0x05F00000	
SDRAM0_TR	0:31	0x00854009	
Universal Interrupt Controller (UIC)			
UIC0_CR		Undefined	
UIC0_ER		0x00000000	
UIC0_MSR		Undefined	
UIC0_PR		Undefined	
UIC0_SR		Undefined	
UIC0_TR		Undefined	
UIC0_VCR		Undefined	
UIC0_VR		Undefined	

7.7 MMIO Register Contents After Reset

MMIO registers are unaffected by core resets, and are generally identical for chip and system resets.

Table 7-4. MMIO Register Contents After Reset

Register	Bits	Reset Value	Comment
General Purpose I/O (GPIO)			
GPIO0_IR	0:31	Undefined	Read-only; follows the GPIO_In input.
GPIO0_ODR	0:31	0x00000000	
GPIO0_OR	0:31	0x00000000	
GPIO0_TCR	0:31	0x00000000	
Inter-Integrated Circuit (IIC)			
IIC0_CLKDIV	0:7	0x00	
IIC0_CNTL	0:7	0x00	
IIC0_DIRECTCNTL	0:7	0x0F	
IIC0_EXTSTS	0:7	0x00	
IIC0_HMADR	0:7	Undefined	
IIC0_HSADR	0:7	Undefined	
IIC0_INTRMSK	0:7	0x00	
IIC0_LMADR	0:7	Undefined	
IIC0_LSADR	0:7	Undefined	
IIC0_MDBUF	0:7	0x00	
IIC0_MDCNTL	0:7	0x00	
IIC0_SDBUF	0:7	0x00	
IIC0_STS	0:7	0x00	
IIC0_XFRCNT	0:7	0x00	
IIC0_XTCNTLSS	0:7	0x00	
OPB Arbiter			
OPBA0_CR	0:31	0x00000000	
OPBA0_PR	0:31	0x00011011	
Serial Port (UART0)			
UART0_DLL	0:7	0x00000000	
UART0_DLM	0:7	0x00000000	
UART0_FCR	0:7	0x00000000	
UART0_IER	0:7	0x00000000	
UART0_IIR	0:7	0x00000001	
UART0_LCR	0:7	0x00000000	

Table 7-4. MMIO Register Contents After Reset

Register	Bits	Reset Value	Comment
UART0_LSR	0:7	0x01100000	
UART0_MCR	0:7	0x00000000	
UART0_MSR	0:7	0xxxxx0000	$UART0\_MSR_{0:3}$ are driven by $\overline{UART0\_DCD}$, $\overline{UART0\_RI}$, $\overline{UART1\_DSR}$[UART1_CTS], and $\overline{UART1\_RTS}$[UART1_DTR].
UART0_RBR	0:7	0x00000000	
UART0_SCR	0:7	Undefined	
UART0_THR	0:7	0x00000000	
Serial Port (UART1)			
UART1_DLL	0:7	0x00000000	
UART1_DLM	0:7	0x00000000	
UART1_FCR	0:7	0x00000000	
UART1_IER	0:7	0x00000000	
UART1_IIR	0:7	0x00000001	
UART1_LCR	0:7	0x00000000	
UART1_LSR	0:7	0x01100000	
UART1_MCR	0:7	0x00000000	
UART1_MSR	0:7	Undefined	
UART1_RBR	0:7	Undefined	
UART1_SCR	0:7	Undefined	
UART1_THR	0:7	0x00000000	

7.8 PPC405CR Chip Initialization

The universal Interrupt controller (UIC) requires initialization for proper operation. The UART controller may require initialization, depending upon the application.

Other peripheral devices can be initialized as appropriate for the system design.

7.8.1 UIC Initialization

The following information does not provide all initialization information for the UIC. Some initialization details are application-dependent.

The polarity and sensitivity of the on-chip interrupts must be initialized for proper chip operation. The fields controlling on-chip interrupts in the UIC Polarity Register ($UIC0\_PR_{0:3, 5:8, 17}$) must be set to 1. See “UIC Polarity Register (UIC0_PR))” on page 9-8 for details. The fields controlling on-chip interrupts in the UIC Trigger Register ($UIC0\_TR_{0:2, 5:8, 17}$) must be set to 0. The field controlling the

external master interrupt trigger (UIC0_TR$_3$) must be set to 1. See “UIC Trigger Register (UIC0_TR)” on page 9-10 for details.

7.8.2 UART Initialization

Bits 19 and 20 of Chip Control Register 0 (CPC0_CR0) control which of two modem control signal pairs, Data Set Ready (DSR) and Data Terminal Ready (DTR) or Clear to Send (CTS) and Ready to Send (RSR), are used. The signal pairs are implemented using the multiplexed pins $\overline{\text{UART1\_DSR}}$[$\overline{\text{UART1\_CTS}}$], controlled by CPC0_CR0$_{19}$, and $\overline{\text{UART1\_RTS}}$[$\overline{\text{UART1\_DTR}}$], controlled by CPC0_CR0$_{20}$. After reset, CPC0_CR0$_{19}$ or CPC0_CR0$_{20}$ must be changed to achieve a required typical pairing.

7.9 PPC405CR Initial Processor Sequencing

After any reset, the processor core fetches the word at address 0xFFFFFFFC and attempts to execute it. Because the only memory configured immediately after reset is the upper 2MB region (0xFFE00000–0xFFFFFFFF), the instruction at 0xFFFFFFFC must be a branch instruction.

Because the processor is initially in big endian mode, initialization code must be in big endian format until the endian storage attribute for the addressed region is changed, or until code branches to a region defined as little endian storage.

Before a reset operation begins, the system must provide non-volatile memory, or memory initialized by some mechanism external to the processor. This memory must be located at address 0xFFFFFFFC in the upper 2MB region. This memory can be attached to the external bus controller (EBC). The upper 2MB bank configuration after reset is 256 wait states, three cycles of address to chip select delay, three cycles of chip select to output enable delay, and seven cycles of hold time. The bus width (8-, 16-, or 32-bit) is controlled by the ROM width strapping signals. See “Pin Strapping” on page 8-1 for details.

7.10 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 PPC405CR 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 manual.

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 will improve performance for unguarded regions.

1. Configure the following storage attribute control registers, if necessary:
   - Initialize the SLER to configure storage byte ordering.
   - Initialize the SU0R to configure storage compression.

2. Before executing instructions as cachable:
   – Invalidate the instruction cache.
   – Initialize the ICCR to configure instruction cachability.
3. Before using storage access instructions:
   – Invalidate the data cache.
   – Initialize CRRO to determine if a store miss results in a line fill (SWOA).
   – Initialize the DCWR to select copy-back or write-through caching.
   – Initialize the DCCR to configure data cachability.
4. Before allowing interrupts (synchronous or asynchronous):
   – Initialize the EVPR to point to vector table.
   – Provide vector table with branches to interrupt handlers.
5. Before enabling asynchronous interrupts:
   – Initialize timer facilities.
   – Initialize MSR to enable appropriate interrupts.
6. Initialize other processor features, such as the MMU, debug, and trace.
7. Initialize non-processor resources.
   – Initialize system memory as required by the operating system or application code.
   – Initialize off-chip system facilities.
8. Start the execution of operating system or application code.

7.11 Initialization Code Example

The following initialization code illustrates the steps that should be taken to initialize the processor before an operating system or user programs begin execution. The example is presented in pseudo-code; function calls are named similarly to PPC405CR mnemonics where appropriate.

```
[/* ———————————————————————————————————————— */
/*      PPC405CR Initialization Pseudo Code                                 */
/*———————————————————————————————————————————————————————————————————— */
@0xFFFFFFFC:                    /* initial instruction fetch from 0xFFFFFFFC   */
   ba(init_code);               /* branch to initialization code               */

@init_code:

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

      mtspr(SGR, guarded_attribute);

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

      mtspr(SLER, endianness);
```

```
    mtspr(SU0R, compression_attribute);

/*-------------------------------------------------------------*/
/* Invalidate the instruction cache and enable cachability     */
/*-------------------------------------------------------------*/

iccci;                              /* invalidate i-cache */
    mtspr(ICCR, i_cache_cachability);                   /* enable I-cache*/
isync;

/*-------------------------------------------------------------------------*/
/* Invalidate the data cache and enable cachability                        */
/*-------------------------------------------------------------------------*/

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 += 32;                  /* point to the next congruence class  */
}
mtspr(CCR0, store-miss_line-fill);
mtspr(DCWR, copy-back_write-thru);
mtspr(DCCR, d_cache_cachability);                   /* 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 */

/*-------------------------------------------------------------------------*/
/* 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(TBL, 0);                      /* reset time base low first to avoid ripple   */
mtspr(TBU, time_base_u);            /* set time base, hi first to catch possible ripple */
mtspr(TBL, 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.  */
/*  ------------------------------------------------------------------------ */
```

Chapter 8. Pin Strapping and Sharing

8.1 Pin Strapping

When power is applied to the PPC405CR, a start-up process is initiated in which internal functions are initialized. Some of these functions have optional choices. Which of the options are used for initialization is determined by the way a specific set of I/O pins (balls) are conditioned. The conditioning is achieved using external pull-up (indicated as 1) or pull-down (indicated as 0) resistors connected to the pins.

While the $\overline{\text{SysReset}}$ input signal is low (system reset), the state of the I/O pins is read to enable default initial conditions before PPC405CR start-up. The actual capture instant is the nearest SysClk clock edge before the deassertion of $\overline{\text{SysReset}}$. The state of the pins as read is stored in the Chip Pin Strapping Register (CPC0_PSR) shown in Figure 8-1. Refer to *PowerPC 405CR Embedded Processor Data Sheet*, which describes the strapping pins.

8.1.1 Chip Pin Strapping Register (CPC0_PSR)

CPC0_PSR contains the state of the strapping pins as read during system reset.

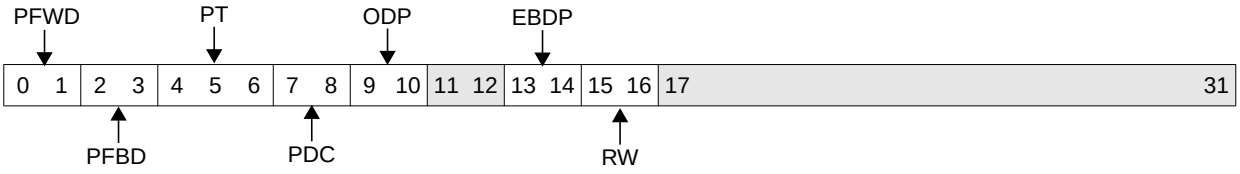


Figure 8-1. Chip Pin Strapping Register (CPC0_PSR)

0:1	PFWD	PLL Forward Divider 00 Bypass Mode 01 Divide by 3 10 Divide by 4 11 Divide by 6	
2:3	PFBD	PLL Feedback Divider 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4	
4:6	PT	PLL Tuning 000 Choice 1; TUNE[5:0] = 010001 001 Choice 2; TUNE[5:0] = 010010 010 Choice 3; TUNE[5:0] = 010011 011 Choice 4; TUNE[5:0] = 010100 100 Choice 5; TUNE[5:0] = 010101 101 Choice 6; TUNE[5:0] = 010110 110 Choice 7; TUNE[5:0] = 010111 111 Choice 8; TUNE[5:0] = 100100	**Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs), visit the technical documents area of the IBM PowerPC web site.

7:8	PDC	PLB Divider from CPU 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4
9:10	ODP	OPB Divider from PLB 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4
11:12		Reserved
13:14	EBDP	External Bus Divider from PLB 00 Divide By 2 01 Divide By 3 10 Divide By 4 11 Divide By 5
15:16	RW	ROM Width 00 8-bit ROM 01 16-bit ROM 10 32-bit ROM 11 Reserved
17:31		Reserved

8.2 Pin Sharing

The PPC405CR uses pin (ball) multiplexing (sharing) to reduce the total pin requirement without significantly reducing functionality. Some of the pins that are multiplexed require DCR register programming to configure the pin for the desired function. It is expected that in an application, a particular pin is programmed to serve one function. While nothing prevents changing the function of a pin during operation, most applications configure a pin once at power-on reset (POR). Table 8-1 lists

the multiplexed PPC405CR signals and indicates the DCR bit that controls the pin. The default signal appears first and the alternate signal is in brackets.

Table 8-1. Multiplexed Pins

Signal	DCR Bit	Description
GPIO1[TS1E]	CPC0_CR0[TRE]	Set of GPIO pins that can be reconfigured for use as the CPU Trace interface.
GPIO2[TS2E]		
GPIO3[TS1O]		
GPIO4[TS2O]		
GPIO5[TS3]		
GPIO6[TS4]		
GPIO7[TS5]		
GPIO8[TS6]		
GPIO9[TrcClk]		
$\overline{\text{PerCS1}}$[GPIO10]	CPC0_CR0[G10E]	Peripheral Chip selects that can be reconfigured for use as GPIOs.
$\overline{\text{PerCS2}}$[GPIO11]	CPC0_CR0[G11E]	
$\overline{\text{PerCS3}}$[GPIO12]	CPC0_CR0[G12E]	
$\overline{\text{PerCS4}}$[GPIO13]	CPC0_CR0[G13E]	
$\overline{\text{PerCS5}}$[GPIO14]	CPC0_CR0[G14E]	
$\overline{\text{PerCS6}}$[GPIO15]	CPC0_CR0[G15E]	
$\overline{\text{PerCS7}}$[GPIO16]	CPC0_CR0[G16E]	
IRQ0[GPIO17]	CPC0_CR0[G17E]	External Interrupts that can be reconfigured for use as GPIOs.
IRQ1[GPIO18]	CPC0_CR0[G18E]	
IRQ2[GPIO19]	CPC0_CR0[G19E]	
IRQ3[GPIO20]	CPC0_CR0[G20E]	
IRQ4[GPIO21]	CPC0_CR0[G21E]	
IRQ5[GPIO22]	CPC0_CR0[G22E]	
IRQ6[GPIO23]	CPC0_CR0[G23E]	
$\overline{\text{UART1\_DSR}}$[$\overline{\text{UART1\_CTS}}$][1]	CPC0_CR0[DCS]	UART1 DSR pin that can be reconfigured for use as UART1 CTS.
$\overline{\text{UART1\_RTS}}$[$\overline{\text{UART1\_DTR}}$][1]	CPC0_CR0[RDS]	UART1 RTS pin that can be reconfigured for use as UART1 DTR.
Note: Typically DSR and DTR are paired and CTS and RTS are paired. With the current multiplex defaults one of the pins must be changed during initialization to achieve the typical pairings.		

Chapter 9. Interrupt Controller Operations

The PPC405CR contains a universal interrupt controller (UIC) that provides all necessary control, status, and communication between the various internal and external interrupt sources and the processor core.

9.1 UIC Overview

The UIC supports 9 internal interrupts and 7 external interrupts. Status reporting (using the UIC Status Register [UIC0_SR]) is provided to ensure that systems software can determine the current and interrupting state of the system and respond appropriately. Software can generate interrupts to simplify software development and for diagnostics.

The interrupts can be programmed, using the UIC Critical Register (UIC0_CR), to generate either a critical or a non-critical interrupt signal to the processor core.

The privileged **mtdcr** and **mfdcr** instructions, which are used by system software, are used to read and write the UIC registers.

An optional critical interrupt vector generator can reduce interrupt handling latency for critical interrupts. Vector calculation is described in detail in “UIC Vector Register (UIC0_VR)” on page 9-15.

9.2 UIC Features

- Support for 9 internal and 7 external interrupts
- Support for asychronous level- or edge-sensitive interrupt types
- Programmable polarity for all interrupt types
- Programmable critical/non-critical interrupt selection for each interrupt bit
- Prioritized critical interrupt vector generation
- A UIC Status Register (UIC0_SR) providing the following information:
  - Current state of interrupts
  - Current state of all enabled interrupts (those masked using the UIC Enable Register (UIC0_ER))

9.3 UIC Interrupt Assignments

The UIC supports various internal and external interrupt sources as shown in Table 9-1.

Table 9-1. UIC Interrupt Assignments

Interrupt	Polarity	Sensitivity	Interrupt Source
0	High	Level	UART0
1	High	Level	UART1
2	High	Level	IIC

Table 9-1. UIC Interrupt Assignments (continued)

Interrupt	Polarity	Sensitivity	Interrupt Source
3	High	Edge	External Master
4	Reserved		
5	High	Level	DMA Channel 0
6	High	Level	DMA Channel 1
7	High	Level	DMA Channel 2
8	High	Level	DMA Channel 3
17	High	Level	ECC Correctable Error
18	Reserved		
19			
20			
21			
22			
23			
24			
25	Programmable	Programmable	External IRQ 0
26	Programmable	Programmable	External IRQ 1
27	Programmable	Programmable	External IRQ 2
28	Programmable	Programmable	External IRQ 3
29	Programmable	Programmable	External IRQ 4
30	Programmable	Programmable	External IRQ 5
31	Programmable	Programmable	External IRQ 6

9.4 Interrupt Programmability

The on-chip interrupts (interrupts 0–3, 5–8, and 17) and the external IRQs (interrupts 25–31) are programmable. However, the polarity and sensitivity of the on-chip interrupts must be programmed as shown in Table 9-1, “UIC Interrupt Assignments,” on page 9-1, using the UIC Polarity Register (UIC0_PR) and the UIC Trigger Register (UIC0_TR), respectively.

9.5 UIC Registers

The UICs include the Device Control Registers (DCRs) listed in Table 9-2.

The registers are accessed using the **mfdcr** and **mtdcr** instructions.

Table 9-2. UIC DCRs

Mnemonic	Register	Address	Access	Page
UIC0_SR	UIC Status Register	0x0C0	Read/Clear	9-3
UIC0_ER	UIC Enable Register	0x0C2	R/W	9-5
UIC0_CR	UIC Critical Register	0x0C3	R/W	9-7
UIC0_PR	UIC Polarity Register	0x0C4	R/W	9-8
UIC0_TR	UIC Trigger Register	0x0C5	R/W	9-10
UIC0_MSR	UIC Masked Status Register	0x0C6	Read-only	9-12
UIC0_VR	UIC Vector Register	0x0C7	Write-only	9-15
UIC0_VCR	UIC Vector Configuration Register	0x0C8	Read-only	9-14

9.5.1 UIC Status Register (UIC0_SR)

To report interrupt status, the UIC0_SR fields capture and hold internal and external interrupts until the fields are intentionally cleared. To clear a field, write 1 to the field.

The values of other UIC registers do not affect the UIC0_SR fields.

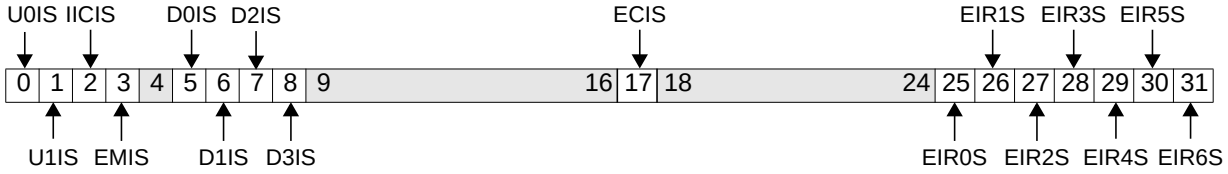


Figure 9-1. UIC Status Register (UIC0_SR)

0	U0IS	UART0 Interrupt Status 0 A UART0 interrupt has not occurred. 1 A UART0 interrupt occurred.
1	U1IS	UART1 Interrupt Status 0 A UART1 interrupt has not occurred. 1 A UART1 interrupt occurred.

2	IICIS	IIC Interrupt Status 0 An IIC interrupt has not occurred. 1 An IIC interrupt occurred.
3	EMIS	External Master Interrupt Status 0 An external master interrupt has not occurred. 1 An external master interrupt occurred.
4		Reserved
5	D0IS	DMA Channel 0 Interrupt Status 0 A DMA channel 0 interrupt has not occurred. 1 A DMA channel 0 interrupt occurred.
6	D1IS	DMA Channel 1 Interrupt Status 0 A DMA channel 1 interrupt has not occurred. 1 A DMA channel 1 interrupt occurred.
7	D2IS	DMA Channel 2 Interrupt Status 0 A DMA channel 2 interrupt has not occurred. 1 A DMA channel 2 interrupt occurred.
8	D3IS	DMA Channel 3 Interrupt Status 0 A DMA channel 3 interrupt has not occurred. 1 A DMA channel 3 interrupt occurred.
9:16		Reserved
17	ECIS	ECC Correctable Error Interrupt Status 0 An ECC correctable error interrupt did not occur. 1 An ECC correctable error interrupt occurred.
18:24		Reserved
25	EIR0S	External IRQ 0 Status 0 An external IRQ 0 interrupt has not occurred. 1 An external IRQ 0 interrupt occurred.
26	EIR1S	External IRQ 1 Status 0 An external IRQ 1 interrupt has not occurred. 1 An external IRQ 1 interrupt occurred.
27	EIR2S	External IRQ 2 Status 0 An external IRQ 2 interrupt has not occurred. 1 An external IRQ 2 interrupt occurred.
28	EIR3S	External IRQ 3 Status 0 An external IRQ 3 interrupt has not occurred. 1 An external IRQ 3 interrupt occurred.

29	EIR4S	External IRQ 4 Status 0 An external IRQ 4 interrupt has not occurred. 1 An external IRQ 4 interrupt occurred.
30	EIR5S	External IRQ 5 Status 0 An external IRQ 5 interrupt has not occurred. 1 An external IRQ 5 interrupt occurred.
31	EIR6S	External IRQ 6 Status 0 An external IRQ 6 interrupt has not occurred. 1 An external IRQ 6 interrupt occurred.

9.5.2 UIC Enable Register (UIC0_ER)

Fields in UIC0_ER, which correspond to fields in UIC0_SR, enable or disable the reporting of the corresponding fields of the UIC0_SR.

If a UIC0_ER field is set to 1, the corresponding field of the UIC0_SR generates a critical or non-critical interrupt signal to the processor core, if the UIC0_SR field is set to 1. If a UIC0_ER field is set to 0, the corresponding field of the UIC0_SR field does not generate a critical or non-critical interrupt signal to the processor core, regardless of the setting of the UIC0_SR field. The critical and non-critical interrupt signals in the processor core are controlled by fields in the Machine State Register (MSR).

The class of generated signals (critical or non-critical) is controlled by UIC0_CR.

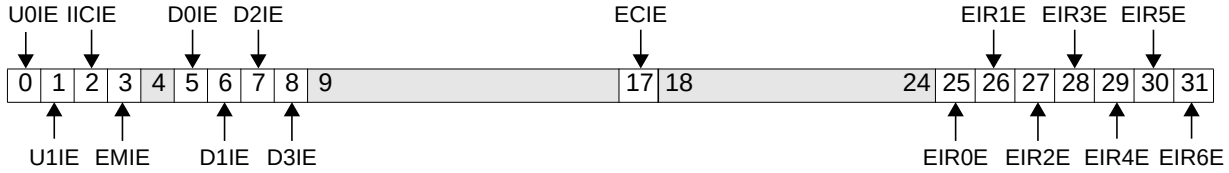


Figure 9-2. UIC Enable Register (UIC0_ER)

0	U0IE	UART0 Interrupt Enable 0 UART0 interrupt is disabled. 1 UART0 interrupt is enabled.
1	U1IE	UART1 Interrupt Enable 0 UART1 interrupt is disabled. 1 UART1 interrupt is enabled.
2	IICIE	IIC Interrupt Enable 0 IIC interrupt is disabled. 1 IIC interrupt is enabled.
3	EMIE	External Master Interrupt Enable 100 External master interrupt is disabled. 1 0xxExternal master interrupt is enabled.

4		Reserved
5	D0IE	DMA Channel 0 Interrupt Enable 0 DMA channel 0 interrupt is disabled. 1 DMA channel 0 interrupt is enabled.
6	D1IE	DMA Channel 1 Interrupt Enable 0 DMA channel 1 interrupt is disabled. 1 DMA channel 1 interrupt is enabled.
7	D2IE	DMA Channel 2 Interrupt Enable 0 DMA channel 2 interrupt is disabled. 1 DMA channel 2 interrupt is enabled.
8	D3IE	DMA Channel 3 Interrupt Enable 0 DMA channel 3 interrupt is disabled. 1 DMA channel 3 interrupt is enabled.
9:16		Reserved
17	ECIE	ECC Correctable Error Interrupt Enable 0 ECC correctable error interrupt is disabled. 1 ECC correctable error interrupt is enabled.
18:24		Reserved
25	EIR0E	External IRQ 0 Interrupt Enable 0 An external IRQ 0 interrupt is disabled. 1 An external IRQ 0 interrupt is enabled.
26	EIR1E	External IRQ 1 Interrupt Enable 0 An external IRQ 1 interrupt is disabled. 1 An external IRQ 1 interrupt is enabled.
27	EIR2E	External IRQ 2 Interrupt Enable 0 An external IRQ 2 interrupt is disabled. 1 An external IRQ 2 interrupt is enabled.
28	EIR3E	External IRQ 3 Interrupt Enable 0 An external IRQ 3 interrupt is disabled. 1 An external IRQ 3 interrupt is enabled.
29	EIR4E	External IRQ 4 Interrupt Enable 0 An external IRQ 4 interrupt is disabled. 1 An external IRQ 4 interrupt is enabled.
30	EIR5E	External IRQ 5 Interrupt Enable 0 An external IRQ 5 interrupt is disabled. 1 An external IRQ 5 interrupt is enabled.
31	EIR6E	External IRQ 6 Interrupt Enable 0 An external IRQ 6 interrupt is disabled. 1 An external IRQ 6 interrupt is enabled.

9.5.3 UIC Critical Register (UIC0_CR)

Fields in UIC0_CR, which correspond to fields in UIC0_SRand UIC0_ER, determine whether an interrupt captured in corresponding UIC0_SR fields generates a non-critical or critical interrupt, if the interrupts are enabled in the corresponding UIC0_ER fields. The processor core handles critical interrupts when MSR[EE] = 1 and non-critical interrupts when MSR[CE]=1.

If a UIC0_CR field is set to 0, an enabled interrupt (captured in the corresponding UIC0_SR field and enabled in the corresponding UIC0_ER field) generates a non-critical interrupt signal to the processor core. If a UIC0_CR field is a 1, a critical interrupt signal is generated.

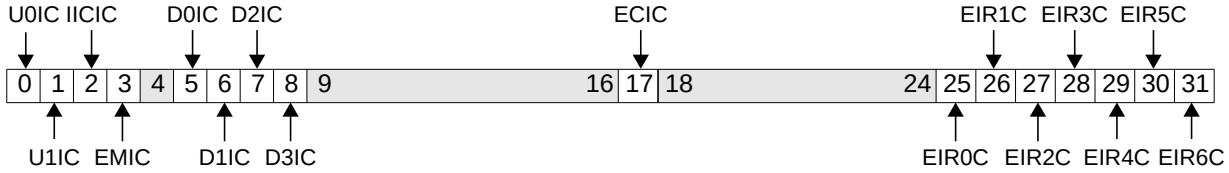


Figure 9-3. UIC Critical Register (UIC0_CR)

0	U0IC	UART0 Interrupt Class 0 UART0 interrupt is non-critical. 1 UART0 interrupt is critical.
1	U1IC	UART1 Interrupt Class 0 UART1 interrupt is non-critical. 1 UART1 interrupt is critical.
2	IICIC	IIC Interrupt Class 0 IIC interrupt is non-critical. 1 IIC interrupt is critical.
3	EMIC	External Master Interrupt Class 0 External master interrupt is non-critical. 1 External master interrupt is critical.
4		Reserved
5	D0IC	DMA Channel 0 Interrupt Class 0 DMA channel 0 interrupt is non-critical. 1 DMA channel 0 interrupt is critical.
6	D1IC	DMA Channel 1 Interrupt Class 0 DMA channel 1 interrupt is non-critical. 1 DMA channel 1 interrupt is critical.
7	D2IC	DMA Channel 2 Interrupt Class 0 DMA channel 2 interrupt is non-critical. 1 DMA channel 2 interrupt is critical.
8	D3IC	DMA Channel 3 Interrupt Class 0 DMA channel 3 interrupt is non-critical. 1 DMA channel 3 interrupt is critical.
9:16		Reserved

17	ECIC	ECC Correctable Error Interrupt Class 0 ECC correctable error interrupt is non-critical. 1 ECC correctable error interrupt is critical.
18:24		Reserved
25	EIR0C	External IRQ 0 Class 0 An external IRQ 0 interrupt is non-critical. 1 An external IRQ 0 interrupt is critical.
26	EIR1C	External IRQ 1 Class 0 An external IRQ 1 interrupt is non-critical. 1 An external IRQ 1 interrupt is critical.
27	EIR2C	External IRQ 2 Class 0 An external IRQ 2 interrupt is non-critical. 1 An external IRQ 2 interrupt is critical.
28	EIR3C	External IRQ 3 Class 0 An external IRQ 3 interrupt is non-critical. 1 An external IRQ 3 interrupt is critical.
29	EIR4C	External IRQ 4 Class 0 An external IRQ 4 interrupt is non-critical. 1 An external IRQ 4 interrupt is critical.
30	EIR5C	External IRQ 5 Class 0 An external IRQ 5 interrupt is non-critical. 1 An external IRQ 5 interrupt is critical.
31	EIR6C	External IRQ 6 Class 0 An external IRQ 6 interrupt is non-critical. 1 An external IRQ 6 interrupt is critical.

9.5.4 UIC Polarity Register (UIC0_PR))

Fields in UIC0_PR, which correspond to the fields in UIC0_SR, determine whether the interrupts associated with the corresponding fields in the UIC0_SR have a positive or negative polarity.

For level-sensitive interrupts, a 0 in a UIC0_PR field causes the corresponding interrupt to be negative active. A 1 in a UIC0_PR field causes the corresponding interrupt to be positive active.

For edge-sensitive interrupts, a 0 in a UIC0_PR field causes the corresponding interrupt to be detected on a falling edge (as polarity changes from 1 to 0). A 1 in a UIC0_PR field causes the corresponding interrupt to be detected on a rising edge (as polarity changes from 0 to 1).

Because the on-chip interrupts (those controlled by $UIC0\_PR_{0:3,\ 5:8,\ 17}$) have positive polarity, the associated fields must be set to 1.

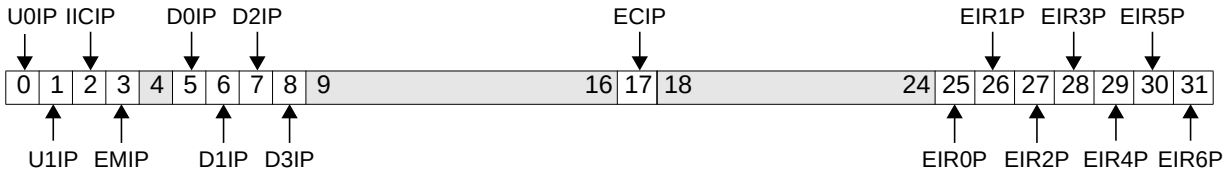


Figure 9-4. UIC Polarity Register (UIC0_PR)

0	U0IP	UART0 Interrupt Polarity 0 UART0 interrupt has negative polarity. 1 UART0 interrupt has positive polarity.	Must be set to 1.
1	U1IP	UART1 Interrupt Polarity 0 UART1 interrupt has negative polarity. 1 UART1 interrupt has positive polarity.	Must be set to 1.
2	IICIP	IIC Interrupt Polarity 0 IIC interrupt has negative polarity. 1 IIC interrupt has positive polarity.	Must be set to 1.
3	EMIP	External Master Interrupt Polarity 0 External master interrupt has negative polarity. 1 External master interrupt has positive polarity.	Must be set to 1.
4		Reserved	
5	D0IP	DMA Channel 0 Interrupt Polarity 0 DMA channel 0 interrupt has negative polarity. 1 DMA channel 0 interrupt has positive polarity.	Must be set to 1.
6	D1IP	DMA Channel 1 Interrupt Polarity 0 DMA channel 1 interrupt has negative polarity. 1 DMA channel 1 interrupt has positive polarity.	Must be set to 1.
7	D2IP	DMA Channel 2 Interrupt Polarity 0 DMA channel 2 interrupt has negative polarity. 1 DMA channel 2 interrupt has positive polarity.	Must be set to 1.
8	D3IP	DMA Channel 3 Interrupt Polarity 0 DMA channel 3 interrupt has negative polarity. 1 DMA channel 3 interrupt has positive polarity.	Must be set to 1.
9:16		Reserved	
17	ECIP	ECC Correctable Error Interrupt Polarity 0 ECC correctable error interrupt has negative polarity. 1 ECC correctable error interrupt has positive polarity.	Must be set to 1.
18:24		Reserved	
25	EIR0P	External IRQ 0 Polarity 0 An external IRQ 0 interrupt has negative polarity. 1 An external IRQ 0 interrupt has positive polarity.	
26	EIR1P	External IRQ 1 Polarity 0 An external IRQ 1 interrupt has negative polarity. 1 An external IRQ 1 interrupt has positive polarity.	

27	EIR2P	External IRQ 2 Polarity 0 An external IRQ 2 interrupt has negative polarity. 1 An external IRQ 2 interrupt has positive polarity.
28	EIR3P	External IRQ 3 Polarity 0 An external IRQ 3 interrupt has negative polarity. 1 An external IRQ 3 interrupt has positive polarity.
29	EIR4P	External IRQ 4 Polarity 0 An external IRQ 4 interrupt has negative polarity. 1 An external IRQ 4 interrupt has positive polarity.
30	EIR5P	External IRQ 5 Polarity 0 An external IRQ 5 interrupt has negative polarity. 1 An external IRQ 5 interrupt has positive polarity.
31	EIR6P	External IRQ 6 Polarity 0 An external IRQ 6 interrupt has negative polarity. 1 An external IRQ 6 interrupt has positive polarity.

9.5.5 UIC Trigger Register (UIC0_TR)

Fields in UIC0_TR, which correspond to fields in UIC0_SR, determine whether interrupts associated with the corresponding UIC0_SR fields are edge-sensitive or level-sensitive.

Edge-sensitive interrupts are triggered depending on whether the associated interrupt signal is rising or falling (changing from 0 to 1 or 1 to 0, respectively). Whether a rising or falling edge causes the trigger is controlled by bits in UIC0_TR.

Level-sensitive interrupts are triggered depending on whether the associated interrupt signal is high (1) or low (0).

If a UIC0_TR field is 0, the associated interrupt is level-sensitive. If the UIC0_TR field is 1, the interrupt is edge-sensitive. Because the on-chip interrupts (those controlled by $UIC0\_TR_{0:2,\ 5:8,\ 17}$) are level-sensitive, the associated fields must be set to 0. The external master interrupt trigger (controlled by $UIC0\_TR_3$) is edge-sensitive; this field must be set to 1.

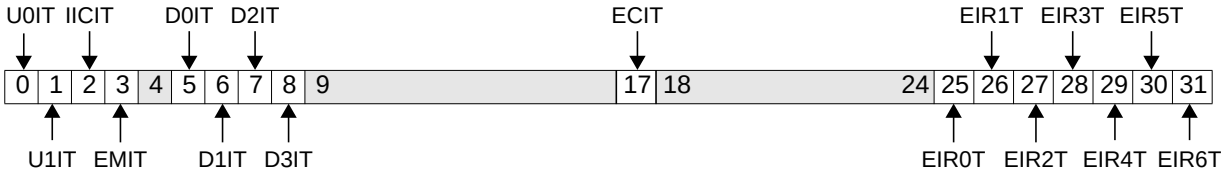


Figure 9-5. UIC Trigger Register (UIC0_TR)

0	U0IT	UART0 Interrupt Trigger 0 UART0 interrupt is level-sensitive. 1 UART0 interrupt is edge-sensitive.	Must be set to 0.
1	U1IT	UART1 Interrupt Trigger 0 UART1 interrupt is level-sensitive. 1 UART1 interrupt is edge-sensitive.	Must be set to 0.

2	IICIT	IIC Interrupt Trigger 0 IIC interrupt is level-sensitive. 1 IIC interrupt is edge-sensitive.	Must be set to 0.
3	EMIT	External Master Interrupt Trigger 0 External master interrupt is level-sensitive. 1 External master interrupt is edge-sensitive.	Must be set to 1.
4		Reserved	
5	D0IT	DMA Channel 0 Interrupt Trigger 0 DMA channel 0 interrupt is level-sensitive. 1 DMA channel 0 interrupt is edge-sensitive.	Must be set to 0.
6	D1IT	DMA Channel 1 Interrupt Trigger 0 DMA channel 1 interrupt is level-sensitive. 1 DMA channel 1 interrupt is edge-sensitive.	Must be set to 0.
7	D2IT	DMA Channel 2 Interrupt Trigger 0 DMA channel 2 interrupt is level-sensitive. 1 DMA channel 2 interrupt is edge-sensitive.	Must be set to 0.
8	D3IT	DMA Channel 3 Interrupt Trigger 0 DMA channel 3 interrupt is level-sensitive. 1 DMA channel 3 interrupt is edge-sensitive.	Must be set to 0.
9:16		Reserved	
17	ECIT	ECC Correctable Error Interrupt Trigger 0 ECC correctable error interrupt is level-sensitive. 1 ECC correctable error interrupt is edge-sensitive.	Must be set to 0.
18:24		Reserved	
25	EIR0T	External IRQ 0 Trigger 0 An external IRQ 0 interrupt is level-sensitive. 1 An external IRQ 0 interrupt is edge-sensitive.	
26	EIR1T	External IRQ 1 Trigger 0 An external IRQ 1 interrupt is level-sensitive. 1 An external IRQ 1 interrupt is edge-sensitive.	

27	EIR2T	External IRQ 2 Trigger 0 An external IRQ 2 interrupt is level-sensitive. 1 An external IRQ 2 interrupt is edge-sensitive.
28	EIR3T	External IRQ 3 Trigger 0 An external IRQ 3 interrupt is level-sensitive. 1 An external IRQ 3 interrupt is edge-sensitive.
29	EIR4T	External IRQ 4 Trigger 0 An external IRQ 4 interrupt is level-sensitive. 1 An external IRQ 4 interrupt is edge-sensitive.
30	EIR5T	External IRQ 5 Trigger 0 An external IRQ 5 interrupt is level-sensitive. 1 An external IRQ 5 interrupt is edge-sensitive.
31	EIR6T	External IRQ 6 Trigger 0 An external IRQ 6 interrupt is level-sensitive. 1 An external IRQ 6 interrupt is edge-sensitive.

9.5.6 UIC Masked Status Register (UIC0_MSR)

This read-only register contains the result of masking the UIC0_SR with UIC0_ER. Reading this register, instead of the actual UIC0_SR, eliminates the need for software to read and apply the enable mask to the contents of the UIC0_SR to determine which enabled interrupt fields are active.

If an interrupt is configured as level-sensitive, and a clear is attempted on the UIC0_SR, the UIC0_SR field is not cleared if the incoming interrupt signal is at the asserted polarity. The interrupt signal must be reset before UIC0_SR can be successfully cleared.

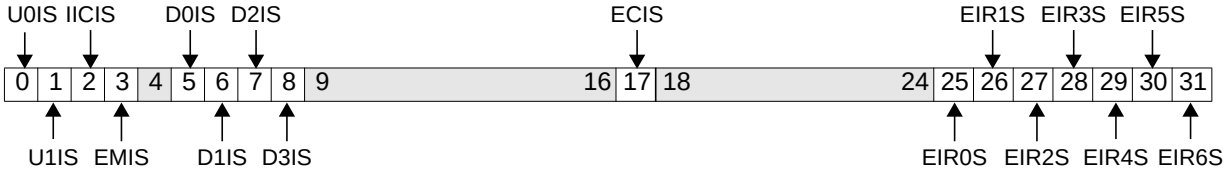


Figure 9-6. UIC Masked Status Register (UIC0_MSR)

0	U0IS	UART0 Masked Interrupt Status 0 A UART0 interrupt has not occurred. 1 A UART0 interrupt occurred.
1	U1IS	UART1 Masked Interrupt Status 0 A UART1 interrupt has not occurred. 1 A UART1 interrupt occurred.

2	IICIS	IIC Masked Interrupt Status 0 An IIC interrupt has not occurred. 1 An IIC interrupt occurred.
3	EMIS	External Master Masked Interrupt Status 0 An external master interrupt has not occurred. 1 An external master interrupt occurred.
4		Reserved
5	D0IS	DMA Channel 0 Masked Interrupt Status 0 A DMA channel 0 interrupt has not occurred. 1 A DMA channel 0 interrupt occurred.
6	D1IS	DMA Channel 1 Masked Interrupt Status 0 A DMA channel 1 interrupt has not occurred. 1 A DMA channel 1 interrupt occurred.
7	D2IS	DMA Channel 2 Masked Interrupt Status 0 A DMA channel 2 interrupt has not occurred. 1 A DMA channel 2 interrupt occurred.
8	D3IS	DMA Channel 3 Masked Interrupt Status 0 A DMA channel 3 interrupt has not occurred. 1 A DMA channel 3 interrupt occurred.
9:16		Reserved
17	ECIS	ECC Correctable Error Masked Interrupt Status 0 An ECC correctable error interrupt did not occur. 1 An ECC correctable error interrupt occurred.
18:24		Reserved
25	EIR0S	External IRQ 0 Masked Status 0 An external IRQ 0 interrupt has not occurred. 1 An external IRQ 0 interrupt occurred.
26	EIR1S	External IRQ 1 Masked Status 0 An external IRQ 1 interrupt has not occurred. 1 An external IRQ 1 interrupt occurred.
27	EIR2S	External IRQ 2 Masked Status 0 An external IRQ 2 interrupt has not occurred. 1 An external IRQ 2 interrupt occurred.
28	EIR3S	External IRQ 3 Masked Status 0 An external IRQ 3 interrupt has not occurred. 1 An external IRQ 3 interrupt occurred.

29	EIR4S	External IRQ 4 Masked Status 0 An external IRQ 4 interrupt has not occurred. 1 An external IRQ 4 interrupt occurred.
30	EIR5S	External IRQ 5 Masked Status 0 An external IRQ 5 interrupt has not occurred. 1 An external IRQ 5 interrupt occurred.
31	EIR6S	External IRQ 6 Masked Status 0 An external IRQ 6 interrupt has not occurred. 1 An external IRQ 6 interrupt occurred.

9.5.7 UIC Vector Configuration Register (UIC0_VCR)

The write-only UIC0_VCR enables software control of interrupt vector generation for critical interrupts. UIC0_VCR contains an address, used as an interrupt vector base address, and specifies interrupt ordering priority. Vector generation is not performed for non-critical interrupts.

UIC0_VCR[VBA] can contain either the base address for an interrupt handler vector table or the base address for the interrupt handler associated with each interrupt. The actual interrupt vector (the address of the interrupt handler that services the interrupt) is generated in the UIC0_VR, using UIC0_VCR[VBA]. Vector generation is described in “UIC Vector Register (UIC0_VR)” on page 9-15. Because the two lowest-order bits of an interrupt handler address are assumed to be 00 to ensure word alignment, 30 bits are sufficient to form the base address.

A general interrupt handler uses the vector to access a table of interrupt vectors. Each interrupt vector table entry contains the address of an interrupt handler for a specific interrupt. Alternatively, UIC0_VCR[VBA] can directly address the interrupt handlers for specific interrupts, which in memory are separated by an offset calculated in UIC0_VR.

UIC0_VCR[PRO] controls whether the interrupt associated with UIC0_SR[0] or UIC0_SR[31] has the highest priority. If UIC0_VCR[PRO] = 0, the interrupt associated with UIC0_SR[0] has the highest priority; if UIC0_VCR[PRO] = 1, the interrupt associated with UIC0_SR[31] has the highest priority. The bit closest to the highest priority field that is programmed in the UIC0_CR as a interrupt has the second highest priority. Priority decreases across the UIC0_SR to the end opposite the highest priority field.

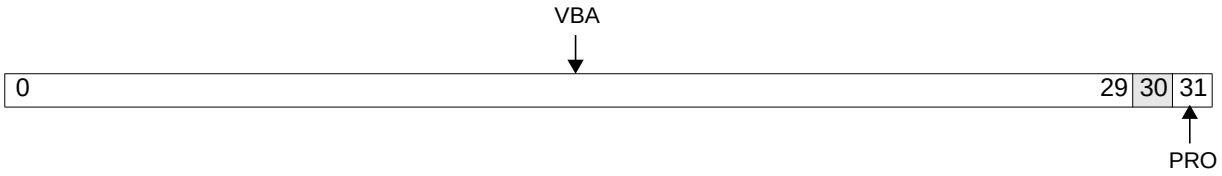


Figure 9-7. UIC Vector Configuration Register (UIC0_VCR)

0:29	VBA	Vector Base Address
30		Reserved

31	PRO	Priority Ordering 0 UIC0_SR[0] is the highest priority interrupt. 1 UIC0_SR[31] is the highest priority interrupt. Note: Vector generation is not performed for non-critical interrupts.

9.5.8 UIC Vector Register (UIC0_VR)

The read-only UIC0_VR contains an interrupt vector that can reduce interrupt handling latency for critical interrupts. Vector generation logic adds an offset to UIC0_VCR[VBA], and the sum is returned in the UIC0_VR. Vectors are not computed for non-critical interrupts.

The interrupt vector is based on the field position of the current highest priority, enabled, active, critical interrupt relative to the highest priority interrupt in the UIC0_SR. The generated vectors can be programmed to point directly to the interrupt handlers.

Programming Note: Regardless of the programming of UIC0_VCR and UIC0_VR registers, the processor always vectors to EVPR[0:16] || 0x100 when a critical interrupt occurs.

The interrupt vector offset is based on the bit position of the current highest priority, enabled, active, critical interrupt relative to the highest priority interrupt in the UIC0_SR. The offset has a fixed value of 512 per bit. The main critical interrupt handler can interpret the vector returned by UIC0_VR as the address of the interrupt handler for that interrupt, assuming the routine is 512 bytes or smaller. Alternatively, the main critical interrupt handler can interpret the vector as a look-up table entry for the address of the interrupt handler for that interrupt.

Figure 9-8. UIC Vector Register (UIC0_VR)

0:31		Interrupt Vector

The following example illustrates the generation of a UIC0_VR vector for external interrupt request IRQ2.

For the example, assume that UIC0_VCR[PRO] = 1, so that UIC0_SR[EIR6S] ($UIC0\_SR_{31}$)has the highest interrupt priority, and that UIC0_SR[EIR2S] ($UIC0\_SR_{27}$) is the current highest priority, enabled, active, critical interrupt. To generate the vector for the interrupt associated with UIC0_SR[EIR2S], internal logic multiplies the difference between the highest priority interrupt bit and the active enabled priority interrupt bit by 512. The interrupt vector offset is therefore $(31 - 27) \times 512 = 4 \times 512$. This offset is added to the base address in UIC0_VCR[VBA], and the UIC0_VR returns UIC0_VCR[VBA] + (4×512).

9.5.8.1 Using the Value in UIC0_VR as a Vector Address or Entry Table Lookup

If an interrupt handler is 512 bytes or smaller, system software can interpret the value returned in the UIC0_VR as an address. In this case, when the interrupt is received, the UIC0_VR is read and software simply jumps to the address represented by the UIC0_VR value. Alternatively, the routine can be at a different address, and system software can treat the value of the UIC0_VR as a pointer, storing the interrupt handler address in the UIC0_VR during system initialization. In this case, when the interrupt is handled, software must read the UIC0_VR, read the entry at the UIC0_VR value, and jump to the entry. Hardware has no knowledge of the method is used, which is determined by system software.

9.5.8.2 Vector Generation Scenarios

For the following sequence, assume that the interrupts are enabled and critical (vectors are not generated for disabled or non-critical interrupts). The sequence illustrates several scenarios for vector generation.

1. An intermediate priority interrupt goes active; its vector is stored in UIC0_VR.
2. A low priority interrupt goes active; UIC0_VR is unchanged.
3. Software reads the vector; UIC0_VR is unchanged.
4. Software resets the intermediate priority interrupt; UIC0_VR contains the vector for the low priority interrupt.
5. A high priority interrupt goes active; UIC0_VR contains the vector for the high priority interrupt.
6. Software resets the high priority interrupt; UIC0_VR contains the vector for the low priority interrupt.
7. Software resets the UIC0_ER field for the low priority interrupt, disabling it; UIC0_VR contains 0x00000000.
8. UIC0_CR is reprogrammed to make the low priority interrupt non-critical and UIC0_ER is reprogrammed to re-enable the low priority interrupt; UIC0_VR continues to contain 0x00000000.

9.6 Interrupt Handling in the Processor Core

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. Exceptions are generated by signals from internal and external peripherals, instructions, internal timer facilities, debug events, or error conditions.

Table 9-4, “Interrupt Vector Offsets,” on page 9-22, lists the interrupts handled by the PPC405CR in the order of interrupt vector offsets. Detailed descriptions of each interrupt follow, in the same order. Table 9-4 also provides an index to the descriptions.

Several registers support interrupt handling and control. “General Interrupt Handling Registers” on page 9-22 describes the general interrupt 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)

9.7 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 (but not required) for the saved instruction pointer to be something else, possibly prohibiting guaranteed software recovery.

Note that “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, if the interrupt was 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 interrupt 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 interrupt handling routine.
3. The instruction having its address reported to the interrupt 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 PPC405CR does not implement any imprecise interrupts. Refer to *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 of as synchronous, nor as precise or imprecise. 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.

9.8 Behavior of the PPC405CR Implementation

All interrupts, except for machine checks, 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 interrupt handling routine. Precise handling also implies that all instructions that precede the instruction whose address is reported to the interrupt 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 interrupt type.

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

Asynchronous precise interrupts include the critical and noncritical external interrupts, and can be caused by on-chip peripherals, timer facility interrupts, and some debug events.

In the PPC405CR, machine checks are handled as critical interrupts (see “Critical and Noncritical Interrupts” on page 9-21). If a machine check is associated with an instruction fetch, the critical

interrupt save/restore register contains the address of the instruction being fetched when the machine check occurred.

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

It is improper to declare an exception when an erroneous word 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 instruction-side machine check interrupt occurs only when execution is attempted. If an exception occurs, execution is suppressed, SRR2 contains the erroneous address, and the ESR indicates that an instruction-side machine check occurred. Although such an interrupt 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 PPC405CR interrupts 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 interrupt handling routine.
- All instructions that precede the instruction whose address is reported to the interrupt 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 interrupt type.

9.9 Interrupt Handling Priorities

The PPC405CR processor core handles only one interrupt at a time. Multiple simultaneous interrupts are handled in the priority order shown in Table 9-3 (assuming, of course, that the interrupt types are enabled).

Multiple interrupts can exist simultaneously, each of which requires the generation of an interrupt. The architecture does not provide for simultaneously reporting more than one interrupt of the same class (critical or non-critical). Therefore, interrupts are ordered with respect to each other. A masking mechanism is available for certain persistent interrupt types.

When an interrupt type is masked, and an event causes an exception which would normally generate an interrupt of that type, the exception *persists* as a *status* bit in a register. However, no interrupt is generated. Later, if the interrupt type is enabled (unmasked), and the exception status has not been cleared by software, the interrupt due to the original exception event is finally generated.

All asynchronous interrupt types can be masked. In addition, certain synchronous interrupt types can be masked.

Table 9-3. Interrupt Handling Priorities

Priority	Interrupt Type	Critical or Noncritical	Causing Conditions
1	Machine check—data	Critical	External bus error during data-side access
2	Debug—IAC	Critical	IAC debug event (in internal debug mode)
3	Machine check—instruction	Critical	Attempted execution of instruction for which an external bus error occurred during fetch
4	Debug—EXC, UDE	Critical	EXC or UDE debug event (in internal debug mode)
5	Critical interrupt input	Critical	Active level on the critical interrupt input
6	Watchdog timer—first time-out	Critical	Posting of an enabled first time-out of the watchdog timer in the TSR
7	Instruction TLB Miss	Noncritical	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 — ZPR[Z*n*] = 00	Noncritical	Instruction translation is active, execution access to the translated address is not permitted because ZPR[Z*n*] = 00 in user mode, and an attempt is made to execute the instruction
9	Instruction storage — TLB_entry[EX] = 0	Noncritical	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 — TLB_entry[G] = 1	Noncritical	Instruction translation is active, the page is marked guarded, and an attempt is made to execute the instruction
10	Program	Noncritical	Attempted execution of illegal instructions, TRAP instruction, privileged instruction in problem state
	System call	Noncritical	Execution of the **sc** instruction
11	Data TLB miss	Noncritical	Valid matching entry for the effective address and process ID of an attempted data access is not found in the TLB
12	Data storage—ZPR[Z*n*] = 00	Noncritical	Data translation is active and data-side access to the translated address is not permitted because ZPR[Z*n*] = 00 in user mode
13	Data storage—TLB_entry[WR] = 0	Noncritical	Data translation is active and write access to the translated address is not permitted because TLB_entry[WR] = 0
	Data storage—TLB_entry[U0] = 1 or SU0R[U*n*] = 1	Noncritical	Data translation is active and write access to the translated address is not permitted because TLB_entry[U0] = 1 or SU0R[U*n*] = 1
14	Alignment	Noncritical	**dcbz** to non-cachable address or write-through storage; non-word aligned **dcread**, **lwarx**, and **stwcx**, as described in Table 9-12
15	Debug—BT, DAC, DVC, IC, TIE	Critical	BT, DAC, DVC, IC, TIE debug event (in internal debug mode)

Table 9-3. Interrupt Handling Priorities (continued)

Priority	Interrupt Type	Critical or Noncritical	Causing Conditions
16	External interrupt input	Noncritical	Interrupts from the external interrupt (external to the processor core) input
17	Fixed Interval Timer (FIT)	Noncritical	Posting of an enabled FIT interrupt in the TSR
18	Programmable Interval Timer (PIT)	Noncritical	Posting of an enabled PIT interrupt in the TSR

9.10 Critical and Noncritical Interrupts

The PPC405CR processes interrupts as noncritical and critical. Twelve interrupts are defined as *noncritical*: data storage, instruction storage, an active external interrupt input, alignment, program, system call, programmable interval timer (PIT), fixed interval timer (FIT), data TLB miss, and instruction TLB miss. Five interrupts are defined as *critical*: machine check interrupts (instruction- and data-side), debug interrupts (any of the seven types), interrupts caused by an active critical interrupt input, and the first time-out from the watchdog timer.

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

If the PPC405CR was executing a multicycle instruction (multiply, divide, or cache operation), the instruction is terminated and its address is written in SRR0.

Aligned scalar loads/stores that are interrupted do not appear on the PLB. An aligned scalar load/store cannot be interrupted after it is requested on the PLB, so the Guarded (G) storage attribute does not need to prevent the interruption of an aligned scalar load/store.

To enhance performance, the DCU can respond to non-cachable load requests by retrieving a line instead of a word. This is controlled by CCR0[LWL]. Note, however, that If CCR0[LWL] = 1, and the target non-cachable region is also marked as guarded (the G storage attribute is set to 1), that the DCU will request on the PLB only those bytes requested by the CPU.

Load/store multiples, load/store string, and misaligned scalar loads/stores that cross a word boundary can be interrupted and restarted upon return from the interrupt handler.

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 interrupt; when the instruction restarts, the registers will simply be written again. Similarly, some of the target memory of a store instruction may have been written, and is written again 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 interrupt handling routine.

Interrupt handling routine instructions are fetched at an address determined by the interrupt type. The address of the interrupt handling routine is formed by concatenating the 16 high-order bits of the

EVPR and the interrupt vector offset. (A user must initialize the EVPR contents at power-up using an **mtspr** instruction.)

Table 9-4 shows the interrupt vector offsets for the interrupt types. Note that there can be multiple sources of the same interrupt type; interrupts of the same type are mapped to the same interrupt vector, regardless of source. In such cases, the interrupt handling routine must examine status registers to determine the exact source of the interrupt.

At the end of the interrupt 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 interrupts are processed similarly. When a critical interrupt 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 interrupt, and the contents of the MSR, respectively. At the end of the critical interrupt handling routine, execution of an **rfci** instruction writes the contents of SRR2 and SRR3 into the program counter and the MSR, respectively.

Table 9-4. Interrupt Vector Offsets

Offset	Interrupt Type	Interrupt Class	Category	Page
0x0100	Critical input interrupt	Asynchronous precise	Critical	9-29
0x0200	Machine check—data	—	Critical	9-30
	Machine check—instruction	—	Critical	9-30
0x0300	Data storage interrupt— MSR[DR]=1 and ZPR[Z*n*] = 0 or TLB_entry[WR] = 0 or TLB_entry[U0] = 1 or SU0R[U*n*] = 1	Synchronous precise	Noncritical	9-31
0x0400	Instruction storage interrupt	Synchronous precise	Noncritical	9-32
0x0500	External interrupt (external to the processor core)	Asynchronous precise	Noncritical	9-33
0x0600	Alignment	Synchronous precise	Noncritical	9-34
0x0700	Program	Synchronous precise	Noncritical	9-35
0x0C00	System Call	Synchronous precise	Noncritical	9-36
0x1000	PIT	Asynchronous precise	Noncritical	9-36
0x1010	FIT	Asynchronous precise	Noncritical	9-37
0x1020	Watchdog timer	Asynchronous precise	Critical	9-37
0x1100	Data TLB miss	Synchronous precise	Noncritical	9-38
0x1200	Instruction TLB miss	Synchronous precise	Noncritical	9-39
0x2000	Debug—BT, DAC, DVC, IAC, IC, TIE	Synchronous precise	Critical	9-39
	Debug—EXC, UDE	Asynchronous precise	Critical	

9.11 General Interrupt Handling Registers

The general interrupt 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).

9.11.1 Machine State Register (MSR)

The MSR is a 32-bit register that holds the current context of the PPC405CR. When a noncritical 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.

Programming Note: The **rfi** and **rfci** instructions can alter reserved MSR fields.

The MSR contents can be read into a general purpose register (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.

Figure 9-9 shows the MSR bit definitions and describes the function of each bit.

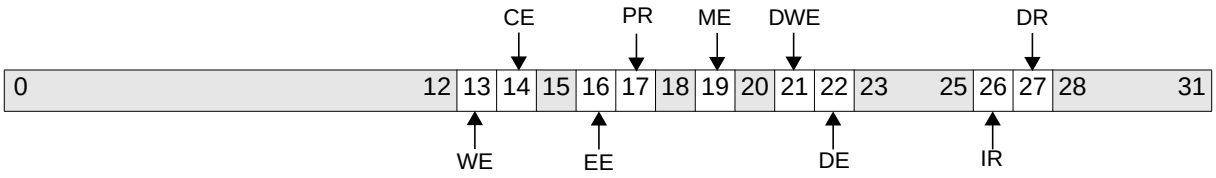


Figure 9-9. Machine State Register (MSR)

0:12		Reserved	
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 interrupt 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 time-out interrupts.
15		Reserved	
16	EE	External Interrupt Enable 0 Asynchronous interrupts (external to the processor core) are disabled. 1 Asynchronous interrupts are enabled.	Controls the non-critical external interrupt input, PIT, and FIT 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 interrupts are disabled. 1 Machine check interrupts are enabled.	
20		Reserved	

21	DWE	Debug Wait Enable 0 Debug wait mode is disabled. 1 Debug wait mode is enabled.
22	DE	Debug Interrupts Enable 0 Debug interrupts are disabled. 1 Debug interrupts are enabled.
23:25		Reserved
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.
27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.
28:31		Reserved

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

SRR0 and SRR1 are 32-bit registers that hold the interrupted machine context when a noncritical 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.

Figure 9-10 shows the bit definitions for SRR0.

.

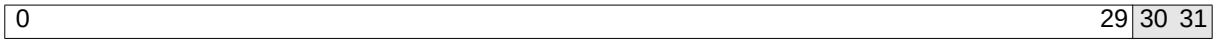


Figure 9-10. Save/Restore Register 0 (SRR0)

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

Figure 9-11 shows the bit definitions for SRR1.

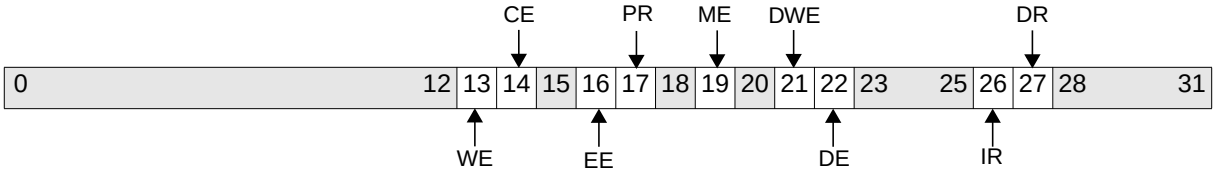


Figure 9-11. Save/Restore Register 1 (SRR1)

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

9.11.3 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.

Figure 9-12 shows the bit definitions for SRR2.

.

Figure 9-12. 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

Figure 9-13 shows the bit definitions for SRR3.

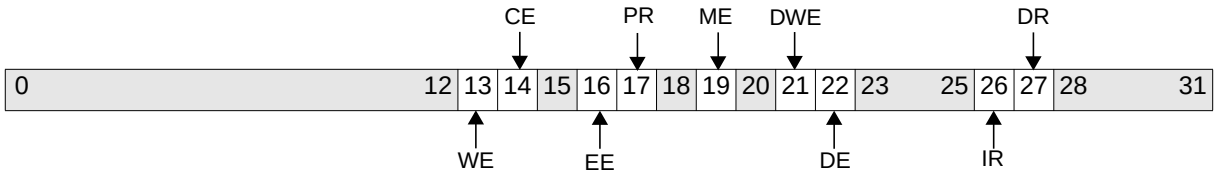


Figure 9-13. Save/Restore Register 3 (SRR3)

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

Because critical interrupts do not automatically clear MSR[ME], SRR2 and SRR3 can be corrupted by a machine check interrupt, if the machine check occurs while SRR2 and SRR3 contain valid data that has not yet been saved by the critical interrupt handler.

9.11.4 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 interrupt handling routine. The 16-bit interrupt vector offsets (shown in Table 9-4, “Interrupt Vector Offsets,” on page 9-22) are concatenated to the right of the high-order 16 bits of the EVPR to form the 32-bit address of an interrupt handling 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.

Figure 9-14 shows the EVPR bit definitions.

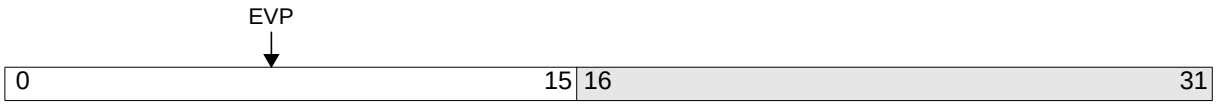


Figure 9-14. Exception Vector Prefix Register (EVPR)

0:15	EVP	Exception Vector Prefix
16:31		Reserved

9.11.5 Exception Syndrome Register (ESR)

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

“Instruction Machine Check Handling” on page 9-30 describes instruction machine checks. “Data Storage Interrupt” on page 9-31 describes data storage interrupts. “Program Interrupt” on page 9-35 describes program interrupts.

Although interrupt handling routines are not required to reset the ESR, it is recommended that instruction machine check handlers reset the ESR; “Instruction Machine Check Handling” on page 9-30 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.

Figure 9-15 shows the ESR bit definitions.

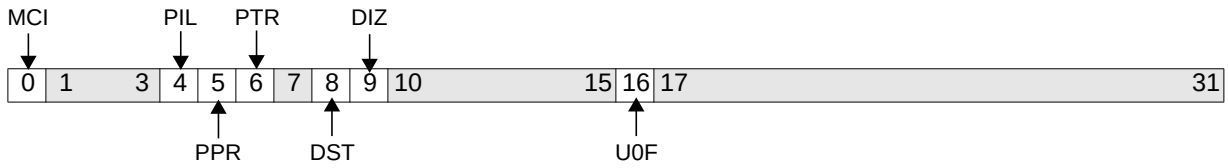


Figure 9-15. Exception Syndrome Register (ESR)

0	MCI	Machine check—instruction 0 Instruction machine check did not occur. 1 Instruction machine check occurred.
1:3		Reserved
4	PIL	Program interrupt—illegal 0 Illegal Instruction error did not occur. 1 Illegal Instruction error occurred.
5	PPR	Program interrupt—privileged 0 Privileged instruction error did not occur. 1 Privileged instruction error occurred.
6	PTR	Program interrupt—trap 0 Trap with successful compare did not occur. 1 Trap with successful compare occurred.
7		Reserved
8	DST	Data storage interrupt—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 interrupt—zone fault 0 Excepting condition was not a zone fault. 1 Excepting condition was a zone fault.
10:15		Reserved

16	U0F	Data storage interrupt—U0 fault 0 Excepting instruction did not cause a U0 fault. 1 Excepting instruction did cause a U0 fault.
17:31		Reserved

In general, ESR bits are set to indicate the type of precise interrupt that occurred; other bits are cleared. However, the machine check—instruction (ESR[MCI]) bit behaves differently. Because instruction-side machine checks can occur without an interrupt being taken (if MSR[ME] = 0), ESR[MCI] can be set even while other ESR-setting interrupts (program, data storage, DTLB-miss) occurring. Thus, data storage and program interrupts leave ESR[MCI] unchanged, clear all other ESR bits, and set the bits associated with any data storage or program interrupts occurred. Enabled instruction-side machine checks (MSR[ME] = 1) set ESR[MCI] and clear the data storage and program interrupt bits.

If a machine check—instruction interrupt occurs but is disabled (MSR[ME] = 0), it sets ESR[MCI] but leaves the data storage and program interrupt bits alone. If a machine check—instruction interrupt occurs while MSR[ME] = 0, *and* the instruction upon which the machine check—instruction interrupt is occurring also is some other kind of ESR-setting instruction (program, data storage, DTLB-miss, or instruction storage interrupt), ESR[MCI] is set to indicate that a machine check—instruction interrupt occurred; the other ESR bits are set or cleared to indicate the other interrupt. These scenarios are summarized in Table 9-5

Table 9-5. ESR Alteration by Various Interrupts

Scenario	ECR[MCI]	$ESR_{4:}$	$ESR_{8:9, 16}$
Program interrupt	Unchanged	Set to type	Cleared
Data storage interrupt	Unchanged	Cleared	Set to Type
Data TLB miss interrupt	Unchanged	Cleared	Cleared
Machine check—instruction	Set to 1	Cleared	Cleared
Disabled MCI, no others	Unchanged	Unchanged	Unchanged
Disabled MCI and program interrupt	Unchanged	Set to type	Cleared

9.11.6 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 interrupt.

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.

Figure 9-16 shows the DEAR bit definitions.

Figure 9-16. Data Exception Address Register (DEAR)

0:31		Address of Data Error (synchronous)

9.12 Critical Input Interrupts

The UICCR can be programmed so that any UIC interrupt can be presented as a critical interrupt input to the processor core. See “UIC Trigger Register (UIC0_TR)” on page 9-10 for details. Critical interrupts are recognized only if enabled by MSR[CE].

MSR[CE] also enables the watchdog timer first-time-out interrupt. However, the watchdog interrupt has a different interrupt vector than the critical pin interrupt. See “Watchdog Timer Interrupt” on page 9-37.

After detecting a critical interrupt, if no synchronous precise interrupts are outstanding, the PPC405CR immediately takes the critical interrupt 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 time-out interrupt from interrupting the critical interrupt handler before SRR2 and SRR3 get saved. MSR[DE] is reset to 0 to disable debug interrupts during the critical interrupt handler.

The MSR is also written with the values shown in Table 9-6, “Register Settings during Critical Input Interrupts,” on page 9-29. 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. Interrupt processing begins at the address in the program counter.

Inside the interrupt 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 9-6. Register Settings during Critical Input Interrupts

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

9.13 Machine Check Interrupts

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

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

When an instruction-side machine check interrupt occurs, the PPC405CR stores the address of the excepting instruction in SRR2. When a data-side machine check occurs, the PPC405CR stores the address of the next sequential instruction in SRR2. Simultaneously, for all machine check interrupts, 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 interrupt handling routine. The other MSR bits are loaded with the values shown in Table 9-7, “Register Settings during Machine Check—Instruction Interrupts,” on page 9-31 and Table 9-8, “Register Settings during Machine Check—Data Interrupts,” on page 9-31. 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. Interrupt 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.

9.13.1 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 interrupt occurs. If enabled by MSR[ME], the processor reports the machine check—instruction interrupt 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 interrupt. Taking the vector automatically clears MSR[ME] and the other MSR fields.

Note that it is improper to declare a machine check—instruction interrupt 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 interrupt will occur only if execution of the instruction is subsequently attempted.

When a machine check occurs on an instruction fetch, the erroneous instruction is never validated in the instruction cache unit (ICU). Fetch requests to cachable memory that miss in the ICU cause an instruction cache line fill (eight words). If any words in the fetched line are associated with an error, an interrupt occurs upon attempted execution and the cache line is invalidated. If any word 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 interrupt occurs while running in code in which MSR[ME] is disabled, the machine check—instruction interrupt 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 interrupt occurred during the disabled execution.

If a new machine check—instruction interrupt occurs after MSR[ME] is enabled again, the new machine check—instruction interrupt is recorded in ESR[MCI] and the machine check—instruction interrupt 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 interrupt occurred while MSR[ME] was disabled. The machine check—instruction interrupt 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 interrupts.

Table 9-7. Register Settings during Machine Check—Instruction Interrupts

SRR2	Written with the address that caused the machine check.
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, ME, DWE, DE,IR, DR←0
PC	EVPR[0:15] || 0x0200
ESR	MCI ← 1 All other bits are cleared.

9.13.2 Data Machine Check Handling

When a machine check occurs on an data access, a machine check—data interrupt occurs. To determine the cause of a machine check, examine the various error reporting registers of the external PLB slaves.

Table 9-8. Register Settings during Machine Check—Data Interrupts

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

9.14 Data Storage Interrupt

The data storage interrupt occurs when the desired access to the effective address is not permitted for any of the following reasons:

- A U0 fault: any store to an EA with the U0 storage attribute set and CCR0[U0XE] = 1
- In the 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** will no-op in this situation, rather than cause an interrupt. The instructions **dcbi**, **dccci**, **icbt**, and **iccci**, being privileged, cannot cause zone fault data storage interrupts.)
  - 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 will cause privileged program interrupts, rather than data storage interrupts.)
- In the 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.

Programming Note: The **icbi**, **icbt**, and **iccci** instructions are treated as loads from the addressed byte with respect to address translation and protection. Instruction cache operations

use MSR[DR], not MSR[IR], to determine translation of their operands. Instruction storage interrupts and Instruction-side TLB Miss Interrupts are associated with the *fetching* of instructions, not with the execution of instructions. Data storage interrupts and data TLB miss interrupts are associated with the *execution* of instruction cache operations.

When a data storage interrupt is detected, the PPC405CR suppresses the instruction causing the interrupt and writes the instruction address in SRR0. The Data Exception Address Register (DEAR) is loaded with the data address that caused the access violation. ESR bits are loaded as shown in Table 9-9, “Register Settings during Data Storage Interrupts,” on page 9-32 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 9-9.

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. Interrupt 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, and the PPC405CR resumes execution at the new program counter address.

For instructions that can simultaneously generate program interrupts (privileged instructions executed in Problem State) and data storage interrupts, the program interrupt has priority.

Table 9-9. Register Settings during Data Storage Interrupts

SRR0	Written with the EA of the instruction causing the data storage interrupt
SRR1	Written with the value of the MSR at the time of the interrupt
MSR	WE, EE, PR, DWE, IR, DR←0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x0300
DEAR	Written with the EA of the failed access
ESR	DST ← 1 if excepting operation is a store (includes **dcbi** and **dcbz**) DIZ ← 1 if access failure caused by a zone protection fault (ZPR[Z*n*] = 00 in user mode) U0F ← 1 if access failure caused by a U0 fault (the U0 storage attribute is set and CCR0[U0XE] = 1) MCI ← unchanged All other bits are cleared.

9.15 Instruction Storage Interrupt

The instruction storage interrupt 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 will save the address of the instruction causing the instruction storage interrupt.

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 interrupt 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 9-10. Register Settings during Instruction Storage Interrupts

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 interrupt
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x0400
ESR	DIZ ← 1If access failure due to a zone protection fault (ZPR[Z*n*] = 00 in user mode) **Note:** If ESR[DIZ] is not set, the interrupt occurred because TBL_entry[EX] was clear in an otherwise accessible zone, or because of an instruction fetch from a storage region marked as guarded. See “Exception Syndrome Register (ESR)” on page 9-26 for details of ESR operation. MCI ← unchanged All other bits are cleared.

9.16 External Interrupt

External interrupts (external to the processor core) are triggered by active levels for non-critical interrupts in the UIC. All external interrupting events are presented to the processor as a single external interrupt. External interrupts are enabled or disabled by MSR[EE].

> **Programming Note:** MSR[EE] also enables PIT and FIT interrupts. However, after timer interrupts, control passes to different interrupt vectors than for the interrupts discussed in the preceding paragraph. Therefore, these timer interrupts are described in “Programmable Interval Timer (PIT) Interrupt” on page 9-36 and “Fixed Interval Timer (FIT) Interrupt” on page 9-37.

9.16.1 External Interrupt Handling

When MSR[EE] = 1 (external interrupts are enabled), a noncritical external interrupt occurs, and this interrupt is the highest priority interrupt 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 noncritical external interrupt, MSR[EE] is set to 0. This disables other external interrupts from interrupting the interrupt handler before SRR0 and SRR1 are saved. The

MSR is also written with the other values shown in Table 9-11, “Register Settings during External Interrupts,” on page 9-34. 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. Interrupt 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 9-11. Register Settings during External Interrupts

SRR0	Written with the address of the next sequential instruction
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x0500

9.17 Alignment Interrupt

Alignment interrupts are caused by **dcbz** instructions to non-cachable or write-through storage and. Table 9-12 summarizes the instructions and conditions causing alignment interrupts.

Table 9-12. Alignment Interrupt Summary

Instructions Causing Alignment Interrupts	Conditions
dcbz	EA in non-cachable or write-through storage
dcread, **lwarx**, **stwcx**.	EA not word-aligned

Execution of an instruction causing an alignment interrupt 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 9-13, “Register Settings during Alignment Interrupts,” on page 9-34. 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. Interrupt 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

Alignment interrupts cannot be disabled. To avoid overwrites of SRR0 and SRR1 by alignment interrupts that occur within a handler, interrupt handlers should save these registers as soon as possible.

Table 9-13. Register Settings during Alignment Interrupts

SRR0	Written with the address of the instruction causing the alignment interrupt
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x0600

Table 9-13. Register Settings during Alignment Interrupts (continued)

DEAR	Written with the address that caused the alignment violation

9.18 Program Interrupt

Program interrupts are caused by attempting to execute:

- An illegal instruction
- A privileged instruction while in the problem state
- Executing a trap instruction with conditions satisfied

The ESR bits that differentiate these situations are listed and described in Table 9-14. When a program interrupt occurs, the appropriate bit is set and the others are cleared. These interrupts are not maskable.

Table 9-14. ESR Usage for Program Interrupts

Bits	Interrupts	Cause
ESR[PIL]	Illegal instruction	Opcode not recognized
ESR[PPR]	Privileged instruction	Attempt to use a privileged instruction in the problem state
ESR[PTR]	Trap	Excepting instruction is a trap

The program interrupt handler does not need to reset the ESR.

When one of the following occurs, the PPC405CR does not execute the instruction, but writes the address of the excepting instruction into SRR0:

- Attempted execution of a privileged instruction in problem state
- Attempted execution of an illegal instruction (including memory management instructions when memory management is disabled

Trap instructions can be used as a program interrupt or a debug event, or both (see “Debug Events” on page 11-16 for information about debug events). When a trap instruction is detected as a program interrupt, the PPC405CR writes the address of the trap instruction into SRR0. See **tw** on page 20-190 and **twi** on page 20-193 (both in Chapter 20, “Instruction Set”) for a detailed discussion of the behavior of trap instructions with various interrupts enabled.

After any program interrupt, the contents of the MSR are written into SRR1 and the MSR bits are written with the values shown in Table 9-15. 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. Interrupt 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 9-15. Register Settings during Program Interrupts

SRR0	Written with the address of the excepting instruction
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged

Table 9-15. Register Settings during Program Interrupts (continued)

PC	EVPR[0:15] || 0x0700
ESR	Written with the type of program interrupt. (See Table 9-14) MCI ← unchanged All other bits are cleared.

9.19 System Call Interrupt

System call interrupts occur when a **sc** instruction is executed. The PPC405CR 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 9-16. 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. Interrupt 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 9-16. Register Settings during System Call Interrupts

SRR0	Written with the address of the instruction following the **sc** instruction
SRR1	Written with the contents of the MSR
MSR	AP, APE, WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x0C00

9.20 Programmable Interval Timer (PIT) Interrupt

For a discussion of the PPC405CR timer facilities, see Chapter 10, “Timer Facilities.” The PIT is described in “Programmable Interval Timer (PIT)” on page 10-4.

If the PIT interrupt is enabled by TCR[PIE] and MSR[EE], the PPC405CR initiates a PIT interrupt after detecting a time-out from the PIT. Time-out 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 PPC405CR 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 9-17. 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. Interrupt processing begins at the address in the program counter.

To clear a PIT interrupt, the interrupt handling routine must clear the PIT interrupt bit, TSR[PIS]. Clearing is performed by writing a word 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 9-17. Register Settings during Programmable Interval Timer Interrupts

SRR0	Written with the address of the next instruction to be executed
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x1000
TSR	PIS ← 1

9.21 Fixed Interval Timer (FIT) Interrupt

For a discussion of the PPC405CR timer facilities, see Chapter 10, “Timer Facilities.” The FIT is described in “Fixed Interval Timer (FIT) Interrupt” on page 9-37.

If the FIT interrupt is enabled by TCR[FIE] and MSR[EE], the PPC405CR initiates a FIT interrupt after detecting a time-out from the FIT. Time-out 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 PPC405CR 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 9-18. 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. Interrupt processing begins at the address in the program counter.

To clear a FIT interrupt, the interrupt handling routine must clear the FIT interrupt bit, TSR[FIS]. Clearing is performed by writing a word 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 9-18. Register Settings during Fixed Interval Timer Interrupts

SRR0	Written with the address of the next sequential instruction
SRR1	Written with the contents of the MSR
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x1010
TSR	FIS ← 1

9.22 Watchdog Timer Interrupt

For a general description of the PPC405CR timer facilities, see Chapter 10, “Timer Facilities.” The watchdog timer (WDT) is described in “Watchdog Timer” on page 10-6.

If the WDT interrupt is enabled by TCR[WIE] and MSR[CE], the PPC405CR initiates a WDT interrupt after detecting the first WDT time-out. First time-out 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 PPC405CR 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 9-19. 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. Interrupt processing begins at the address in the program counter.

To clear the WDT interrupt, the interrupt handling routine must clear the WDT interrupt bit TSR[WIS]. Clearing is done by writing a word 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 PPC405CR resumes execution at the contents of the program counter.

Table 9-19. Register Settings during Watchdog Timer Interrupts

SRR2	Written with the address of the next sequential instruction
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, DE, DWE, IR, DR ← 0 ME ← unchanged
PC	EVPR[0:15] || 0x1020
TSR	WIS ← 1

9.23 Data TLB Miss Interrupt

The data TLB miss interrupt 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 interrupt is precise. Program flow vectors to EVPR[0:15] || 0x1100.

The following registers are modified to the values specified in Table 9-20.

Table 9-20. Register Settings during Data TLB Miss Interrupts

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 interrupt
MSR	WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x1100
DEAR	Set to the effective address of the failed access

Table 9-20. Register Settings during Data TLB Miss Interrupts (continued)

ESR	DST ← 1 if excepting operation is a store operation (includes **dcbi**, **dcbz**, and **dccci**). MCI ← unchanged All other bits are cleared.

Programming Note: Data TLB miss interrupts can happen whenever data translation is enabled. Therefore, ensure that SRR0 and SRR1 are saved before enabling translation in an interrupt handler.

9.24 Instruction TLB Miss Interrupt

The instruction TLB miss interrupt 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 interrupt is precise with respect to the attempted execution of the instruction. Program flow vectors to EVPR[0:15 || 0x1200.

The following are modified to the values specified in Table 9-21

Table 9-21. Register Settings during Instruction TLB Miss Interrupts

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 interrupt
MSR	AP, APE, WE, EE, PR, DWE, IR, DR ← 0 CE, ME, DE ← unchanged
PC	EVPR[0:15] || 0x1200

Programming Note: Instruction TLB miss interrupts can happen whenever instruction translation is active. Therefore, insure that SRR0 and SRR1 are saved before enabling translation in an interrupt handler.

9.25 Debug Interrupt

Debug interrupts can be either *synchronous* or *asynchronous*. These debug events generate synchronous interrupts: branch taken (BT), data address compare (DAC), data value compare (DVC), instruction address compare (IAC), instruction completion (IC), and trap instruction (TIE). The exception (EXC) and unconditional (UDE) debug events generate asynchronous interrupts. See “Debug Events” on page 11-16 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 9-22.

Table 9-22. SRR2 during Debug Interrupts

Debug Event	Address Saved in SRR2
BT DAC IAC TIE	Address of the instruction causing the event

Table 9-22. SRR2 during Debug Interrupts

Debug Event	Address Saved in SRR2
DVC IC	Address of the instruction *following* the instruction that causing the event
EXC	Interrupt vector address of the initial exception that caused the exception debug event
UDE	Address of next instruction to be executed at time of UDE

SRR3 is written with the contents of the MSR and the MSR is written with the values shown in Table 9-23, “Register Settings during Debug Interrupts,” on page 9-40. 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. Interrupt 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 9-23. Register Settings during Debug Interrupts

SRR2	Written with an address as described in Table 9-22
SRR3	Written with the contents of the MSR
MSR	WE, CE, EE, PR, DE, DWE, IR, DR ← 0 ME ← unchanged
PC	EVPR[0:15] || 0x2000
DBSR	Set to indicate type of debug event.

Chapter 10. Timer Facilities

The PPC405CR provides four timer facilities: a time base, a Programmable Interval Timer (PIT), a fixed interval timer (FIT), and a watchdog timer. The PIT is a Special Purpose Register (SPR). These facilities, which are driven by the same base clock, can, among other things, be used for:

- Time-of-day functions
- Data logging functions
- Peripherals requiring periodic service
- Periodic task switching

Additionally, the watchdog timer can help a system to recover from faulty hardware or software.

Figure 10-1 shows the relationship of the timers and the clock source to the time base.

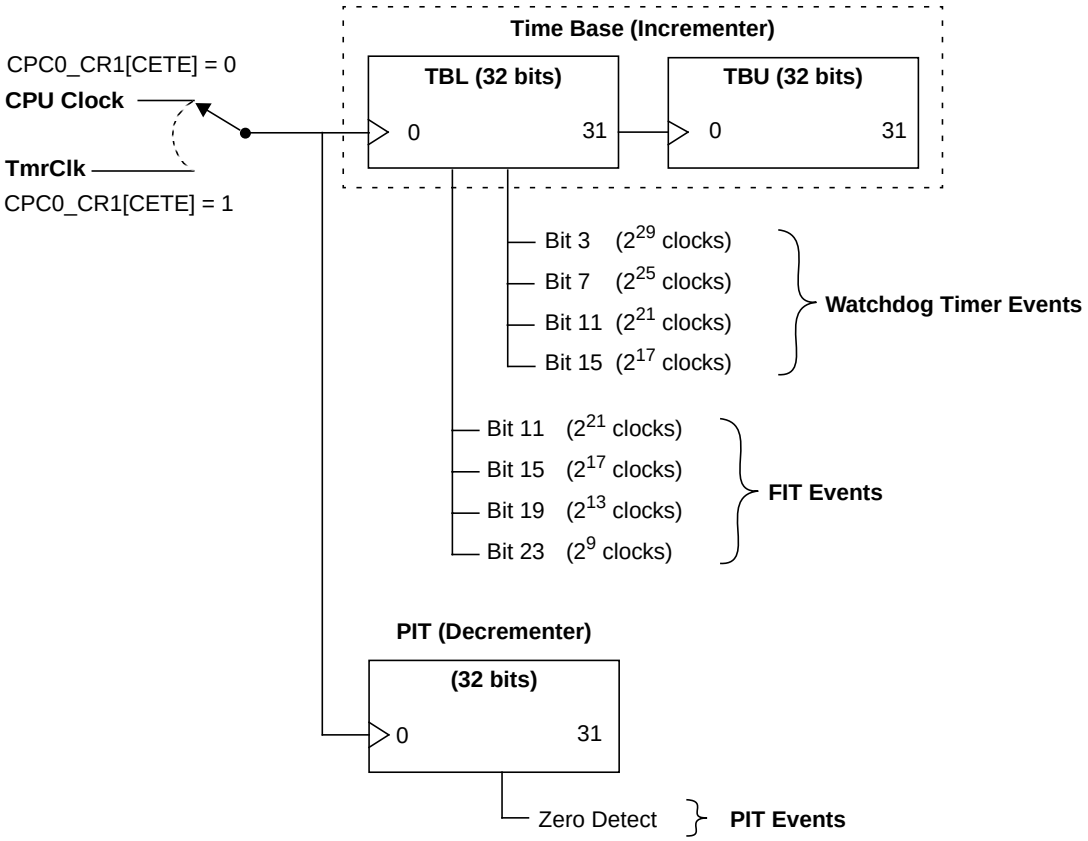


Figure 10-1. Relationship of Timer Facilities to the Time Base

10.1 Time Base

The PPC405CR implements a 64-bit time base as required in *The PowerPC Architecture*. The time base, which increments once during each period of the source clock, provides a time reference. The source clock is either the CPU clock or an external clock connected to the TmrClk input. The CETE field of Chip Control Register 1 (CPC0_CR1[CETE]) selects the clock source.

Read access to the time base is through the **mftb** instruction. **mftb** provides user-mode read-only access to the time base. The TBR numbers (0x10C and 0x10D; TBL and TBU, respectively) that specify the time base registers to **mftb** are not SPR numbers. However, the PowerPC Architecture allows an implementation to handle **mftb** as **mfspr**. Accordingly, these register numbers cannot be used for other SPRs. PowerPC compilers cannot use **mftb** with register numbers other than those specified in the PowerPC Architecture as read-access time base registers (0x10C and 0x10D).

Write access to the time base, using **mtspr**, is privileged. Different register numbers are used for read access and write access. Writing the time base is accomplished by using SPR 0x11C and SPR 0x11D (TBL and TBU, respectively) as operands for **mtspr**.

The period of the 64-bit time base is approximately 2925 years for a 200 MHz clock source. The time base does not generate interrupts, even when it wraps. For most applications, the time base is set once at system reset and only read thereafter. Note that the FIT and the watchdog timer (discussed below) are driven by 0→1 transitions of bits from the TBL. Transitions caused by software alteration of TBL have the same effect as transitions caused by normal incrementing of the time base.

Figure 10-2 illustrates the TBL.

Figure 10-2. Time Base Lower (TBL)

0:31		Time Base Lower	Current count; low-order 32 bits of time base.

Figure 10-3 illustrates the TBU.

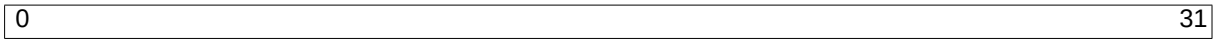


Figure 10-3. Time Base Upper (TBU)

0:31		Time Base Upper	Current count, high-order 32 bits of time base.

Table 10-1 summarizes the TBRs, instructions used to access the TBRs, and access restrictions.

Table 10-1. Time Base Access

Instructions		Register Number	Access Restrictions
TBU Upper 32 bits	mftbu RT *Extended mnemonic for* **mftb RT,TBU**	0x10D	Read-only
	mttbu RS *Extended mnemonic for* **mtspr TBU,RS**	0x11D	Privileged; write-only
TBL Lower 32 bits	mftb RT *Extended mnemonic for* **mftb RT,TBL**	0x10C	Read-only
	mttbl *Extended mnemonic for* **mtspr TBL,RS**	0x11C	Privileged; write-only

10.1.1 Reading the Time Base

The following code provides an example of reading the time base. **mftb** moves the low-order 32 bits of the time base to a GPR; **mftbu** moves the high-order 32 bits of the time base to a second GPR.

```
loop:
        mftbu Rx            # load from TBU
        mftb  Ry            # load from TBL
        mftbu Rz            # load from TBU
        cmpw Rz, Rx         # see if old = new
        bne   loop          # loop/reread if rollover occurred
```

The comparison and loop ensure that a consistent pair of values is obtained.

10.1.2 Writing the Time Base

The following code provides an example of writing the time base. Writing the time base is privileged. **mttbl** moves the contents of a GPR to the low-order 32 bits of the time base; **mttbu** moves the contents of a second GPR to the high-order 32 bits of the time base.

```
lwz     Rx, upper               # load 64-bit time base value into Rx and Ry
lwz     Ry, lower
li      Rz, 0
mttbl   Rz                      # force TBL to 0 to avoid rollover while writing TBU
mttbu   Rx                      # set TBU
mttbl   Ry                      # set TBL
```

10.2 Programmable Interval Timer (PIT)

The PIT is a 32-bit SPR that decrements at the same rate as the time base. The PIT is read and written using **mfspr** and **mtspr**, respectively. Writing to the PIT also 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 occurs:

1. If the PIT is in auto-reload mode (the ARE field of the Timer Control Register (TCR) is 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 1 simply causes a PIT content of 0.
2. TSR[PIS] is set to 1.
3. If enabled (TCR[PIE] = 1 and the EE field of the Machine State Register (MSR) is 1), a PIT interrupt is taken. See “Programmable Interval Timer (PIT) Interrupt” on page 9-36 for details of register behavior during a PIT interrupt.

The interrupt handler should use software to reset the PIS field of the Timer Status Register (TSR). This is done by using **mtspr** to write a word 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 PIT interrupt to become pending, a near-simultaneous decrement to 0 might have done so.
4. Write a 1 to TSR[PIS] (PIT Interrupt Status bit). This clears TSR[PIS] to 0 (see “Timer Status Register (TSR)” on page 10-8). This also clears any pending PIT interrupt. Because the PIT stops decrementing, no further PIT events are possible.

If the auto-reload feature is disabled (TCR[ARE] = 0) when the PIT decrements to 0, the PIT remains 0 until software uses **mtspr** to reload it.

After a reset, TCR[ARE] = 0, which disables the auto-reload feature.

Figure 10-4 illustrates the PIT.

Figure 10-4. Programmable Interval Timer (PIT)

0:31		Programmed interval remaining	Number of clocks remaining until the PIT event

10.2.1 Fixed Interval Timer (FIT)

The FIT provides timer interrupts having a repeatable period. The FIT is functionally similar to an auto-reload PIT, except that only a smaller fixed selection of interrupt periods are available.

The FIT exception occurs on 0→1 transitions of selected bits from the time base, as shown in Table 10-2.

Table 10-2. FIT Controls

TCR[FP]	TBL Bit	Period (Time Base Clocks)	Period (200 Mhz Clock)
0, 0	23	2^9 clocks	2.56 µsec
0, 1	19	2^{13} clocks	40.96 µsec
1, 0	15	2^{17} clocks	0.655 msec
1, 1	11	2^{21} clocks	10.49 msec

The TSR[FIS] field 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. “Fixed Interval Timer (FIT) Interrupt” on page 9-37 describes register settings during a FIT interrupt.

The interrupt handler should reset TSR[FIS]. This is done by using **mtspr** to write a word 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.

10.3 Watchdog Timer

The watchdog timer aids system recovery from software or hardware faults.

A watchdog timeout occurs on 0→1 transitions of a selected bit from the time base, as shown in the following table.

Table 10-3. Watchdog Timer Controls

TCR[WP]	TBL Bit	Period (Time Base Clocks)	Period (200 MHz Clock)
0,0	15	2^{17} clocks	0.655 msec
0,1	11	2^{21} clocks	10.49 msec
1,0	7	2^{25} clocks	0.168 sec
1,1	3	2^{29} clocks	2.684 sec

If a watchdog timeout occurs while TSR[WIS] = 0 and TSR[ENW] = 1, a watchdog interrupt occurs if the interrupt is enabled by TCR[WIE] and MSR[CE]. “Watchdog Timer” on page 10-6 describes register behavior during a watchdog interrupt.

The interrupt handler should reset the TSR[WIS] bit. This is done by using **mtspr** to write a word 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 watchdog timeout occurs while TSR[WIS] = 1 and TSR[ENW] = 1, a hardware reset occurs if enabled by a non-zero value of TCR[WRC]. In other words, a reset can occur if a watchdog timeout occurs while a previous watchdog timeout is pending. 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 watchdog timer reset capability. After a reset, the initial value of TCR[WRC] = 00.

Figure 10-5 describes the watchdog state machine. In the figure, numbers in parentheses refer to descriptions of operating modes that follow the table.

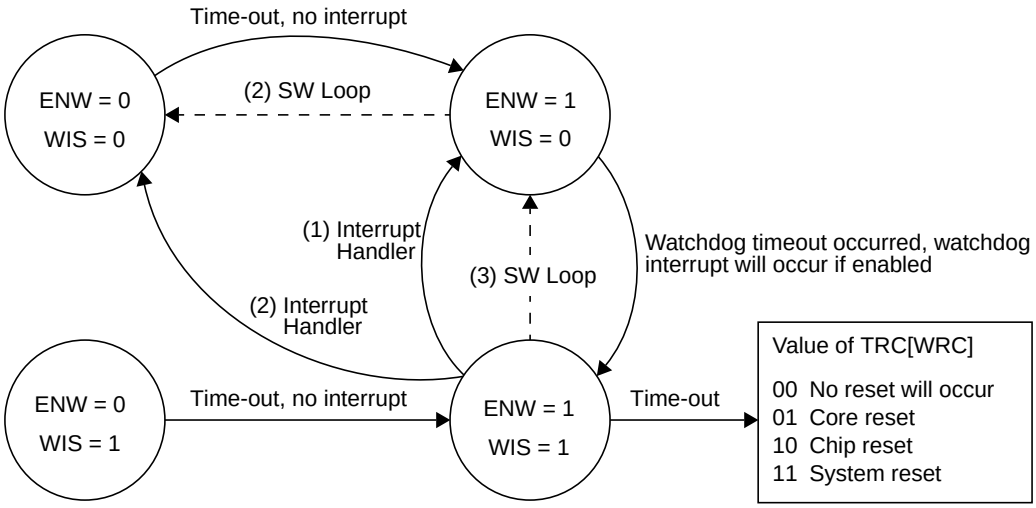


Figure 10-5. Watchdog Timer State Machine

Enable Next Watchdog TSR[ENW]	Watchdog Timer Status TSR[WIS]	Action When Timer Interval Expires
0	0	Set TSR[ENW] = 1.
0	1	Set TSR[ENW] = 1.
1	0	Set 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 current TCR[WRC] to TSR[WRS] and clear TCR[WRC], disabling the watchdog timer.

The controls described in Figure 10-5 imply three different ways of using the watchdog timer. The modes assume that TCR[WRC] was set to allow processor reset by the watchdog timer:

1. Always take a pending watchdog interrupt, and never attempt to prevent its occurrence. (This mode is described in the preceding text.)

   a. Clear TSR[WIS] in the watchdog timer handler.

   b. Never use TSR[ENW].

2. Always take a pending watchdog interrupt, but avoid it whenever possible by delaying a reset until a second watchdog timer occurs.

   This assumes that a recurring code loop of known maximum 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.

   a. Clear TSR[ENW] to 0 in loop or in FIT interrupt 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.

b. Clear TSR[WIS] in watchdog timer handler.

It is not expected that a watchdog interrupt will occur every time, but only if an exceptionally high execution load delays clearing of TSR[ENW] in the usual time frame.

3. Never take a watchdog 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 only guarantees one watchdog timeout period before a reset occurs.

a. Clear TSR[WIS] in the loop or in FIT handler.

b. Never use TSR[ENW] but have it set.

10.4 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 a word to the TSR, using **mtspr,** having a 1 in all fields to be cleared and a 0 in all other fields. The data written to the TSR is not direct data, but a mask. A 1 clears the field and a 0 has no effect.

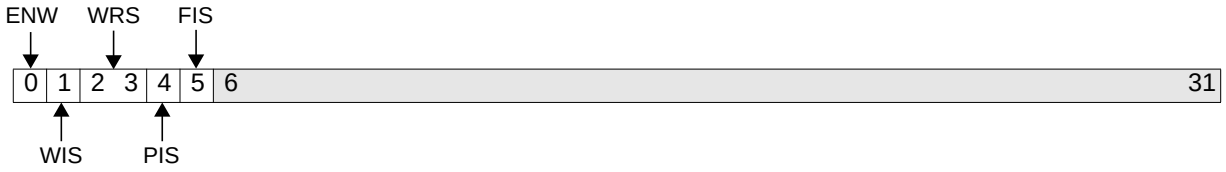


Figure 10-6. Timer Status Register (TSR)

0	ENW	Enable Next Watchdog 0 Action on next watchdog event is to set TSR[ENW] = 1. 1 Action on next watchdog event is governed by TSR[WIS].	Software must reset TSR[ENW] = 0 after each watchdog timer event.
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

10.5 Timer Control Register (TCR)

The TCR controls PIT, FIT, and watchdog timer operation.

The TCR[WRC] field is cleared to 0 by all processor resets. (Chapter 7, “Reset and Initialization,” 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 watchdog timer reset function.

All processor resets clear TCR[ARE] to 0, disabling the auto-reload feature of the PIT.

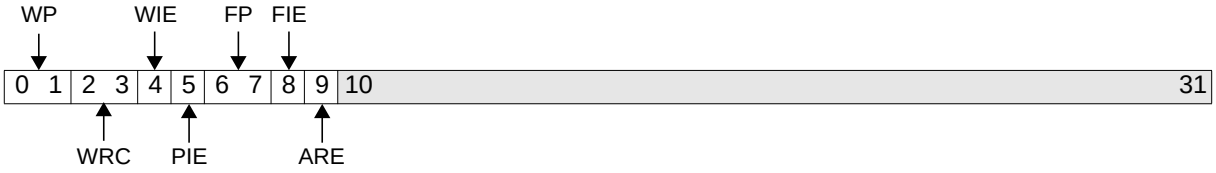


Figure 10-7. Timer Control Register (TCR)

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 will occur. 01 Core reset will be forced by the Watchdog. 10 Chip reset will be forced by the Watchdog. 11 System reset will be 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 watchdog interrupt. 1 Enable watchdog 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 Disables on reset. 0 Disable auto reload. 1 Enable auto reload.
10:31		Reserved

Chapter 11. Debugging

The debug facilities of the PPC405CR include support for debug modes for debugging during hardware and software development, and 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.

11.1 Development Tool Support

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. The RISCTrace™ feature of RISCWatch is an example of a development tool that uses the real-time instruction trace capability of the PPC405CR.

11.2 Debug Interfaces

The PPC405CR provides JTAG and trace interfaces to support hardware and software test and debug. Typically, the JTAG interface connects to a debug port external to the PPC405CR; the debug port is typically connected to a JTAG connector on a processor board.

The trace interface connects to a trace port, also external to the PPC405CR, that is typically connected to a trace connector on the processor board.

11.3 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 Std 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	The JTAG debug port implements the four required JTAG signals: TCK, TMS, TDI, and TDO, and the optional $\overline{\text{TRST}}$ signal.
JTAG Clock Requirements	The frequency of the TCK signal can range from DC to one-half of the internal chip clock frequency.
JTAG Reset Requirements	The JTAG debug port logic is reset at the same time as a system reset. Upon receiving $\overline{\text{TRST}}$, the JTAG debug port returns to the Test-Logic Reset state.

11.4 JTAG Connector

A 16-pin male 2x8 header connector is suggested as the JTAG debug port connector. This connector definition matches the requirements of the RISCWatch debugger from IBM. The connector is shown in Figure 11-1 and the signals are shown in Table 11-1. The connector should be placed as close as possible to the chip to ensure signal integrity.

Note that position 14 does not contain a pin.

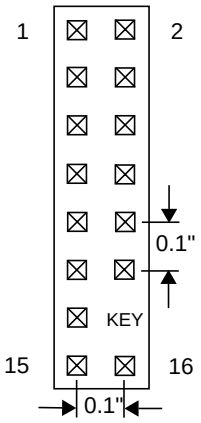


Figure 11-1. JTAG Connector Physical Layout (Top View)

Table 11-1. 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 I/O Voltage
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 pullup resistor should be connected to this signal to reduce chip power consumption. The pullup resistor is not required.
2. The +POWER signal, sourced from the target development board, indicates whether the I/O voltage at which 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 (note that the PPC405CR can have either +5V or +3.3V I/O, but the processor itself must be powered by +3.3V). A series resistor (1K ohm or less) should be used to provide short circuit current-limiting protection.
3. A 10K ohm pullup resistor must be connected to these signals to ensure proper chip operation when these inputs are not used.

11.4.1 JTAG Instructions

The JTAG debug port provides the standard *extest*, *idcode*, *sample/preload*, and *bypass* instructions and the optional *highz* and *clamp* instructions. Invalid instructions behave as the *bypass* instruction.

Table 11-2. JTAG Instructions

Instruction	Code	Comments
Extest	1111000	IEEE 1149.1 standard.
	1111001	Reserved.
Sample/Preload	1111010	IEEE 1149.1 standard.
IDCode	1111011	IEEE 1149.1 standard.
Private	xxxx100	Private instructions
HighZ	1111101	IEEE 1149.1a-1993 optional
Clamp	1111110	IEEE 1149.1a-1993 optional
Bypass	1111111	IEEE 1149.1 standard.

11.4.2 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. BSDL 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; 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.

11.4.3 JTAG Implementation

PPC405CR JTAG interface I/Os (TDI, TDO, TMs, TCK, and $\overline{\text{TRST}}$) are 5V tolerant and do not contain internal pull up resistors.

The optional JTAG instructions, *idcode* and *highz*, offer additional JTAG functionality. The *idcode* instruction returns the PPC405CR JTAG ID, which is unique for each chip version. The *highz* instruction disables all chip outputs regardless of whether they are included in the JTAG boundary scan chain.

The PPC405CR provides boundary scan structures on most I/O signals. However, the following signals are excluded because of speed and functional considerations:

- DrvrInh1
- DrvrInh2
- RcvrInh
- TestEn

11.4.4 JTAG ID Register (CPC0_JTAGID)

CPC0_JTAGID is a Device Control Register that enables manufacturing, part number, and version information to be determined through the TAP. The **mfdcr** instruction is used to read this register.

Refer to *PowerPC 405CR Embedded Processor Data Sheet* for the values of the CPC0_JTAGID fields.

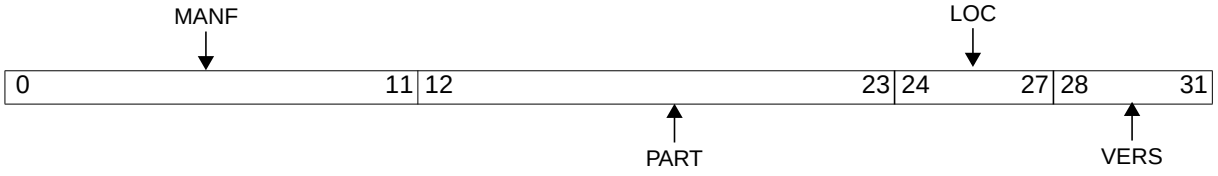


Figure 11-2. JTAG ID Register (CPC0_JTAGID)

0:11	MANF	Manufacturer Identifier
12:23	PART	Part Number
24:27	LOC	Developer Location
28:31	VERS	Version

11.5 Trace Port

The PPC405CR implements a trace status interface to support the tracing of code running in real-time. This interface enables the connection of an external trace tool, such as RISCWatch, and allows for user-extended trace functions. A software tool with trace capability, such as RISCWatch with RISCTrace, can use the data collected from this port to trace code running on the processor. The result is a trace of the code executed, including code executed out of the instruction cache if it was enabled. Information on trace capabilities, how trace works, and how to connect the external trace tool is available in *RISCWatch Debugger User's Guide*.

A 20-pin male 2x10 header connector is recommended for connecting to the trace status port of the PPC405CR. The connector, shown in Figure 11-3, and the signal descriptions in Table 11-3 match the requirements of RISCTrace, when used with the RISCWatch processor probe with RISCTrace option.

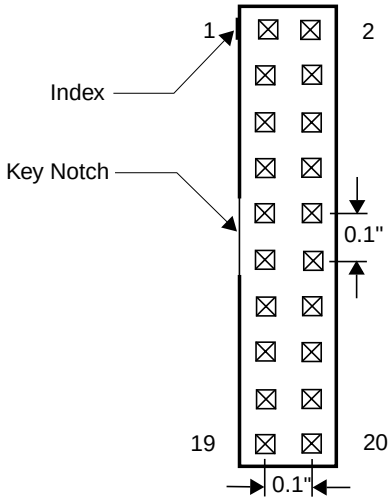


Figure 11-3. RISCTrace Header (Top View)

Table 11-3 describes the assignment of the RISCTrace signals and the processor core outputs to the header pins:

Table 11-3. RISCTrace Header Pin Description

Pin	Signal Name	Pin	Signal Name
1	No connect	11	No connect
2	No connect	12	TS1o (odd)
3	TrcClk	13	TS2o
4	No connect	14	TS1e (even)
5	No connect	15	TS2e
6	No connect	16	TS3
7	No connect	17	TS4
8	No connect	18	TS5
9	No connect	19	TS6

Table 11-3. RISCTrace Header Pin Description (continued)

Pin	Signal Name	Pin	Signal Name
10	No connect	20	GND

11.6 Debug Modes

The PPC405CR supports the following debug modes, each of which supports a type of debug tool or debug task commonly used in embedded systems development:

- Internal debug mode, which supports ROM monitors
- External debug mode, which supports JTAG debuggers
- Debug wait mode, which supports processor stopping or stepping for JTAG debuggers while servicing interrupts
- Real-time trace mode, which supports trigger events for real-time tracing

Internal and external debug modes can be enabled simultaneously. Both modes are controlled by fields in Debug Control Register 0 (DBCR0). Real-time trace mode is available only if internal, external, and debug wait modes are disabled.

11.6.1 Internal Debug Mode

Internal debug mode provides access to architected processor resources and supports setting hardware and software breakpoints and monitoring processor status. In this mode, debug events generate debug interrupts, 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.

To enable internal debug mode, the Debug Control Register 0 (DBCR0) field IDM is set to 1 (DBCR0[IDM] = 1). To enable debug interrupts, MSR[DE] = 1. A debug interrupt occurs on a debug event only if DBCR0[IDM] = 1 and MSR[DE] = 1.

11.6.2 External Debug Mode

External debug mode provides access to 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 architected processor resources can be accessed and altered. External bus activity continues in external debug mode.

The JTAG mechanism can pass instructions to the processor for execution, allowing a JTAG debugger 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.

Because external debug mode relies only on internal processor resources, it can be used to debug system hardware and software.

In this mode, access to the processor is through the JTAG debug port.

To enable external debug mode, DBCR0[EDM] = 1. To enable debug interrupts, MSR[DE] = 1. A debug interrupt occurs on a debug event only if DBCR0[EDM] = 1 and MSR[DE] = 1.

11.6.3 Debug Wait Mode

In debug wait mode, debug events cause the PPC405CR to enter a state in which interrupts can be serviced while the processor appears to be stopped.

Debug wait mode provides access to architected processor resources in a manner similar to external debug mode, except that debug wait mode allows the servicing of interrupt handlers. It supports stopping, starting, and stepping the processor, setting hardware and software breakpoints, and monitoring processor status. In this mode, if a debug event caused the processor to become architecturally frozen, an interrupt causes the processor to run an interrupt handler and return to the architecturally frozen state upon returning from the interrupt handler. While the processor is frozen, normal instruction execution stops and architected processor resources can be accessed and altered. External bus activity continues in debug wait mode.

The processor enters debug wait mode when internal and external debug modes are disabled (DBCR0[IDM, EDM] = 0), debug wait mode is enabled (MSR[DWE] = 1), debug wait is enabled by the JTAG debugger, and a debug event occurs.

For example, while the PPC405CR is in debug wait mode, an external device might generate an interrupt that requires immediate service. The PPC405CR can service the interrupt (vector to an interrupt handler and execute the interrupt handler code) and return to the previous stopped state.

Debug wait 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.

In this mode, access to the processor is through the JTAG debug port.

11.6.4 Real-time Trace Debug Mode

Real-time trace debug mode supports the generation of trigger events for tracing the instruction stream being executed out of the instruction cache in real-time. In this mode, debug events can be used to control the collection of trace information through the use of trigger event generation. The broadcast of trace information is independent of the use of debug events as trigger events.This mode does not alter the processor performance.

A trace event occurs when internal and external debug modes are disabled (DBCR0[IDM, EDM] = 0) and a debug events occurs.

When a trace event occurs, a trace device can capture trace signals that provide the instruction trace information. Most trace events generated from debug events are blocked when internal debug, external debug, or debug wait modes are enabled

11.7 Processor Control

The PPC405CR provides the following debug functions for processor control. Not all facilities are available in all debug modes.

Instruction Step The processor is stepped one instruction at a time, while stopped, using the JTAG debug port.

Instruction Stuff While the processor is stopped, instructions can be stuffed into the processor and executed using the JTAG debug port.

Halt The 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 architected processor resources can be accessed and altered using the JTAG debug port. Normal execution resumes when the halt signal is deactivated.

Stop The processor can be stopped using the JTAG debug port. Activating a stop causes the processor to become architecturally frozen. While frozen, normal instruction execution stops and the architected processor resources can be accessed and altered using the JTAG debug port.

Reset An external reset signal, the JTAG debug port, or DBCR0 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 “Debug Events” on page 11-16.

Freeze Timers The JTAG debug port or DBCR0 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.

11.8 Processor Status

The processor execution status, exception status, and most recent reset can be monitored.

Execution Status The JTAG debug port can monitor processor execution status to determine whether the processor is stopped, waiting, or running.

Exception Status The JTAG debug port can monitor the status of pending synchronous exceptions.

Most Recent Reset The JTAG debug port or an **mfspr** instruction can be used to read the Debug Status Register (DBSR) to determine the type of the most recent reset.

11.9 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.

11.9.1 Debug Control Registers

The debug control registers (DBCR0 and DBCR1) can enable and configure debug events, reset the processor, control timer operation during debug events, enable debug interrupts, and set the processor debug mode.

11.9.1.1 Debug Control Register 0 (DBCR0)

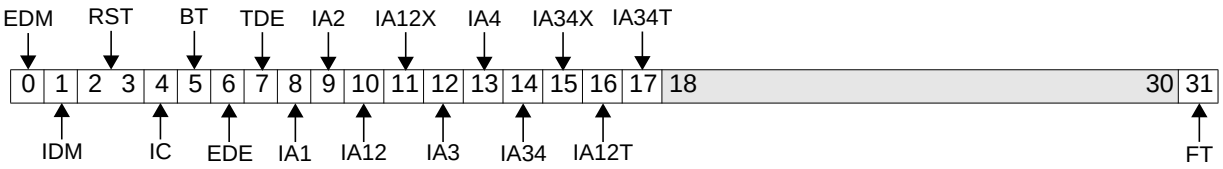


Figure 11-4. Debug Control Register 0 (DBCR0)

0	EDM	External Debug Mode 0 Disabled 1 Enabled
1	IDM	Internal Debug Mode 0 Disabled 1 Enabled
2:3	RST	Reset 00 No action 01 Core reset 10 Chip reset 11 System reset Causes a processor reset request when set by software.
		Attention: Writing 01, 10, or 11 to this field causes a processor reset request.
4	IC	Instruction Completion Debug Event 0 Disabled 1 Enabled
5	BT	Branch Taken Debug Event 0 Disabled 1 Enabled
6	EDE	Exception Debug Event 0 Disabled 1 Enabled
7	TDE	Trap Debug Event 0 Disabled 1 Enabled
8	IA1	IAC 1 Debug Event 0 Disabled 1 Enabled
9	IA2	IAC 2 Debug Event 0 Disabled 1 Enabled

10	IA12	Instruction Address Range Compare 1–2 0 Disabled 1 Enabled	Registers IAC1 and IAC2 define an address range used for IAC address comparisons.
11	IA12X	Enable Instruction Address Exclusive Range Compare 1–2 0 Inclusive 1 Exclusive	Selects the range defined by IAC1 and IAC2 to be inclusive or exclusive.
12	IA3	IAC 3 Debug Event 0 Disabled 1 Enabled	
13	IA4	IAC 4 Debug Event 0 Disabled 1 Enabled	
14	IA34	Instruction Address Range Compare 3–4 0 Disabled 1 Enabled	Registers IAC3 and IAC4 define an address range used for IAC address comparisons.
15	IA34X	Instruction Address Exclusive Range Compare 3–4 0 Inclusive 1 Exclusive	Selects range defined by IAC3 and IAC4 to be inclusive or exclusive.
16	IA12T	Instruction Address Range Compare 1-2 Toggle 0 Disabled 1 Enable	Toggles range 12 inclusive, exclusive DBCR[IA12X] on debug event.
17	IA34T	Instruction Address Range Compare 3–4 Toggle 0 Disabled 1 Enable	Toggles range 34 inclusive, exclusive DBCR[IA34X] on debug event.
18:30		Reserved	
31	FT	Freeze timers on debug event 0 Timers not frozen 1 Timers frozen	

11.9.1.2 Debug Control Register1 (DBCR1)

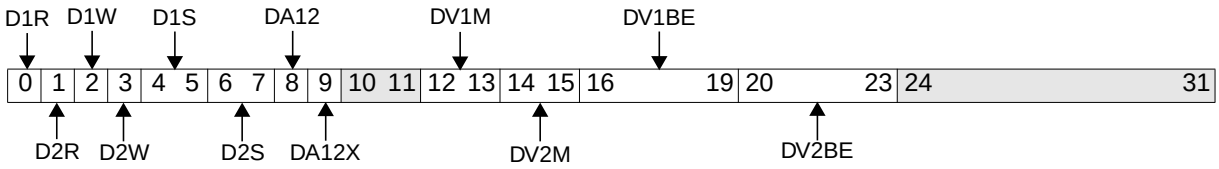


Figure 11-5. Debug Control Register 1 (DBCR1)

0	D1R	DAC1 Read Debug Event 0 Disabled 1 Enabled	
1	D2R	DAC 2 Read Debug Event 0 Disabled 1 Enabled	
2	D1W	DAC 1 Write Debug Event 0 Disabled 1 Enabled	
3	D2W	DAC 2 Write Debug Event 0 Disabled 1 Enabled	
4:5	D1S	DAC 1 Size 00 Compare all bits 01 Ignore lsb (least significant bit) 10 Ignore two lsbs 11 Ignore five lsbs	Address bits used in the compare: Byte address Halfword address Word address Cache line (8-word) address
6:7	D2S	DAC 2 Size 00 Compare all bits 01 Ignore lsb (least significant bit) 10 Ignore two lsbs 11 Ignore five lsbs	Address bits used in the compare: Byte address Halfword address Word address Cache line (8-word) address
8	DA12	Enable Data Address Range Compare 1:2 0 Disabled 1 Enabled	Registers DAC1 and DAC2 define an address range used for DAC address comparisons
9	DA12X	Data Address Exclusive Range Compare 1:2 0 Inclusive 1 Exclusive	Selects range defined by DAC1 and DAC2 to be inclusive or exclusive
10:11		Reserved	

12:13	DV1M	Data Value Compare 1 Mode 00 Undefined 01 AND 10 OR 11 AND-OR	Type of data comparison used: All bytes selected by DBCR1[DV1BE] must compare to the appropriate bytes of DVC1. One of the bytes selected by DBCR1[DV1BE] must compare to the appropriate bytes of DVC1. The upper halfword or lower halfword must compare to the appropriate halfword in DVC1. When performing halfword compares set DBCR1[DV1BE] = 0011, 1100, or 1111.
14:15	DV2M	Data Value Compare 2 Mode 00 Undefined 01 AND 10 OR 11 AND-OR	Type of data comparison used All bytes selected by DBCR1[DV2BE] must compare to the appropriate bytes of DVC2. One of the bytes selected by DBCR1[DV2BE] must compare to the appropriate bytes of DVC2. The upper halfword or lower halfword must compare to the appropriate halfword in DVC2. When performing halfword compares set DBCR1[DV2BE] = 0011, 1100, or 1111.
16:19	DV1BE	Data Value Compare 1 Byte 0 Disabled 1 Enabled	Selects which data bytes to use in data value comparison
20:23	DV2BE	Data Value Compare 2 Byte 0 Disabled 1 Enabled	Selects which data bytes to use in data value comparison
24:31		Reserved	

11.9.2 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 DBSR fields is performed by writing a word 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.

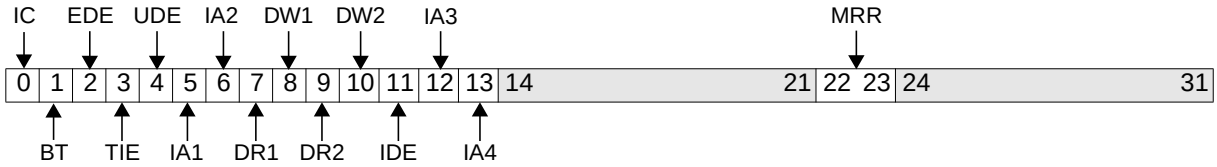


Figure 11-6. Debug Status Register (DBSR)

0	IC	Instruction Completion Debug Event 0 Event did not occur 1 Event occurred
1	BT	Branch Taken Debug Event 0 Event did not occur 1 Event occurred
2	EDE	Exception Debug Event 0 Event did not occur 1 Event occurred
3	TIE	Trap Instruction Debug Event 0 Event did not occur 1 Event occurred
4	UDE	Unconditional Debug Event 0 Event did not occur 1 Event occurred
5	IA1	IAC1 Debug Event 0 Event did not occur 1 Event occurred
6	IA2	IAC2 Debug Event 0 Event did not occur 1 Event occurred
7	DR1	DAC1 Read Debug Event 0 Event did not occur 1 Event occurred
8	DW1	DAC1 Write Debug Event 0 Event did not occur 1 Event occurred
9	DR2	DAC2 Read Debug Event 0 Event did not occur 1 Event occurred
10	DW2	DAC2 Write Debug Event 0 Event did not occur 1 Event occurred

11	IDE	Imprecise Debug Event 0 No circumstance that would cause a debug event (if MSR[DE] = 1) occurred 1 A debug event would have occurred, but debug exceptions were disabled (MSR[DE] = 1)	
12	IA3	IAC3 Debug Event 0 Event did not occur 1 Event occurred	
13	IA4	IAC4 Debug Event 0 Event did not occur 1 Event occurred	
14:21		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	

11.9.3 Instruction Address Compare Registers (IAC1–IAC4)

The PPC405CR can take a debug event upon an attempt to execute an instruction from an address. The address, which must be word-aligned, is defined in an IAC register. The DBCR0[IA1, IA2] fields of DBCR0 controls the instruction address compare (IAC) debug event.

Figure 11-7. Instruction Address Compare Registers (IAC1–IAC4)

0:29		Instruction Address Compare word address	Omit two low-order bits of complete address.
30:31		Reserved	

11.9.4 Data Address Compare Registers (DAC1–DAC2)

The PPC405CR can take a debug event upon storage or cache references to addresses specified in the DAC registers. The specified addresses in the DAC registers are EAs of operands of storage

references or cache instructions.The fields DBCR1[D1R], [D2R] and DBCR[D1W], [D2W] control the DAC-read and DAC-write debug events, respectively.

Addresses in the DAC registers specify exact byte EAs for DAC debug events. However, one may want to take a debug event on any byte within a halfword (ignore the least significant bit (LSb) of the DAC), on any byte within a word (ignore the two LSbs of DAC), or on any byte within eight words (ignore four LSbs of DAC). DBCR1[D1S, D2S] control the addressing options.

Errors related to execution of storage reference or cache instructions prevent DAC debug events.

Figure 11-8. Data Address Compare Registers (DAC1–DAC2)

0:31		Data Address Compare (DAC) byte address	DBCR0[D1S] determines which address bits are examined.

11.9.5 Data Value Compare Registers (DVC1–DVC2)

The PPC405CR can take a debug event upon storage or cache references to addresses specified in the DAC registers, that also require the data at that address to match the value specified in the DVC registers. The data address compare for a DVC events works the same as for a DAC event. Cache operations do not cause DVC events. If the data at the address specified matches the value in the corresponding DVC register a DVC event will occur. The fields DBCR1[DV1M, DV2M] control how the data value are compared.

Errors related to execution of storage reference or cache instructions prevent DVC debug events.

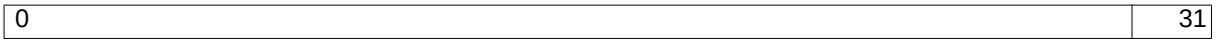


Figure 11-9. Data Value Compare Registers (DVC1–DVC2)

0:31		Data Value to Compare

11.9.6 Debug Events

Debug events, enabled and configured by DBCR0 and DBCR1 and recorded in the DBSR, cause debug operations. A debug event occurs when an event listed in Table 11-4, “Debug Events,” on page 11-16 is detected. The debug operation is performed after the debug event.

In internal debug mode, the processor generates a debug interrupt 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 stops on a debug event with the debug interrupt pending. When external and internal debug mode are both disabled, and debug wait mode is enabled the processor stops, but can be restarted by an interrupt. When all debug modes are disabled, debug events are recorded in the DBSR, but no action is taken.

Table 11-4 lists the debug events and the related fields in DBCR0, DBCR1, and DBSR. DBCR0 and DBCR1 enable the debugs events, and the DBSR fields report their occurrence.

Table 11-4. Debug Events

Event	Enabling DBCR0, DBCR1 Fields	Reporting DBSR Fields	Description
Instruction Completion	IC	IC	Occurs after completion of an instruction.
Branch Taken	BT	BT	Occurs before execution of a branch instruction determined to be taken.
Exception Taken	EDE	EXC	Occurs after an exception.
Trap Instruction	TDE	TIE	Occurs before execution of a trap instruction where the conditions are such that the trap will occur.
Unconditional	UDE	UDE	Occurs immediately upon being set by the JTAG debug port.
Instruction Address Compare	IA1, IA2, IA3, IA4, IA12, IA12X, IA12T, IA34, IA34X, IA34T	IA1, IA2, IA3, IA4	Occurs before execution of an instruction at an address that matches an address defined by the Instruction Address Compare Registers (IAC1–IAC4).
Data Address Compare	D1R, D1W, D1S, D2R, D2W, D2S, DA12, DA12X	DR2,DW2	Occurs before execution of an instruction that accesses a data address that matches the contents of the specified DAC register.
Data Value Compare	DV1M, DV2M, DV1BE, DV2BE	DR1, DW1	Occurs after execution of an instruction that accesses a data address for which a DAC occurs, and for which the value at the address matches the value in the specified DVC register.
Imprecise		IDE	Indicates that another debug event occurred while MSR[DE] = 0

11.9.7 Instruction Complete Debug Event

This debug event occurs after the completion of an instruction. If DBCR0[IDM] = 1, DBCR0[EDM] = 0 and MSR[DE] =0 this debug event is disabled.

11.9.8 Branch Taken Debug Event

This debug event occurs before execution of a branch instruction determined to be taken. If DBCR0[IDM] = 1, DBCR0[EDM] = 0 and MSR[DE] =0 this debug event is disabled.

11.9.9 Exception Taken Debug Event

This debug event occurs after an exception. Exception debug events always include the non-critical class of exceptions. When DBCR0[IDM] = 1 and DBCR0[EDM] = 0 the critical exceptions are not included.

11.9.10 Trap Taken Debug Event

This debug event occurs before execution of a trap instruction where the conditions are such that the trap will occur. When trap is enabled for a debug event, external debug mode is enabled, internal debug mode is enabled with MSR[DE] enabled, or debug wait mode is enabled, a trap instruction will not cause a program exception.

11.9.11 Unconditional Debug Event

This debug event occurs immediately upon being set by the JTAG debug port.

11.9.12 IAC Debug Event

This debug event occurs before execution of an instruction at an address that matches an address defined by the Instruction Address Compare Registers (IAC1–IAC4). DBCR0[IA1, IA2, IA3, IA4] enable IAC debug events IAC can be defined as an exact address comparison to one of the IAC*n* registers or on a range of addresses to compare defined by a pair of IAC*n* registers.

11.9.12.1 IAC Exact Address Compare

In this mode each IAC*n* register specifies an exact address to compare. These are enabled by setting DBCR0[IA*n*] = 1 and disabling IAC range compare (DBCR0[IA12X] = 0 for IAC1 and IAC2 and DBCR0[IA23X] = 0 for IAC3 and IAC4). The corresponding DBSR[IA*n*] bit displays the results of the debug event.

11.9.12.2 IAC Range Address Compare

In this mode a pair of IAC*n* registers are used to define a range of addresses to compare:

Range 1:2 corresponds to IAC1 and IAC2
Range 3:4 corresponds to IAC3 and IAC4

To enable Range 1:2, DBCR0[IA12] = 1 and DBCR0[IA1] or DBCR0[IA2] =1. An IAC event will be seen on the DBSR[IA*n*] field that corresponds to the enabled DBCR0[IA*n*] field. If DBCR0[IA1] and DBCR0[IA2] are enabled, the results of the event are reported on both DBSR fields. Setting DBCR0[IA12] =1 prohibits IAC1 and IAC2 from being used for exact address compares.

To enable Range 3:4, DBCR0[IA34] = 1 and DBCR0[IA3] or DBCR0[IA4] =1. An IAC event will be seen on the DBSR[IA*n*] field that corresponds to the enabled DBCR0[IA*n*] field. If DBCR0[IA3] and DBCR0[IA4] are enabled, the results of the event will be reported on both DBSR fields. Setting DBCR0[IA34] =1 prohibits IAC3 and IAC4 from being used for exact address compares.

Ranges can be defined as inclusive, as shown in the preceding examples, or exclusive, using DBCR0[IA12X] (corresponding to range 1:2) and DBCR0[IA34X] (corresponding to range 3:4), as follows:

DBCR0[IA12] = 1: Range 1:2 = IAC1 ≤ range < IAC2.
DBCR0[IA12X] = 1: Range 1:2 = Range low < IAC1 or IAC2 ≤ Range high
DBCR0[IA34] = 1: Range 3:4 = IAC3 ≤ range < IAC4.
DBCR0[IA34X] = 1: Range 3:4 = Range low < IAC3 or IAC4 ≤ Range high

Figure 11-10 shows the range selected in an inclusive IAC range address compare. Note that the address in IAC1 is considered part of the range, but the address in IAC2 is not, as shown in the preceding examples. The thick lines indicate that the indicated address is included in the compare results.

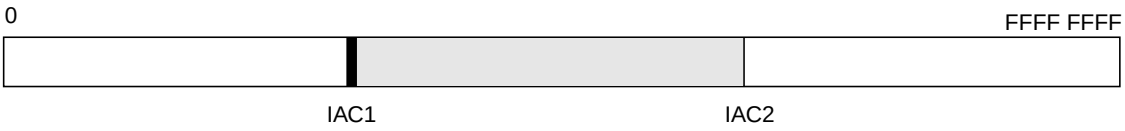


Figure 11-10. Inclusive IAC Range Address Compares

Figure 11-11 shows the range selected in an inclusive IAC range address compare. Note that the address in IAC1 is not considered part of the range, but the address in IAC2 is, along with the highest memory address, as shown in the preceding examples.

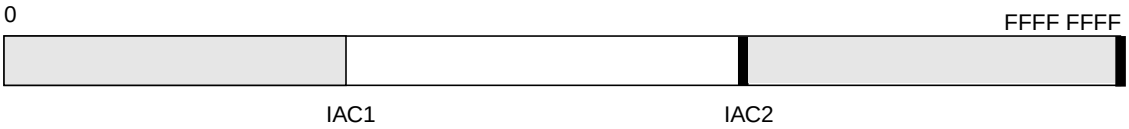


Figure 11-11. Exclusive IAC Range Address Compares

To toggle the range from inclusive to exclusive or from exclusive to inclusive on a IAC range debug event, DBCR0[IA12T] (corresponding to range 1:2) and DBCR0[IA34T] (corresponding to range 3:4) are used. If these fields are set, the DBCR0[IA12X] or DBCR0[IA34X] fields toggle on an IAC debug event, changing the defined range.

When a toggle is enabled (DBCR0[IA12T] for range 1:2 or DBCR0[IA34T] = 1 for range 3:4), and DBCR0[IDM] =1, DBCR0[EDM] = 0, and MSR[DE] = 0, IAC range comparisons for the corresponding toggle field are disabled.

11.9.13 DAC Debug Event

This debug event occurs before execution of an instruction that accesses a data address that matches the contents of the specified DAC register. DBCR1[D1R, D2R, D1W, D2W] enable DAC debug events for address comparisons on DAC1 and DAC2 for read instructions, DAC2 for read instructions, DAC1 for write instructions, DAC2 for write instructions respectively. Loads are reads and stores are writes. DAC can be defined(DBCR1[D1R, D2R])as an exact address comparison to one of the DACn registers or a range of addresses to compare defined by DAC1 and DAC2 registers.

11.9.13.1 DAC Exact Address Compare

In this mode, each DAC*n* register specifies an exact address to compare. Thes registers are enabled by setting one or more of DBCR1[D1R,D2R,D1W,D2W] = 1, and disabling DAC range compare DBCR1[DA12X] = 0. The corresponding DBSR[DR1,DR2,DW1,DW2] field displays the results of a DAC debug event.

The address for a DAC is the effective address (EA) of a storage reference instruction. EAs are always generated within a single aligned word of memory. Unaligned load and store, strings, and multiples generate multiple EAs to be used in DAC comparisons.

Data address compare (DAC) debug events can be set to react to any byte in a larger block of memory, in addition to reacting to a byte address match. The DAC Compare Size fields (DBCR1[D1S, D2S]) allow DAC debug events to react to byte, halfword, word, or 8-word line address by ignoring a number of LSBs in the EA.

DAC 1 Size	
00 Compare all bits	Byte address
01 Ignore LSB (least significant bit)	Halfword address
10 Ignore two LSBs	Word address
11 Ignore five LSBs	Cache line (8-word) address

The user must determine how the addresses of interest are accessed, relative to byte, halfword, word, string, and unaligned storage instructions, and adjust the DAC compare size field appropriately to cover the addresses of interest.

For example, suppose that a DAC debug event should react to byte 3 of a word-aligned target. A DAC set for exact compare would not recognize a reference to that byte by load/store word or load/store halfword 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 word operations to the misaligned byte are to be detected). The wider capture range may result 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.

While load/store string instructions are inherently byte addressed the processor will generate EAs containing the largest portion of an aligned word address as possible. It may not be possible to DAC on a specific individual byte using load/store string instructions.

11.9.13.2 DAC Range Address Compare

In this mode, the pair of DAC1 and DAC2 registers are used to define a range of addresses to compare.

To enable DAC range, DBCR1[DA12] = 1 and one or more of DBCR1[D1R,D2R,D1W,D2W] =1. The DAC event is seen on the DBSR[DR1,DR2,DW1,DW2] field that corresponds to the DBCR1[D1R,D2R,D1W,D2W] field that is enabled. For example, if DBCR1[D1R] and DBCR1[D2R] are enabled, the results of a DAC debug event are reported on DBSR[DR1, DR2]. Setting DBCR1[DA12] =1 prohibits DAC1 and DAC2 from being used for exact address compares.

Ranges are defined to be inclusive or exclusive, using the DBCR1[DA12X], as follows:

DBCR1[DA12] = 1: Range = DAC1 ≤ range < DAC2.
DBCR1[DA12X] = 1: Range = Range low < DAC1 or DAC2 ≤ Range high.

Figure 11-12 shows the range selected in an inclusive DAC range address compare. Note that the address in DAC1 is considered part of the range, but the address in DAC2 is not, as shown in the preceding examples. The thick lines indicate that the indicated address is included in the compare results.

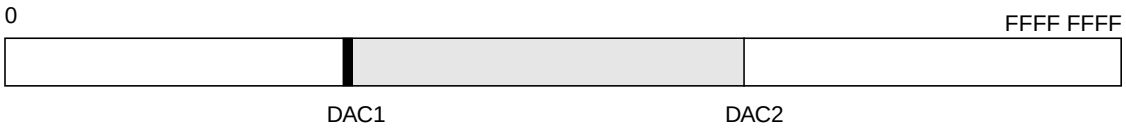


Figure 11-12. Inclusive DAC Range Address Compares

Figure 11-13 shows the range selected in an exclusive DAC range address compare. Note that the address in DAC1 is not considered part of the range, but the address in DAC2 is, along with the highest memory address, as shown in the preceding examples.

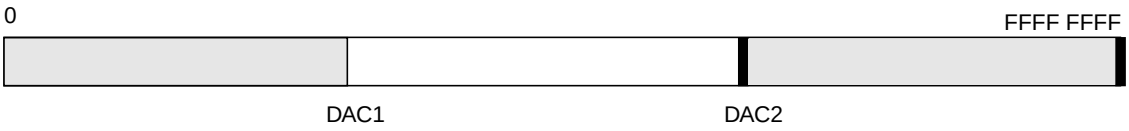


Figure 11-13. Exclusive DAC Range Address Compares

The DAC Compare Size fields (DBCR1[D1S, D2S]) are not used by DAC range comparisons.

11.9.13.3 DAC Applied to Cache Instructions

Some cache instructions can cause DAC debug events. There are several special cases.

Table 11-5 summarizes possible DAC debug events by cache instruction:

Table 11-5. DAC Applied to Cache Instructions

Instruction	Possible DAC Debug Event	
	DAC-Read	**DAC-Write**
dcba	No	Yes
dcbf	No	Yes
dcbi	No	Yes
dcbst	No	Yes
dcbt	Yes	No
dcbz	No	Yes
dccci	No	No
dcread	No	No
dcbtst	Yes	No
icbi	Yes	No

Table 11-5. DAC Applied to Cache Instructions (continued)

Instruction	Possible DAC Debug Event	
	DAC-Read	DAC-Write
icbt	Yes	No
iccci	No	No
icread	No	No

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 can also 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.

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 regardless of cachability.

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.

Even though **dcread** and **dccci** are not address-specific (they affect a congruence class regardless of the instruction operand address), and are considered “loads,” in the PPC405CR 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 in the PPC405CR.

11.9.13.4 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 operand 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.

11.9.14 Data Value Compare Debug Event

A data value compare (DVC) debug event can occur only after execution of a load or store instruction to an address that compares with the address in one of the DAC*n* registers and has a data value that matches the corresponding DVC*n* register. Therefore, a DVC debug event requires both the data address comparison and the data value comparison to be true. A DVCn debug event when enabled in the DBCR1 supercedes a DACn debug event since the DVCn and the DACn both use the same DACn register.

DVC1 debug events are enabled by setting the appropriate DAC enable DBCR1[D1R,D1W] to cause an address comparison and by setting anybit combination in the DBCR1[DV1BE]. DVC2 debug

events are enabled by setting the appropriate DAC enable DBCR1[D2R,D2W] to cause an address comparison and by setting any bit combination in the DBCR1[DV1BE]. Each bit in DBCR1[DV1BE, DV2BE] correspondes to a byte in DVC1 and DVC2. Exact address compare and range address compare work the same for DVC as for a simple DAC.

DBSR[DR1] and DBSR[DW1] record status for DAC1 debug events. Which DBSR bit is set depends on the setting of DBCR1[D1R] and DBCR[D1W]. If DBCR1[D1R] = 1, DBSR[DR1] = 1, assuming that a DVC event occurred. Similarly, if DBCR1[D1W] = 1, DBSR[DW1] = 1, assuming that a DVC event occurred.

Similarly, DBSR[DR2] and DBSR[DW2] record status for DAC2 debug events. Which DBSR bit is set depends on the setting of DBCR1[D2R] and DBCR[D2W]. If DBCR1[D2R] = 1, DBSR[DR2] = 1, assuming that a DVC event occurred. Similarly, if DBCR1[D2W] = 1, DBSR[DW2] = 1, assuming that a DVC event occurred.

In the following example, a DVC1 event is enabled by setting DBCR1[D1R] = 1, DBCR1[D1W] = 1, DBCR1[DA12] = 0, and DBCR1[DV1BE] = 0000. When the data address and data value match the DAC1 and DVC1, a DVC1 event is recorded in DBSR[DR1] or DBSR[DW1], depending on whether the operation is a load (read) or a store (write). This example corresponds to the last line of Table 11-6.

In Table 11-6, *n* is 1 or 2, depending on whether the bits apply to DAC1, DAC2, DVC1, and DVC2 events. “Hold” indicates that the DBSR holds its value unless cleared by software. “RA” indicates that the operation is a read (load) and the data address compares (exact or range). “WA” indicates that the operation is a write (store) and the data address compares (exact or range). “RV” indicates that the operation is a read (load), the data address compares (exact or range), and the data value compares according to DBCR1[DVC*n*].

Table 11-6. Setting of DBSR Bits for DAC and DVC Events

			DBCR1			DBSR	
DACn Event	**DVCn Enabled**	**DVCn Event**	**[DnR]**	**[DnW]**	**[DA12]**	**[DRn]**	**[DWn]**
0	—	—	—	—	—	Hold	Hold
—	—	—	0	0	—	Hold	Hold
1	0	—	0	1	—	Hold	WA
1	0	—	1	0	—	RA	Hold
1	0	—	1	1	—	RA	WA
1	1	0	—	—	—	Hold	Hold
1	1	1	0	1	—	Hold	WV
1	1	1	1	0	—	RV	Hold
1	1	1	1	1	—	RV	WV

The settings of DBCR1[DV1M] and DBCR1[DV2M] are more precisely defined in Table 11-8 and Table 11-9. (*n* enables the table to apply to DBCR1[DV1M, DV2M] and DBCR1[DV1BE, DV2BE]). DV*n*BE$_m$ indicates bytes selected (or not selected) for comparison in DBCR1[DV*n*BE].

When DBCR1[DV*n*M] = 01, the comparison is an AND; all bytes must compare to the appropriate bytes of DVC1.

When DBCR1[DV*n*M] = 10, the comparison is an OR; at least one of the selected bytes must compare to the appropriate bytes of DVC1.

When DBCR1[DV*n*M] = 11, the comparison is an AND-OR (halfword) comparison. This is intended for use when DBCR1[DV*n*BE] is set to 0011, 0111, or 1111. Other values of DBCR1[DV*n*BE] can be compared, but the results are more easily understood using the AND and OR comparisons. In Table 11-7, "not" is $\neg$, AND is $\wedge$, and OR is $\vee$.

Table 11-7. Comparisons Based on DBCR1[DVnM]

DBCR1[DVnM] Setting	Operation	Comparison
00	—	Undefined
01	AND	$(\neg DV_nBE_0 \vee (DVC1[byte\ 0] = data[byte\ 0])) \wedge$ $(\neg DV_nBE_1 \vee (DVC1[byte\ 1] = data[byte\ 1])) \wedge$ $(\neg DV_nBE_2 \vee (DVC1[byte\ 2] = data[byte\ 2])) \wedge$ $(\neg DV_nBE_3 \vee (DVC1[byte\ 3] = data[byte\ 3]))$
10	OR	$(DV_nBE_0 \wedge (DVC1[byte\ 0] = data[byte\ 0])) \vee$ $(DV_nBE_1 \wedge (DVC1[byte\ 1] = data[byte\ 1])) \vee$ $(DV_nBE_2 \wedge (DVC1[byte\ 2] = data[byte\ 2])) \vee$ $(DV_nBE_3 \wedge (DVC1[byte\ 3] = data[byte\ 3]))$
11	AND-OR	$(DV_nBE_0 \wedge (DVC1[byte\ 0] = data[byte\ 0])) \wedge$ $(DV_nBE_1 \wedge (DVC1[byte\ 1] = data[byte\ 1])) \vee$ $(DV_nBE_2 \wedge (DVC1[byte\ 2] = data[byte\ 2])) \wedge$ $(DV_nBE_3 \wedge (DVC1[byte\ 3] = data[byte\ 1]))$

Table 11-8 illustrates comparisons for aligned DVC accesses, that is, words, halfwords, or bytes on naturally aligned boundaries (all byte accesses are aligned).

Table 11-8. Comparisons for Aligned DVC Accesses

Access	DBCR1[DVnBE] Setting	Value	Operation
Word	All	Word value	AND
Halfword (Low-Order)	All	Halfword value replicated	AND-OR
Halfword (High-Order)	All	Halfword value replicated	AND-OR
Byte	All	Byte value replicated	OR

For halfword accesses, the halfword value is replicated in the "empty " halfword in the DVC register, for example, if the low-order halfword is to be compared, its value is stored in the low-order halfword and the high-order halfword of the register. Similarly, a byte value is replicated in each byte in the register.

Table 11-9 illustrates comparisons for misaligned DVC accesses. In the “DVC1” and “DVC2” columns, “x” indicates a don’t care.

Table 11-9. Comparisons for Misaligned DVC Accesses

Access	Operation	DVC1 (Hex)	DVC2 (Hex)	DBCR1[DV1BE] Setting	DBCR1[DV2BE] Setting	DBCR1[D2S] Setting
Word (Offset 1)	AND	xx112233	44xxxxxx	123	0	01
Word (Offset 2)	AND	xxxx1122	3344xxxx	23	01	10
Word (Offset 3)	AND	xxxxxx11	223344xx	3	012	10
Halfword (Offset 1)	AND	xx1122xx		12	12	10
Halfword (Offset 3)	AND	xxxxxx11	22xxxxxx	3	0	10

Note: Misaligned accesses stop the processor on the instruction causing the compare hit. The second part of an instruction is not performed if the first part of the compare hits.

11.9.15 Imprecise Debug Event

The imprecise debug event is not an independent debug event, but indicates that a debug event occurred while MSR[DE] = 0. This is useful in internal debug mode if a debug event occurs while in a critical interrupt handler. On return from interrupt, a debug interrupt occurs if MSR[DE] = 1. If DBSR[IDE] = 1, the debug event causing the interrupt occurred sometime earlier, not immediately after a debug event.

Chapter 12. Clock and Power Management

The PPC405CR provides a clock and power management (CPM) controller that reduces power dissipation by stopping clocks in unused or dormant functional units. Use of the CPM controller requires careful programming and special consideration to avoid compromising system and functional unit integrity.

The CPM controller supports three different types of sleep interfaces to the functional units:

- In a CPM class 1 interface, the CPM_Sleep_N signal is asserted by the CPM controller when a register bit is set by software. The functional unit is unconditionally put to sleep. There is no other communication with the functional unit.
- In a CPM class 2 interface, the functional unit uses a combination of its internal state and external inputs to determine whether or not it can be put to sleep. If sleeping is permitted, the functional unit asserts the Sleep_Req signal to the CPM controller that responds by asserting CPM_Sleep_N if the enable for that unit is set. The CPM_Sleep_N signal to a class 2 unit is deasserted when the CPM controller enable bit for that unit is reset, or when the unit deasserts its Sleep_Req signal.
- The CPM class 3 interface has a CPM_SleepInit signal that is asserted by the CPM controller to request that a functional unit go to sleep. If the unit can sleep, it asserts the Sleep_Req signal to the CPM controller. The CPM_Sleep_N signal is then asserted by the CPM controller to shut off the class 3 clocks in the functional unit. The functional unit or the CPM controller can end the sleep state. If the CPM controller enable bit for the unit is reset, the CPM controller immediately deasserts CPM_SleepInit and CPM_Sleep_N.

12.1 CPM Registers

Table 12-1 lists the registers used to program the CPM controller.

Table 12-1. CPM Registers

Register Name	DCR Address	Access	Reset Value (0:16) (bits 17:31 reserved)
CPC0_ER	0x0B9	Read/Write	0x0000xxxx
CPC0_FR	0x0BA	Read/Write	0x0000xxxx
CPC0_SR	0x0B8	Read Only	0xFFFFxxxx

Each functional unit has one bit in each of CPC0_ER, CPC0_FR, and CPC0_SR. The bit assignment is the same in the three registers. Figure 12-1 on page 12-2 shows the bit assignment and CPM class for each PPC405CR functional units.

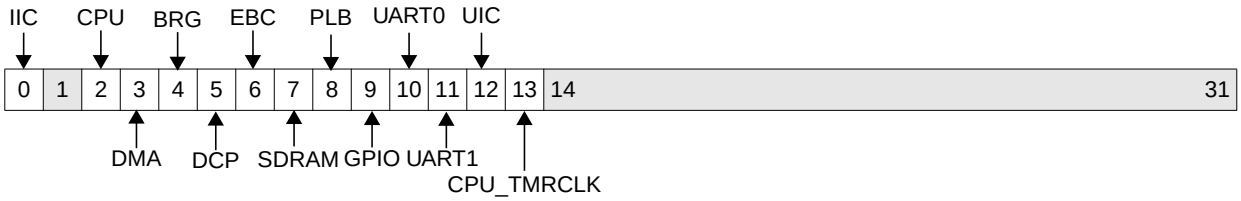


Figure 12-1. CPM Registers (CPC0_ER, CPC0_FR, CPC0_SR)

0	IIC	IIC Interface	Class 3
1		Reserved	
2	CPU	PPC405CR Processor Core	Class 2
3	DMA	DMA Controller	Class 2
4	BRG	PLB to OPB Bridge	Class 2
5	DCP	CodePack	Class 2
6	EBC	ROM/SRAM Peripheral Controller	Class 2
7	SDRAM	SDRAM Memory Controller	Class 2
8	PLB	PLB Bus Arbiter	Class 2
9	GPIO	General Purpose Interrupt Controller	Class 1
10	UART0	Serial Port 0	Class 1
11	UART1	Serial Port 1	Class 1
12	UIC	Universal Interrupt Controller	Class 1
13	CPU_TMRCLK	CPU Timers	Class 1
14:31		Reserved	

12.1.1 CPM Enable Register (CPC0_ER)

The CPC0_ER bits enable the process of putting a functional unit to sleep. The class of a unit determines how its interface signals are controlled when the bit associated with the unit is set to 1.

Class 1 When an associated CPC0_ER bit is set to 1, the CPM_Sleep_N signal to the class 1 unit is asserted. When the bit is set to 0, CPM_Sleep_N is deasserted. There are some additional considerations to avoid generating extraneous interrupts when waking the UIC. Before enabling sleep mode (setting DPC0_ER[UIC] to 1), save the contents of the UIC Masked Status Register (UIC_MSR) and UIC Enable Register (UIC0_ER), and disable all interrupts by setting UIC0_ER to 0. After exiting sleep mode, write the ones complement of the saved contents of the UIC_MSR to the UIC Status Register (UIC0_SR), and restore the state of the UIC0_ER.

Class 2 When an associated CPC0_ER bit is set to 1, and the Sleep_Req signal from the class 2 unit is asserted (the unit is requesting sleep state), CPM_Sleep_N to the class 2 unit is asserted. When the bit is set to 0, the CPM_Sleep_N signal is deasserted.

Class 3 When an associated CPC0_ER bit is set to 1, the CPM_SleepInit signal to the class 3 unit is asserted (the CPM controller is requesting permission to put the unit to sleep). When the class 3 unit activating the Sleep_Req in response, (the unit is giving permission to be put to sleep), CPM_Sleep_N signal to the class 3 unit is asserted. When the bit is set to 0, CPM_SleepInit and CPM_Sleep_N are deasserted.

12.1.2 CPM Force Register (CPC0_FR)

Setting a CPC0_FR bit forces assertion of the CPM_Sleep_N signal to the functional unit. For a class 1 unit, this is equivalent to setting the CPC0_ER bit associated with the unit. For class 2 or class 3 units, CPM_Sleep_N is asserted regardless of the state of the Sleep_Req signal coming from the unit.

12.1.3 CPM Status Register (CPC0_SR)

The read-only CPC0_SR shows the current state of all CPM_Sleep_N signals. If the bit is set, the signal is asserted.

Chapter 13. Decompression Controller Operation

The decompression controller enables PPC405CR instructions to be stored in memory in a compressed form, thus, reducing memory requirements and overall system cost. Because the processor core cannot execute compressed instructions, hardware is provided to decompress the compressed instructions. The decompression controller decompresses instructions on-the-fly, as the processor core fetches instructions from memory. The decompression unit is located in the path between memory and the processor core.

When decompressed instructions are stored in the instruction cache unit (ICU), the performance penalty is minimized. Increased latency occurs only when instruction fetches miss in the cache. Average latency is further reduced by a buffer in the decompression hardware that holds more instructions than are requested by the CPU. If a subsequent fetch occurs to a nearby memory location that is already in the buffer, data is returned from the buffer; no memory cycle is required. In such cases, initial latency is less than a normal uncompressed fetch from memory.

Code to be stored in compressed form is compressed by software after compiling and before loading. Thus, code compression can be performed once by the application developer. The compressed code is in the load image.

The decompression controller, located between the processor core and memory, is transparent to the processor core. When the processor core performs an instruction fetch, the decompression controller works with the memory controller (SDRAM or external bus controller [EBC]) to provide a decompressed instruction stream to the CPU. The memory controllers deliver the compressed stream of instructions to the decompression controller, which decompresses the instructions and delivers the resulting instruction stream to the processor core.

13.1 Code Compression and Decompression

While code decompression is performed in hardware when the processor core fetches instructions in a compressed code address space, code compression is performed only once, between the compile and load steps in the usual programming sequence.

13.1.1 Code Compression

Compression software analyzes compiled code, finds the set of most frequently used instructions, and builds decode tables from those instructions.

The compression software splits each instruction, analyzes the upper and lower halfwords of the instruction to find the most common 16-bit patterns. The most frequently used patterns are tagged for insertion in a decode table. Infrequently used instructions are tagged as not compressed. The 16-bit patterns that match those in the decode table are stored in the compressed code image, along with uncompressed instructions. The decode table is loaded into the decompression controller before attempted execution of the compressed instruction streams.

After the decode table is built, a compiled code image is compressed, using a scheme employing a tag and an index. The decompression controller generates an address into the decode table to select the original 16-bit pattern.

The compression software stores compressed and uncompressed instructions in 64-byte blocks, called compression blocks. The number of bytes used to store compressed code varies in each compression block, depending on the encoding. Because the length resulting compressed block varies, the compression software generates an index table that translates addresses of uncompressed 64-byte blocks, called target instruction addresses (TIAs), to addresses of the compressed variable length blocks. The index table also contains information indicating, for each block, whether the compression block was compressed or left uncompressed. The index table is loaded into system memory with the compressed blocks of program code. The decompression controller accesses the index table when a fetch is made to compressed program space.

13.1.2 Code Decompression

The decompression controller is located between the processor core and the SDRAM and EBC memory controllers, isolating the memory controllers from the processor local bus (PLB). The decompression controller is transparent to the processor core and the memory controllers. The processor core side of the decompression controller is a PLB slave to the PLB bus, and the memory controller side is a PLB master to the memory controllers. Instruction-side requests for external memory go though the decompression controller, and are either intercepted for decompression or allowed to pass through unaltered.

The decompression controller must distinguish between compressed and uncompressed code space. To accomplish this, the processor core provides a signal with each instruction memory reference, indicating whether the reference is in compressed or uncompressed memory space. The CPU's U0 storage attribute is used to implement this signal. When address translation is enabled (the IR field of the Machine State Register (MSR) is 1), a corresponding U0 bit is implemented in translation lookaside buffer (TLB) entries in the memory management unit (MMU). When address translation is disabled (MSR[IR] = 0, the Storage User-defined 0 Register (SU0R) implements the U0 storage attribute for memory regions. The U0 storage attribute indicates whether a page is marked as compressed.

13.1.3 Instruction Fetches to Compressed Pages

The decompression controller intercepts instruction fetches to code in uncompressed pages, locates the compressed code in memory, and decompresses the instruction stream before forwarding it to the processor core. These steps are detailed in the following procedure:

1. Locate compressed code
   a. Calculate offset into index table.
   b. Retrieve index from index table.
   c. Calculate address of compression block using the index table entry.
   d. Retrieve compression block.
2. Decode instructions into the output buffer, if required.
3. Return instructions to the processor core.

The decompression controller contains a 64B output buffer that stores instruction words as they are decompressed. Because code is compressed and decompressed in 64B blocks and the CPU requests smaller line fills (32 bytes), some accesses to compressed instructions may already be available in the decompression controller output buffer, decompressed and ready for execution. In this case, line fill requests are serviced from the output buffer, without having to address external memory.

Step 2, above, indicates that compressed code is decoded “if required.” Some compression blocks do not contain compressed code, even though they reside in compressed code space. The compression algorithm can infrequently result in blocks that are larger than the original code. In such cases, the compression software does not compress the instruction block and marks it as uncompressed. The decoder will not attempt to decompress such blocks.

13.1.4 Instruction Fetches to Uncompressed Pages

The decompression controller does not intercept instruction fetches to code in uncompressed memory space. These fetches are passed, unaltered, to the appropriate memory controller. The decompression controller also provides unaltered data, from the appropriate memory controller, back to the CPU in the case of a read. This transaction does not incur any additional latency.

13.1.5 Performance

The decompression controller is designed so that read/write requests to uncompressed memory spaces incur no additional latency over a system without a decompression controller.

The decompression controller adds additional latency to some read accesses to compressed memory space. The decompression controller requires additional cycles to locate compressed code, fetch it, decode the instructions, and place them in the output buffer. However, because instructions are encoded in 64-byte compression blocks, the decompression controller continues to decompress instructions until its 64-byte output buffer is full. In such cases, the decompression controller acts as a prefetch buffer, and subsequent fetches to instructions in the same 64-byte block actually incur less latency than if the decompression controller were not present.

More significant latencies occur when the processor core executes a branch out of the current 64-byte block. If the branch is to a TIA that is not at the beginning of a 64-byte block, the decompression controller must fetch and decode the block containing the TIA, byte by byte, until it reaches the desired instruction. The worst latencies occur when the TIA is outside of the 128-byte group, since the decompression core must then go through its entire decode process, including a refetch of the index table entry.

13.2 Decompression Controller Registers

The registers that control operation of the decompression controller, listed in Table 13-1, are indirectly accessed using the Decompression Controller Address Register (DCP0_CFGADDR) and Decompression Controller Data Register (DCP0_CFGDATA) listed in Table 13-2. The DCP0_CFGADDR and DCP0_CFGDATA are accessed using the **mfdcr** and **mtdcr** instructions.

Descriptions of the registers follow Table 13-2.

The DCP0_CFGADDR stores the address of the target register, while the DCP0_CFGDATA register is used to write or read the target register contents.

The following procedure accesses the decompression controller registers.

1. Write the offset from Table 13-2 to the Decompression Controller Address Register (DCP0_CFGADDR).
2. Read data from or write data to the Decompression Controller Data Register (DCP0_CFGDATA).

Table 13-1. DCRs Used to Access the Decompression Controller Registers

Register	DCR Address	Access	Description
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register

Table 13-2. Offsets for Decompression Controller Registers

Register	Offset	R/W	Description
DCP0_ITOR0	0x00	R/W	Index Table Origin Register 0
DCP0_ITOR1	0x01	R/W	Index Table Origin Register 1
DCP0_ITOR2	0x02	R/W	Index Table Origin Register 2
DCP0_ITOR3	0x03	R/W	Index Table Origin Register 3
DCP0_ADDR0	0x04	R/W	Address Decode Definition Register 0
DCP0_ADDR1	0x05	R/W	Address Decode Definition Register 1
DCP0_CFG	0x40	R/W	Decompression Controller Configuration Register
DCP0_ID	0x41	R	Decompression Controller ID Register
DCP0_VER	0x42	R	Decompression Controller Version Register
DCP0_PLBBEAR	0x50	R	Bus Error Address Register (PLB)
DCP0_MEMBEAR	0x51	R	Bus Error Address Register (EBC/SDRAM)
DCP0_ESR	0x52	R/Clear	Bus Error Status Register 0 (Masters 0–3)
DCP0_RAM0-DCP0_RAM3FF	0x400–0x7FF	R/W	SRAM/ROM Read/Write

13.2.1 Index Table Origin Registers (DCP0_ITOR0–DCP0_ITOR3)

DCP0_ITOR0–DCP0_ITOR3 each contain the high-order address bits of the index table for a compression region.

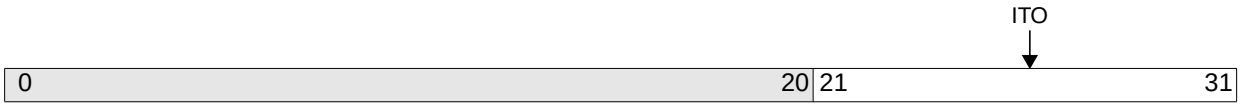
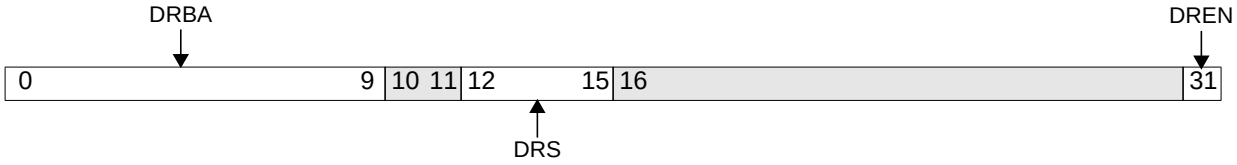


Figure 13-1. Decompression Index Table Origin Registers (DCP0_ITOR0–DCP0_ITOR3)

0:20		Reserved	
21:31	ITO	Index Table Origin	High-order index table address bits

13.2.2 Decompression Address Decode Definition Registers (DCP0_ADDR0–DCP0_ADDR1)

DCP0_ADDR0–DCP0_ADDR1 each define an address space used by the decompression controller for compressed instructions, and for uncompressed accesses that are passed to a memory controller.



Address Decode Definition Registers (DCP0_ADDR0–DCP0_ADDR1)

0:9	DRBA	Decode Region Base Address	High-order decode region address bits
10:11		Reserved	
12:15	DRS	Decode Region Size 0000–0100 Reserved 0101 4MB 0110 8MB 0111 16MB 1000 32MB 1001 64MB 1010 128MB 1011 256MB 1100 512MB 1101 1GB 1110 2GB 1111 4GB	
16:30		Reserved	
31	DREN	Enable Decode Region 0 Decoding is disabled for region 1 Decoding is enabled for region.	

13.2.3 Decompression Configuration Register (DCP0_CFG)

DCP0_CFG is used to configure the operation of the decompression controller.

DCP0_CFG[IKB] must be set to 0 to enable decompression. The value of this field determines whether the U0 storage attribute is recognized; the storage attribute must be recognized for decompression to occur.

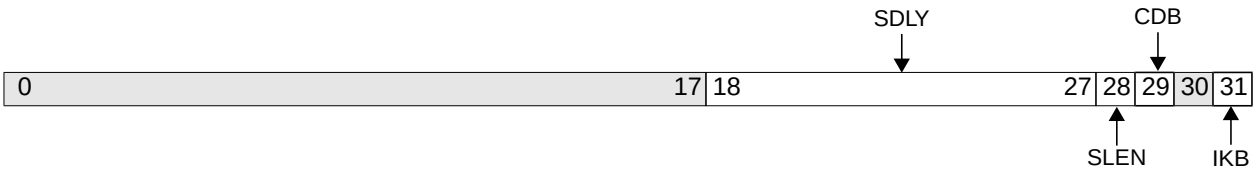


Figure 13-2. Decompression Controller Configuration Register (DCP0_CFG)

0:17		Reserved	
18:27	SLDY	Sleep Delay	Sleep delay
18:27	SLDY	Sleep Delay	Sleep delay
28	SLEN	Sleep Enable 0 Disable sleep 1 Enable sleep	
29	CDB	Clear Decompression Buffer 0 Normal operation 1 Clear decompression buffer	Self-clearing; always reads 0
30		Reserved	
31	IKB	Enable Decrompression 0 Decompression is enabled; U0 storage attribute is recognized 1 Decompression is disabled; U0 storage attribute is ignored	

13.2.4 Decompression Controller ID Register (DCP0_ID)

DCP0_ID contains the decompression controller ID.

Figure 13-3. Decompression Controller ID Register (DCP0_ID)

0:31		Decompression Controller ID	Read-only, value is 0000504D

13.2.5 Decompression Controller Version Register (DCP0_VER)

DCP0_VER contains the decompression controller version.

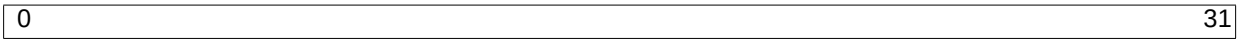


Figure 13-4. Decompression Controller Version Register (DCP0_VER)

0:31		Decompression ControllerVersion	Read-only, value is 0x00000200

13.2.6 Decompression Controller PLB Error Address Register (DCP0_PLBBEAR)

On an error, DCP0_PLBBEAR contains the address on the PLB that caused the decompression controller to generate cycles to the SDRAM and EBC memory controllers.

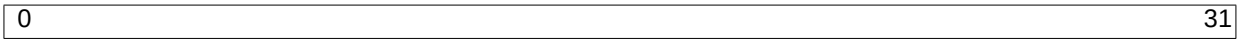


Figure 13-5. Decompression Controller PLB Error Address Register (DCP0_PLBBEAR)

0:31		Address of PLB Error

13.2.7 Decompression Controller Bus Error Address Register (DCP0_MEMBEAR)

On an error, DCP0_MEMBEAR contains the address, presented to the memory controllers, that caused an error in the decompression controller. The address was presented to the memory controllers for either an ITE fetch or a block burst request.

Figure 13-6. Decompression Controller Bus Error Address Register (DCP0_MEMBEAR)

0:31		Address of SDRAM or EBC Error

13.2.8 Decompression Controller Error Status Register 0 (DCP0_ESR)

DCP0_ESR reports error status.

Status information includes error type and master read/write status. DCP0_ESR0 field and DCP0_PLBBEAR or DCP0_MEMBEAR lock status is also reported.

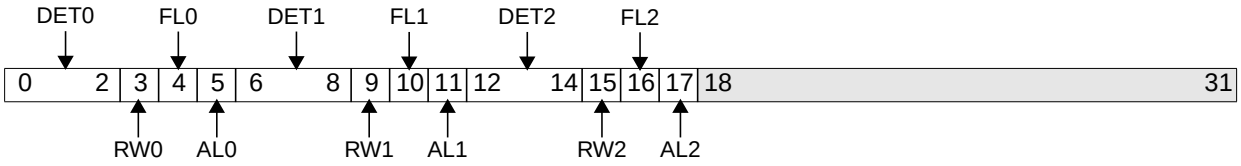


Figure 13-7. Decompression Controller Error Status Register 0 (DCP0_ESR)

0:2	DET0	Decompression Error Type for Master 0 000 Time-out during ITE fetch 001 Time-out during block fetch 010 Memory controller error during ITE fetch 011 Memory controller error during block fetch 100–111 Reserved	Master 0 is the processor core instruction cache unit (ICU).
3	RW0	Read/Write Status for Master 0 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
4	FL0	DCP0_ESR Field Lock for Master 0 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
5	AL0	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 0 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
6:8	DET1	Decompression Error Type for Master 1	See description for DCP0_ESR[DET0]. Master1 is the processor core data cache unit (DCU).
9	RW1	Read/Write Status for Master 1 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
10	FL1	DCP0_ESR Field Lock for Master 1 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
11	AL1	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 1 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
12:14	DET2	Decompression Error Type for Master 2	See description for DCP0_ESR[DET0]. Master 2 is the external master.

15	RW2	Read/Write Status for Master 2 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
16	FL2	DCP0_ESR Field Lock for Master 2 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
17	AL2	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 2 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
18:31		Reserved	

For more information, refer to *CodePack™ PowerPC Code Compression Utility User's Manual,* Version 3.0.

Part IV. PPC405CR External Interfaces

Chapter 14. SDRAM Controller

The SDRAM controller provides a 32-bit interface to SDRAM memory with optional Error Checking and Correction (ECC). The memory controller provides flexible, fully programmable timings for interfacing to a wide variety of SDRAM devices. It supports four physical banks of dual- or quad-bank SDRAMs, where each physical bank is from 4 MB to 256 MB in size. In addition, the SDRAM controller supports the use of registered SDRAMs.

The controller supports page mode operation with bank interleaving and maintains up to four open pages. To improve performance, the controller features separate 32-byte read and 128-byte write buffers. For applications where data integrity must be guaranteed, optional ECC provides standard single error correction and double error detection. Designers also have the opportunity to reduce system power by placing the SDRAM controller in sleep and/or self-refresh mode.

14.1 Interface Signals

In many systems the memory controller can directly interface to SDRAM, relieving designers from the need to buffer signals or add circuitry to phase adjust the SDRAM clock. However, since each application is different, a detailed timing analysis should always be performed. Please note that while the SDRAM address signals (MemAddr12:0 and BA1:0) utilize industry standard bit ordering, the data bus (MemData0:31) follows the PowerPC bit numbering convention. That is, MemData0 is the most significant bit and DQM0 is the byte enable for MemData0:7.

Figure 14-1 illustrates the SDRAM signal I/Os.

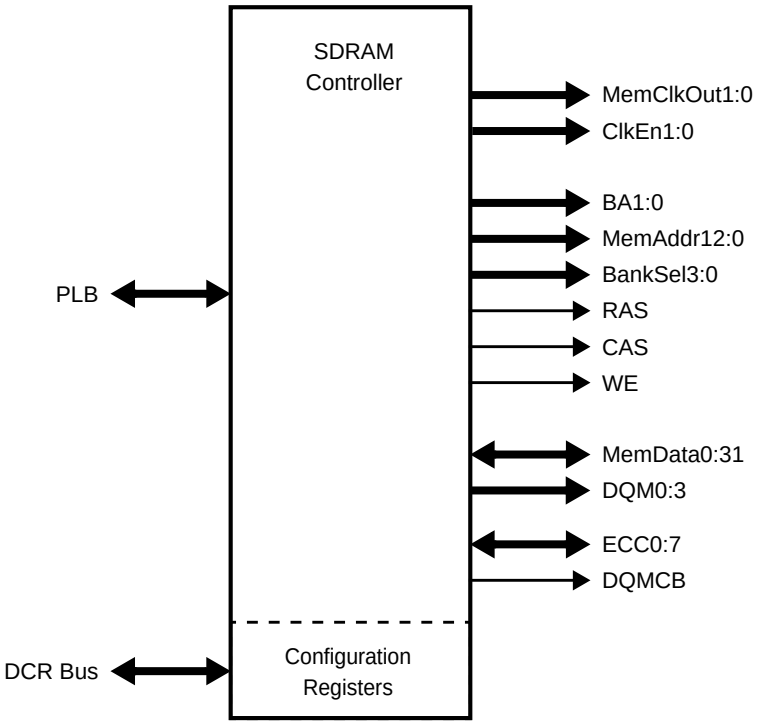


Figure 14-1. SDRAM Controller Signals

The usage and signal state after a PPC405CR reset for each of these signals is shown in Table 14-1 on page 14-2.

Table 14-1. SDRAM Signal Usage and State During/Following Reset

Signal	Reset State	Usage
MemClkOut1:0	Toggling[1]	Two copies of the SDRAM clock.
ClkEn1:0	1	SDRAM clock enable.
BA1:0	2'b0	Bank address. Used to select an internal SDRAM bank in dual- and quad-bank SDRAM devices.
MemAddr12:0	13b'0	Memory address. See Table 14.3.5, "Logical Address to Memory Address Mapping," on page 14-8 for additional details.
$\overline{\text{BankSel3:0}}$	4b'1	Bank Selects. Used to select between different physical banks of SDRAM memory.
$\overline{\text{RAS}}$	1	Row Address Strobe.
$\overline{\text{CAS}}$	1	Column Address Strobe.
$\overline{\text{WE}}$	1	Write Enable.
MemData0:31	High-Z	Data input/output. MemData0 is the most significant bit.
DQM0:3	1	Data Mask, an input mask for write accesses and an output enable during read operations. DQM0 applies to MemData0:7, DQM1 to MemData8:15, DQM2 to MemData16:23 and DQM3 to MemData24:31.
ECC0:7	High-Z	ECC check bits.
DQMCB	1	Data Mask for ECC check bits.

Note 1: During power-up, MemClkOut1:0 tracks SysClk until the internal PLL begins to lock. At that time MemClkOut1:0 transitions to the clock rate configured via the strapping resistors.

14.2 Accessing SDRAM Registers

After a system reset, software is required to configure and then enable the SDRAM controller. When this is complete, the SDRAM memory is ready for access by the processor, or any other master in the PPC405CR. Once the controller is operating, it is usually not necessarily for software to access the SDRAM configuration registers. The status registers, however, are useful for determining the state of the memory controller or querying information regarding any corrected or uncorrectable ECC errors that may have occurred.

All SDRAM configuration and status registers are accessed using the **mtdcr** and **mfdcr** PowerPC instructions. Access to these registers is performed using an indirect addressing method through the SDRAM0_CFGADDR and SDRAM0_CFGDATA registers.

Table 14-2. SDRAM Controller DCR Addresses

Register	DCR Address	Access	Description
SDRAM0_CFGADDR	0x010	R/W	SDRAM Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	SDRAM Controller Data Register

Table 14-3 lists the indirectly accessed SDRAM configuration and status registers.

Table 14-3. SDRAM Controller Configuration and Status Registers

Mnemonic	Address Offset	Access	Description	Page
SDRAM0_BESR0	0x00	R/Clear	Bus Error Syndrome Register 0	14-17
SDRAM0_BESR1	0x08	R/Clear	Bus Error Syndrome Register 1	14-17
SDRAM0_BEAR	0x10	R/W	Bus Error Address Register	14-16
SDRAM0_CFG	0x20	R/W	SDRAM Configuration	14-3
SDRAM0_STATUS	0x24	R	SDRAM Controller Status	14-5
SDRAM0_RTR	0x30	R/W	Refresh Timer Register	14-13
SDRAM0_PMIT	0x34	R/W	Power Management Idle Timer	14-19
SDRAM0_B0CR	0x40	R/W	Memory Bank 0 Configuration Register	14-5
SDRAM0_B1CR	0x44	R/W	Memory Bank 1 Configuration Register	14-5
SDRAM0_B2CR	0x48	R/W	Memory Bank 2 Configuration Register	14-5
SDRAM0_B3CR	0x4C	R/W	Memory Bank 3 Configuration Register	14-5
SDRAM0_TR	0x80	R/W	SDRAM Timing Register	14-9
SDRAM0_ECCCFG	0x94	R/W	ECC Configuration	14-14
SDRAM0_ECCESR	0x98	R	ECC Error Status	14-16

To read or write one of these SDRAM controller registers, software must first write the register's address offset into the SDRAM0_CFGADDR register. The target register can then be read or written through the SDRAM0_CFGDATA DCR address. The following PowerPC code illustrates this procedure by reading the SDRAM0_CFG register.

```
li        r3,SDRAM0_CFG              ! address offset of SDRAM0_CFG
mtdcr     SDRAM0_CFGADDR,r3          ! set offset address
mfdcr     r4,SDRAM0_CFGDATA          ! read value of SDRAM0_CFG
```

Programming Note: Reserved fields in SDRAM controller configuration registers must not change when the register is written. To modify bit fields within a register, read the register, use a mask to clear the target bits, OR in the new field value, and then write the result back.

14.3 SDRAM Controller Configuration and Status

Software must program a number of SDRAM control registers before the SDRAM controller can be started and memory accessed. At a minimum, this involves writing to the SDRAM Configuration Register (SDRAM0_CFG), SDRAM Timing Register (SDRAM0_TR) and one or more Memory Bank Configuration Registers (SDRAM0_BnCR).

14.3.1 Memory Controller Configuration Register (SDRAM0_CFG)

After a system reset the SDRAM controller is disabled and must be configured before memory transactions can occur. The Memory Controller Configuration Register (SDRAM0_CFG) serves to both enable the controller and select from various memory controller features. These features include enabling power management, ECC error checking, and registered SDRAM support. Each of these settings is global and applies to the entire SDRAM subsystem.

Prior to enabling the SDRAM controller by setting SDRAM0_CFG[DCE], the SDRAM0_TR, SDRAM0_RTR, SDRAM0_PMIT, and SDRAM0_BnCR registers must be configured. This is because

once SDRAM0_CFG[DCE]=1 writing any of the listed SDRAM registers does not actually update the target register. Write access to SDRAM0_CFG is independent of SDRAM0_CFG[DCE]. However, software must ensure that the SDRAM controller is idle when updating SDRAM0_CFG[3:10]. This guarantees that the register update does not affect any in-progress SDRAM operations.

Before software enables the SDRAM controller by setting SDRAM0_CFG[DCE] it must ensure that the SDRAM power-on delay has been satisfied. For example, the IBM 16MB SDRAM requires a 100µs pause; the 64MB SDRAM requires a 200µs pause. Once enabled, the SDRAM controller automatically performs the following initialization procedure:

1. Issues the precharge command to all banks.
2. Waits SDRAM0_TR[PTA] cycles.
3. Performs eight $\overline{CAS}$ before $\overline{RAS}$ refresh cycles, each separated by SDRAM0_TR[RFTA] clock cycles)
4. Issues the mode register write command to each bank.
5. Perform eight $\overline{CAS}$ before $\overline{RAS}$ refresh cycles (each separated by SDRAM0_TR[RFTA] clock cycles).
6. Waits SDRAM0_TR[RFTA] clock cycles.

The SDRAM is then available for read and write access.

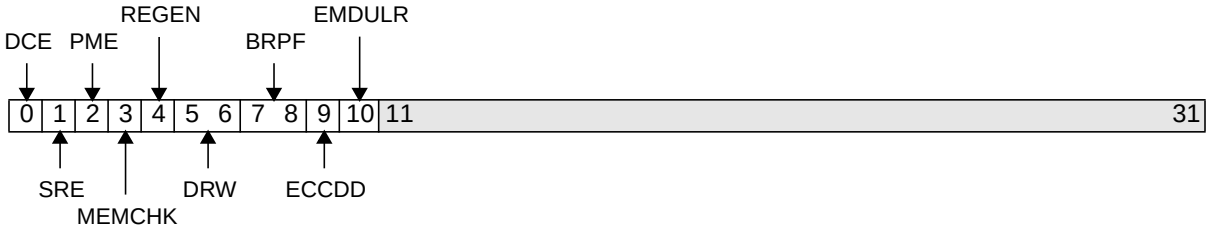


Figure 14-2. Memory Controller Configuration (SDRAM0_CFG)

0	DCE	SDRAM Controller Enable 0 Disable 1 Enable	All SDRAM controller configuration registers must be initialized and valid prior to setting DCE.
1	SRE	Self-Refresh Enable 0 Disable 1 Enable	See “Self-Refresh” on page 14-18.
2	PME	Power Management Enable 0 Disable 1 Enabled	See “Power Management” on page 14-19.
3	MEMCHK	Memory Data Error Checking 0 None 1 ECC	See “Error Checking and Correction (ECC)” on page 14-14.
4	REGEN	Registered Memory Enable 0 Disabled 1 Enabled	

5:6	DRW	SDRAM Width 00 32-bit 01 Reserved 10 Reserved 11 Reserved	Must be set to 0b00.
7:8	BRPF	Burst Read Prefetch Granularity 00 Reserved 01 16 bytes 10 32 bytes 11 Reserved	Most applications should set with field to 0b01.
9	ECCDD	ECC Driver Disable 0 Check bit data output on ECC0:7. 1 ECC0:7 are placed in high-Z state.	Regardless of whether ECC checking is enabled, SDRAM writes cause the chip to write check bit data on ECC0:7. If ECC is unused, setting ECCDD = 1 disables these drivers.
10	EMDULR	Enable Memory Data Unless Read 0 MemData0:31 are high-Z unless a memory write is being performed. 1 MemData0:31 are driven unless a memory read is being performed.	
11:31		Reserved	

14.3.2 Memory Controller Status (SDRAM0_STATUS)

The Memory Controller Status Register (SDRAM0_STATUS) allows software to determine the current state of the SDRAM controller. If SDRAM0_STATUS[MRSCMP]=0 the SDRAM is not accessible for either read or write operations. Similarly, the SDRAM is also inaccessible when the SDRAM is in self-refresh mode, SDRAM0_STATUS[SRSTATUS]=1.

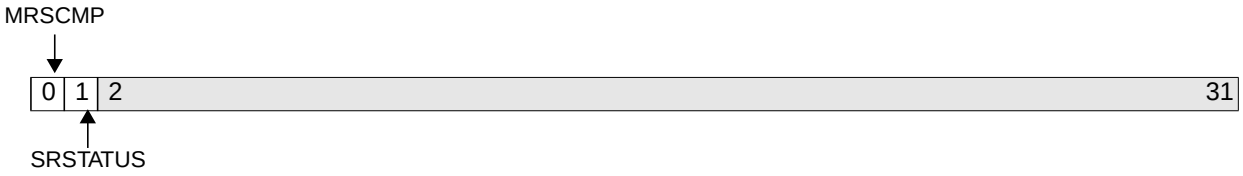


Figure 14-3. Memory Controller Status (SDRAM0_STATUS)

0	MRSCMP	Mode Register Set Complete 0 MRS not complete 1 MRS completed	Set to 1 when the SDRAM controller completes the Mode Register Set Command, which results from setting SDRAM0_CFG[DCE]. Clearing SDRAM0_CFG[DCE] causes this bit to clear in the following MemClkOut1:0 cycle.
1	SRSTATUS	Self-Refresh State 0 Not in Self-Refresh Mode 1 Self-Refresh Mode	See “Self-Refresh” on page 14-18.
2:31		Reserved	

14.3.3 Memory Bank 0–3 Configuration (SDRAM0_B0CR–SDRAM0_B3CR)

These registers are used to configure and enable memory in each respective bank. Only SDRAM banks with SDRAM0_BnCR[BE]=1 are initialized when SDRAM0_CFG[DCE] is set to 1 and

subsequently available for access. Since the SDRAM0_BnCR registers cannot be modified when SDRAM0_CFG[DCE]=1, adding or removing memory banks requires that the SDRAM controller be disabled and then reinitialized.

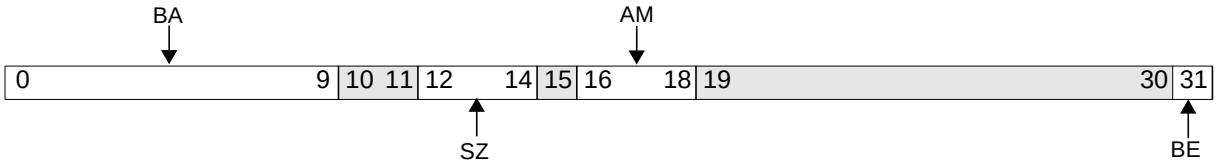


Figure 14-4. Memory Bank 0–3 Configuration Registers (SDRAM0_B0CR–SDRAM0_B3CR)

0:9	BA	Base Address	The base address must be aligned on a boundary that matches the size of the region as defined by the SZ field. For example, a 4MB region must begin on an address that is divisible by 4MB.
10:11		Reserved	
12:14	SZ	Size 000 4M byte 001 8M byte 010 16M byte 011 32M byte 100 64M byte 101 128M byte 110 256M byte 111 Reserved	
15		Reserved	
16:18	AM	Addressing Mode 000 Mode 1 001 Mode 2 010 Mode 3 011 Mode 4 100 Mode 5 101 Mode 6 110 Mode 7 111 Reserved	See “SDRAM Addressing Modes” on page 14-7.
19:30		Reserved	
31	BE	Memory Bank Enable 0 Bank is disabled 1 Bank is enabled	

.

Table 14-4. SDRAM Addressing Modes

Addressing Mode	SDRAM Memory Organization
1	11 x 9 - 2 Bank 11 x 10 - 2 Bank
2	12 x 9 - 4 Bank 12 x 10 - 4 Bank
3	13 x 9 - 4 Bank 13 x 10 - 4 Bank 13 x 11 - 4 Bank
4	12 x 8 - 2 Bank 12 x 8 - 4 Bank 13 x 8 - 2 Bank
5	11 x 8 - 2 Bank 11 x 8 - 4 Bank
6	13 x 8 - 2 Bank 13 x 8 - 4 Bank
7	13 x 9 - 2 Bank 13 x 10 - 2 Bank

14.3.4 Page Management

The SDRAM controller supports page mode operation with bank interleaving and maintains up to four open pages in the memory subsystem. The SDRAM controller page management unit (PMU) tracks memory accesses (activate, read/write, precharge, and refresh) and maintains a directory of up to four open pages. All PMU entries are allocated and deallocated based on current and pending accesses. Allocated entries are checked against the address of the pending access, and a page hit occurs when a match exists. All PMU directory entries are deallocated when a $\overline{CAS}$ before $\overline{RAS}$ refresh occurs.

Open pages can be spread across the system memory array on different bank selects ($\overline{BankSeln}$) or be contained in a single bank select, depending on the memory access sequences and the memory subsystem implementation. For a single bank memory subsystem, the number of open pages is limited to the number of internal banks associated with the SDRAM devices in that bank. For example, a single bank implementation consisting of SDRAMs with two internal banks can have two open pages. In this case, the maximum of two open pages is a limitation of the memory subsystem implementation, not the SDRAM controller.

The SDRAM page size for page hits varies, depending on the address mode programmed in SDRAM0_BnCR[AM]. Table 14-5, “SDRAM Page Size,” on page 14-7 details the relationship of the address mode to the page size.

Table 14-5. SDRAM Page Size

Address Mode	Page Size
1,2,3,7	2 KB
4,5,6	1 KB

14.3.5 Logical Address to Memory Address Mapping

SDRAM memory requires that addresses be divided into row and column portions. The relationship between a logical address and the SDRAM row and column address is determined by the address mode programmed in SDRAM0_BnCR[AM] and is shown in Table 14-6.

Table 14-6. Logical Address Bit on BA1:0 and MemAddr12:0 Versus Addressing Mode.

Mode	Bank Size	Address Phase	BA		MemAddr												
	Organization[1]		1	0	12	11	10	9	8	7	6	5	4	3	2	1	0
1	8 MB	Row	7	9	7	9	10	11	12	13	14	15	16	17	18	19	20
	11 x 9 x 2	Column	7	9	7	9	AP	8	21	22	23	24	25	26	27	28	29
	16 MB	Row	7	9	7	9	10	11	12	13	14	15	16	17	18	19	20
	11 x 10 x 2	Column	7	9	7	9	AP	8	21	22	23	24	25	26	27	28	29
2	32 MB	Row	7	8	7	9	10	11	12	13	14	15	16	17	18	19	20
	12 x 9 x 4	Column	7	8	7	4	AP	6	21	22	23	24	25	26	27	28	29
	64 MB	Row	7	8	7	9	10	11	12	13	14	15	16	17	18	19	20
	12 x 10 x 4	Column	7	8	7	4	AP	6	21	22	23	24	25	26	27	28	29
3	64 MB	Row	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 9 x 4	Column	6	7	7	4	AP	5	21	22	23	24	25	26	27	28	29
	128 MB	Row	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 10 x 4	Column	6	7	7	4	AP	5	21	22	23	24	25	26	27	28	29
	256 MB	Row	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 11 x 4	Column	6	7	7	4[2]	AP	5	21	22	23	24	25	26	27	28	29
4	8/16 MB	Row	8	21	8	9	10	11	12	13	14	15	16	17	18	19	20
	12 x 8 x 2/4	Column	8	21	8	4	AP	6	21	22	23	24	25	26	27	28	29
5	4/8 MB	Row	9	21	9	21	10	11	12	13	14	15	16	17	18	19	20
	11 x 8 x 2/4	Column	9	21	9	21	AP	8	21	22	23	24	25	26	27	28	29
6	16/32 MB	Row	7	21	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 8 x 2/4	Column	7	21	8	9	AP	8	21	22	23	24	25	26	27	28	29
7	32 MB	Row	7	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 9 x 2	Column	7	7	7	4	AP	6	21	22	23	24	25	26	27	28	29
	64 MB	Row	7	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	13 x 10 x 2	Column	7	7	7	4	AP	6	21	22	23	24	25	26	27	28	29

Note 1: Memory organization is the number of rows x columns x internal banks.

Note 2: Column address bit 10 sent out on MemAddr11 for 13 x 11 x 4 parts.

Note 3: All read operations transfer 64-bits from SDRAM memory to the controller. Therefore, MemAddr0 is first driven with 0 and then 1. For data items narrower than 64-bits, the requested byte(s) are fulfilled from the 64-bit doubleword.

14.3.6 SDRAM Timing Register (SDRAM0_TR)

The SDRAM Timing Register sets the timing parameters for all SDRAM memory banks. This register must be written prior to setting SDRAM0_CFG[DCE]. If SDRAM0_CFG[DCE]=1, writes to this register appear to complete, but do not affect the contents of SDRAM0_TR.

If the SDRAM interface is operated in registered mode, (SDRAM0_CFG[REGE]=1) a programmed $\overline{\text{CAS}}$ latency of 2 clocks (SDRAM0_TR[CASL] = 2‘b01) corresponds to a registered $\overline{\text{CAS}}$ latency of 3 clocks, and a programmed latency of 3 clocks (SDRAM0_TR[CASL] = 2‘b10) corresponds to a registered $\overline{\text{CAS}}$ latency of 4 clocks. Programming the $\overline{\text{CAS}}$ latency to 4 clocks (SDRAM0_TR[CASL] = 2‘b11), corresponding to a registered $\overline{\text{CAS}}$ latency of 5 clocks, is not supported.

See “Selected Timing Diagrams” on page 14-10 for timing diagrams illustrating how the fields in SDRAM0_TR affect the signalling on the SDRAM memory interface.

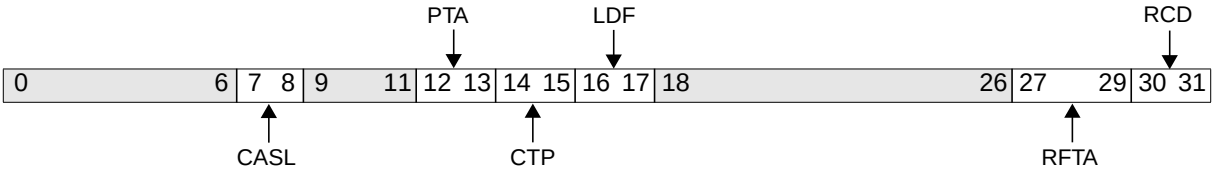


Figure 14-5. SDRAM Timing Register (SDRAM0_TR)

0:6		Reserved
7:8	CASL	SDRAM $\overline{\text{CAS}}$ latency. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
9:11		Reserved
12:13	PTA	SDRAM Precharge Command to next Activate Command minimum. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
14:15	CTP	SDRAM Read / Write Command to Precharge Command minimum. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
16:17	LDF	SDRAM Command Leadoff. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
18:26		Reserved

27:29	RFTA	SDRAM $\overline{CAS}$ before $\overline{RAS}$ Refresh Command to next Activate Command minimum. 000 4 MemClkOut1:0 cycles 001 5 MemClkOut1:0 cycles 010 6 MemClkOut1:0 cycles 011 7 MemClkOut1:0 cycles 100 8 MemClkOut1:0 cycles 101 9 MemClkOut1:0 cycles 110 10 MemClkOut1:0 cycles 111 Reserved
30:31	RCD	SDRAM RAS to CAS Delay 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles

14.3.7 Selected Timing Diagrams

The SDRAM controller is capable of performing many different types of read and write operations. For example, while the CPU generally issues four doubleword line fills and line writes, the EBC external master interface can issue various types of burst transactions. Since the SDRAM controller is often servicing read and write requests from several masters, the exact sequence of operations on the external SDRAM interface cannot be predicted. Therefore, the timing diagrams in this section should be considered as examples of the signalling that can be observed on the SDRAM interface, and not the only types of transactions that occur.

The timing diagrams in this section are intended to illustrate cycle-based SDRAM programmable timing parameters only. As such, AC specific timing information should not be inferred from the timing diagrams. Instead, please refer to the PPC405CR data sheet for AC specifications.

Table 14-7 summarizes the SDRAM memory timing parameters used to annotate the waveforms. These parameters are set in the SDRAM Timing Register (SDRAM0_TR).

Table 14-7. SDRAM Memory Timing Parameters

Name	Function	Description
RCD	Activate to Read/Write Command	Minimum number of clock cycles from an activate command to a read or write command. Corresponds to SDRAM $\overline{RAS}$ to $\overline{CAS}$ assertion delay.
RFTA	Refresh to Activate	Minimum number of clock cycles from a $\overline{CAS}$ before $\overline{RAS}$ refresh command to the next activate command.
CTP	Command to Precharge	Minimum number of clock cycles from a read or write command to a precharge command.
PTA	Precharge to Active	Minimum number of clock cycles required to wait following a Precharge Command to issuing the next activate command.
CASL	$\overline{CAS}$ Latency	$\overline{CAS}$ access latency.
LDF	Command Leadoff Delay	Number of clock cycles from address/command assertion to bank select ($\overline{BankSeln}$) assertion.

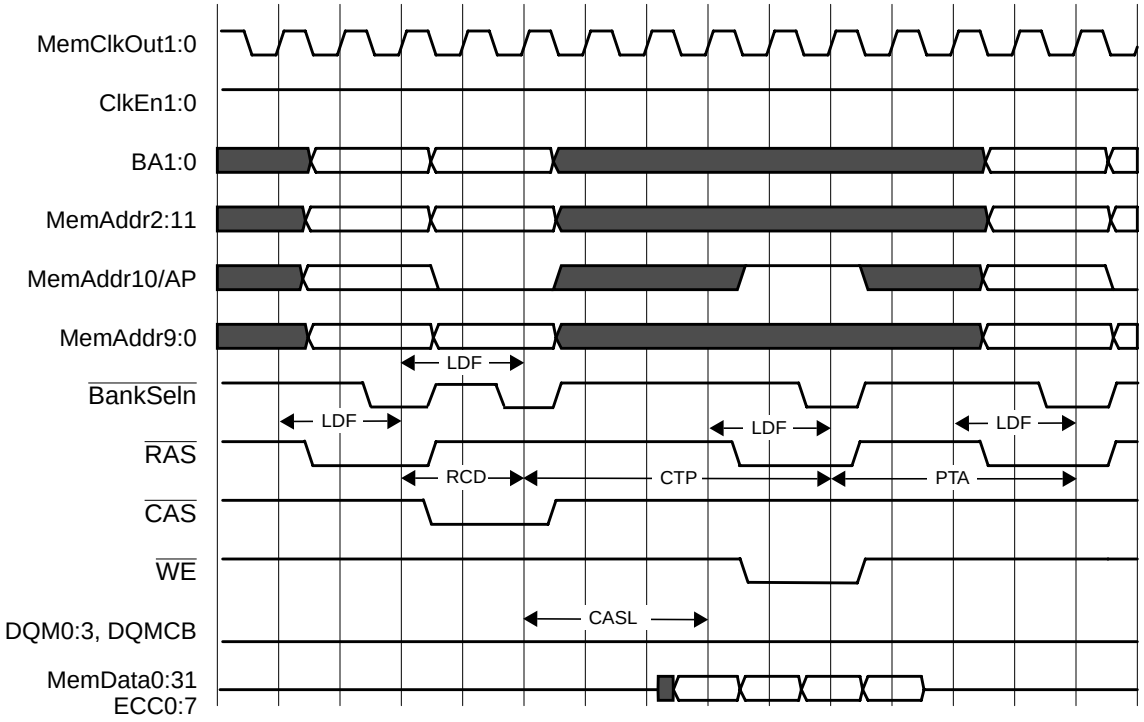


Figure 14-6. Activate, Four Word Read, Precharge, Activate

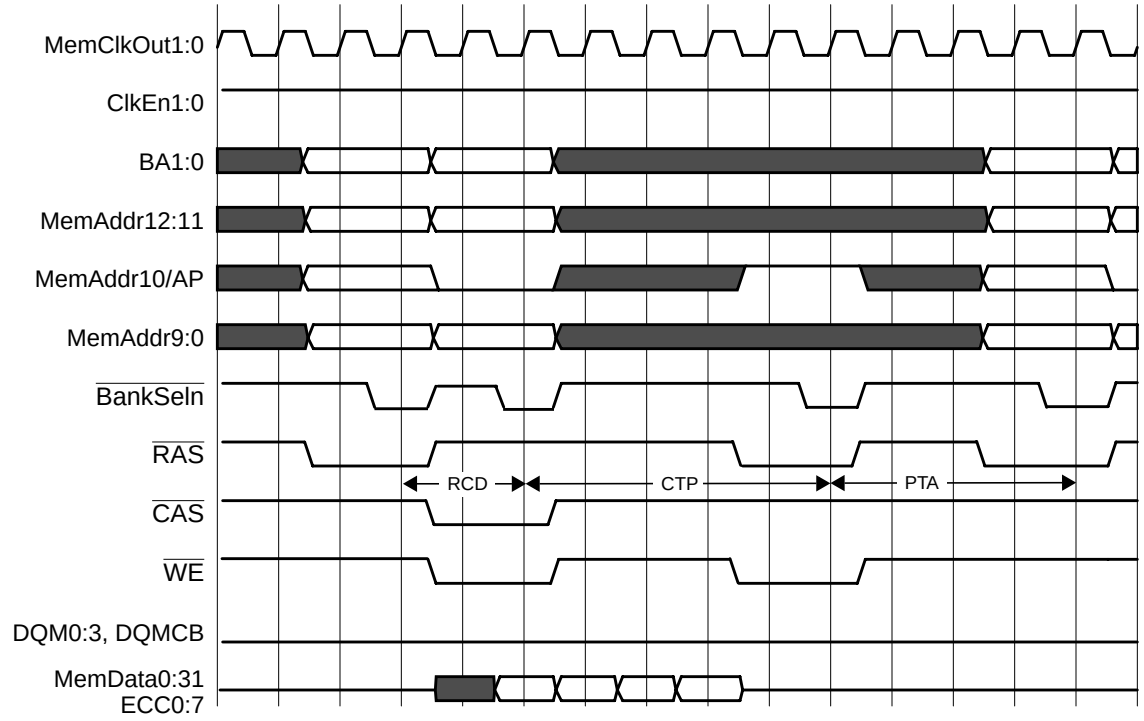


Figure 14-7. Activate, Four Word Write, Precharge, Activate

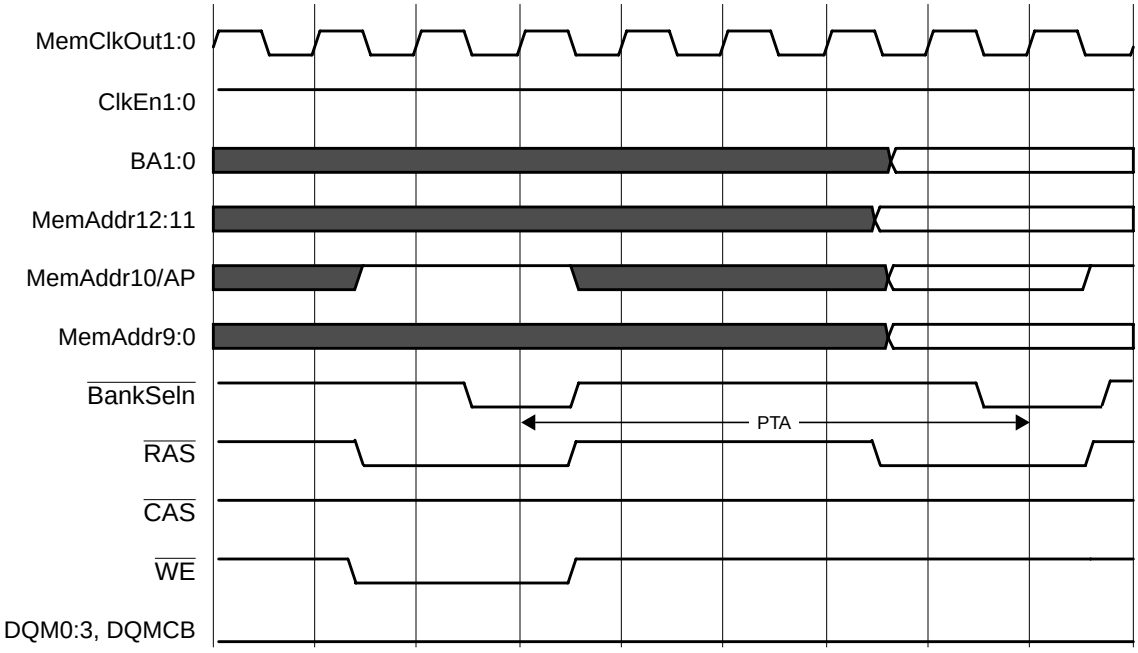


Figure 14-8. Precharge All, Activate

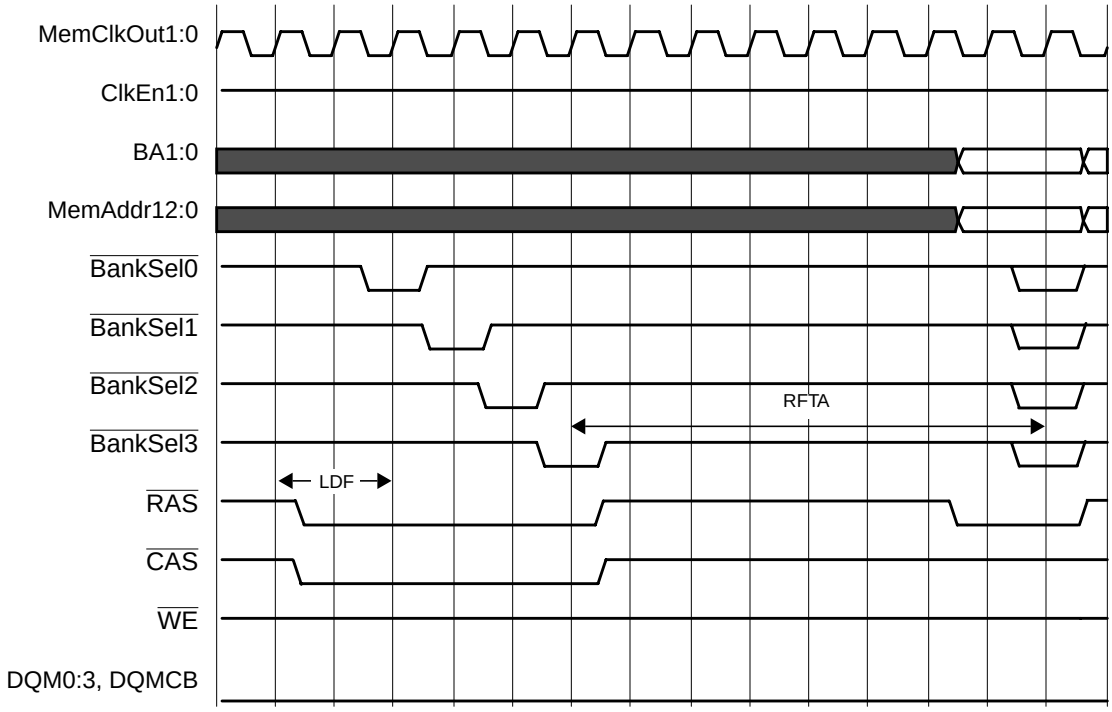


Figure 14-9. CAS Before RAS Refresh

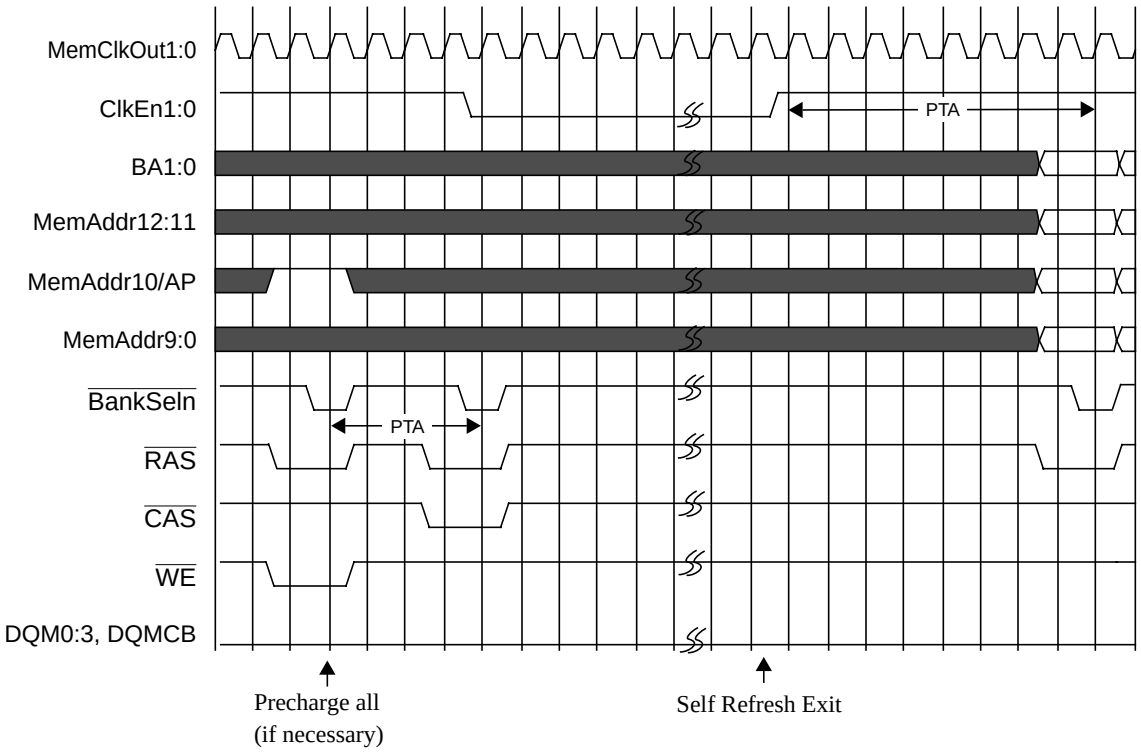


Figure 14-10. Self-Refresh Entry and Exit

14.3.8 Auto ($\overline{\text{CAS}}$ Before $\overline{\text{RAS}}$) Refresh

Refresh of odd memory banks is staggered from the refresh of even memory banks. Only enabled SDRAM banks (SDRAM0_BnCR[BE]=1) are initialized when the controller is enabled (SDRAM0_CFG[DCE] set to 1) and refreshed during normal operation. Once the memory controller is enabled and the initialization sequence has completed, the refresh mechanism starts automatically with refreshing of the memory continuing independent of SDRAM0_CFG[DCE].

Refresh requests are generated internally when the refresh timer expires. The refresh interval is programmable via the Refresh Timer Register (SDRAM0_RTR). During refresh, all SDRAM accesses are delayed until the refresh cycle completes.

14.3.9 Refresh Timer Register (SDRAM0_RTR)

The Refresh Timer Register determines the memory refresh rate for the SDRAM. The internal counter runs at the controller clock frequency, thus if MemClkOut1:0 is 100 MHz, a value of 0x05F0 produces a refresh interval of 15.20 s (1520 x 10 ns = 15.20 s). This register is programmable to accommodate other SDRAM clock frequencies.

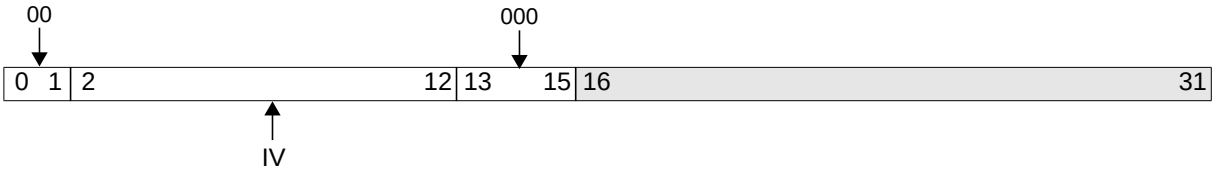


Figure 14-11. Refresh Timing Register (SDRAM0_RTR)

0:1		Always 0b00	
2:12	IV	Interval Programmable between 0b00000000000 and 0b11111111111 (that is, 0x0000 to 0x3FF8 for the 16-bit field).	Reset value is 0b00010111110 (that is, 0x05F0 for the complete 16-bit field)
13:15		Always 0b000	
16:31		Reserved	

14.4 Error Checking and Correction (ECC)

Error Checking and Correction (ECC) corrects all single bit errors and detects all double bit errors when reading from SDRAM memory. In addition, ECC detects any and all errors which may exist in an aligned 4-bit nibble. As detailed in Table 14-8, “Additional Latency when using ECC,” on page 14-14, the ECC function is transparent in terms of latency, except for partial write operations. On partial writes, a read-modify-write sequence, including bus turn-around, is required to generate the write check bits and store the resultant data.

Table 14-8. Additional Latency when using ECC

PLB Transaction	Added Latency
Read	None
Burst or full single beat write	None
Partial write	SDRAM0_TR[CASL] + 4 clocks

ECC is enabled for all SDRAM banks by setting SDRAM0_CFG[MEMCHK]=1 during the SDRAM initialization procedure. Software is then required to write each word of SDRAM to set the check bits to a valid state. When this is complete, ECC checking and correction is enabled on a per-bank basis through the SDRAM0_ECCCFG register.

14.4.1 ECC Configuration Register (SDRAM0_ECCCFG)

Write access to SDRAM0_ECCCFG is independent of SDRAM0_CFG[DCE]. Software must ensure that the SDRAM controller is idle when updating SDRAM0_ECCCFG. This guarantees that the register update does not affect any in-progress SDRAM operations.

Figure 14-12. ECC Configuration Register (SDRAM0_ECCCFG)

0:7		Reserved	
8:11	CEn	ECC Correction Enable for bank n. 0 Disabled 1 Enabled	When CEn is set, ECC correction is enabled for bank n ($\overline{\text{BankSeln}}$). When cleared, the ECC logic ignores the check bits and passes read data along unmodified.
12:31		Reserved	

14.4.2 Correctable ECC Errors

During SDRAM memory read operations the ECC logic automatically detects and corrects any single bit error that occurs within each 32-bit word of SDRAM memory. This includes errors detected during the read portion of the read-modify-sequence sequence required for partial (less than 64-bit) SDRAM writes.

If a correctable ECC error occurs during a memory read the corrected data is returned to the requesting master. When a correctable error is detected during the read portion of a partial write the corrected data is combined with the write data and written back to memory with new ECC check bits. In both cases, the memory bank ($\overline{\text{BankSeln}}$) that caused the error is logged in SDRAM0_ECCESR[BKnE] and the byte lane that experienced the correctable error is identified in SDRAM0_ECCESR[BLnCE]. Furthermore, the correctable error bit (SDRAM0_ECCESR[CE]) is set causing an ECC Correctable Error interrupt to the Universal Interrupt Controller. The interrupt remains active until software clears SDRAM0_ECCESR[CE].

14.4.3 Uncorrectable ECC Errors

Uncorrectable ECC errors may occur during SDRAM read and SDRAM partial write operations. An uncorrectable error detected during a memory read results in the data from system memory (unchanged) being returned to the requesting master. An uncorrectable error detected on the read portion of a read-modify-write sequence for an SDRAM partial write results in the data from system memory (unchanged) being combined with the write data and written back to memory with new ECC check bits.

Whenever an uncorrectable ECC error occurs the errant memory address is captured in SDRAM0_BEAR. The error status for the master that initiated the memory operation is logged in either SDRAM0_BESR0 or SDRAM0_BESR1. The memory bank ($\overline{\text{BanknSel}}$) that experienced the error is recorded in SDRAM0_ECCESR[BKnE] and an uncorrectable error is flagged via SDRAM0_ECCESR[UE].

Since an uncorrectable error also results in an error signal to the master that initiated the transfer, other side effects may occur. For example, an instruction fetch with an uncorrectable error causes a machine check. In the case of a DMA transfer, the DMA channel stops and an error is logged.

14.4.4 ECC Error Status Register (SDRAM0_ECCESR)

The ECC Error Status Register (SDRAM0_ECCESR) tracks ECC related errors encountered during SDRAM memory accesses. Bits in SDRAM0_ECCESR are cleared by writing a 32-bit value to SDRAM0_ECCESR with a 1 in any bit position that is to be cleared and 0 in all other bit positions.

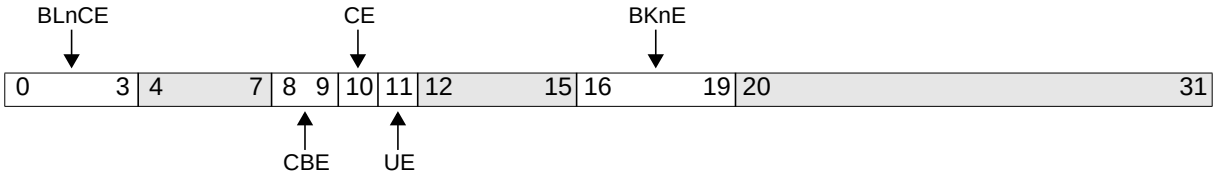


Figure 14-13. ECC Error Status Register (SDRAM0_ECCESR)

0:3	BLnCE	Byte Lane n Corrected Error 0 No error 1 Error occurred in byte lane n
4:7		Reserved
8:9	CBE	Error Detected in Check bits 00 No error 01 Error in lower check bits 10 Error in upper check bits 11 Error in both sets of check bits
10	CE	Correctable Error
11	UE	Uncorrectable Error
12:15		Reserved
16:19	BKnE	Bank n Error 0 No error 1 Error occurred in bank n
20:31		Reserved

14.4.5 Bus Error Address Register (SDRAM0_BEAR)

The SDRAM Bus Error Address Register (SDRAM0_BEAR) is a 32-bit register containing the address of the access where a correctable or uncorrectable ECC error occurred. The contents of the SDRAM0_BEAR are accessed indirectly through the SDRAM_CFGADDR and SDRAM0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

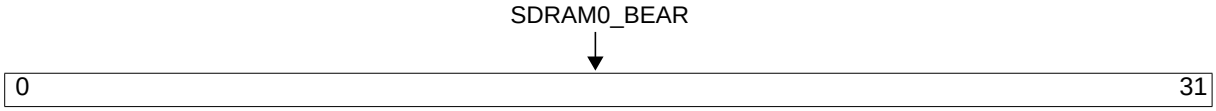


Figure 14-14. Bus Error Address Register (SDRAM0_BEAR)

0:31	SDRAM0_BEAR	Address of ECC Error.

14.4.6 Bus Error Syndrome Register 0 (SDRAM0_BESR0)

This register tracks errors encountered during CPU and external bus master accesses to SDRAM memory. Bits in SDRAM0_BESR0 are cleared by writing a 32-bit value to SDRAM0_BESR0 with a 1 in any bit position that is to be cleared and 0 in all other bit positions.

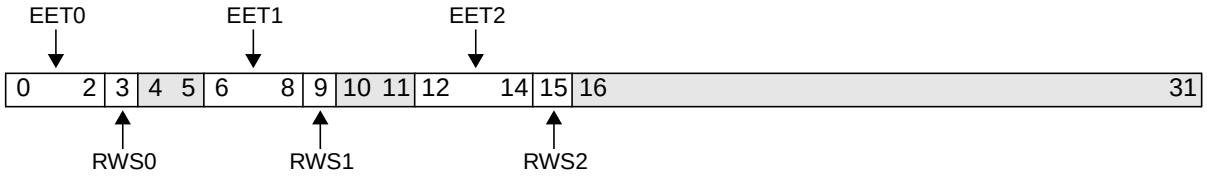


Figure 14-15. Bus Error Syndrome Register 0 (SDRAM0_BESR0)

0:2	EET0	Error type for master 0 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 0 is the processor instruction fetcher.
3	RWS0	Read/write status for master 0 0 Error operation was a write operation 1 Error operation was a read operation	
4:5		Reserved	
6:8	EET1	Error type for master 1 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 1 is the processor data side.
9	RWS1	Read/write status for master 1 0 Error operation was a write operation 1 Error operation was a read operation	
10:11		Reserved	
12:14	EET2	Error type for master 2 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 2 is the external bus master.
15	RWS2	Read/write status for master 2 0 Error operation was a write operation 1 Error operation was a read operation	
16:31		Reserved	

14.4.7 Bus Error Syndrome Register 1 (SDRAM0_BESR1)

This register tracks errors encountered during DMA accesses to SDRAM memory. Bits in SDRAM0_BESR1 are cleared by writing a 32-bit value to SDRAM0_BESR1 with a 1 in any bit position that is to be cleared and 0 in all other bit positions.

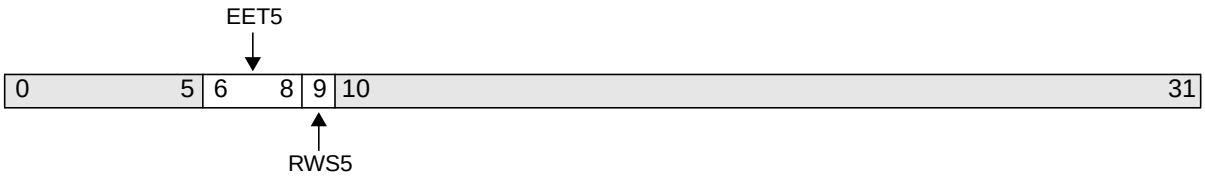


Figure 14-16. Bus Error Status Register 1 (SDRAM0_BESR1)

0:5		Reserved
6:8	EET5	Error type for master 5 Master 5 is the DMA controller. 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved
9	RWS5	Read/write status for master 5 0 Error operation was a write operation 1 Error operation was a read operation
10:31		Reserved

14.5 Self-Refresh

The SDRAM controller supports self-refresh operation for applications desiring lower power. When the SDRAM memory is placed in self-refresh mode it is no longer accessible for read or write accesses. Prior to placing the SDRAM controller in self-refresh mode all pending and previously queued requests targeting the SDRAM controller must be allowed to complete. Self-refresh entry is then initiated by setting SDRAM0_CFG[SRE]. When set, the SDRAM controller:

1. Completes the current SDRAM operation.
2. Issues precharge all commands to close all open pages.
3. Performs an auto-refresh cycle.
4. Enters self-refresh mode and sets SDRAM0_STATUS[SRSTATUS].

The SDRAM controller maintains the SDRAM in self-refresh mode, independent of any pending memory access requests, until SDRAM0_CFG[SRE] is cleared. Any attempt to read or write SDRAM memory during this time will stall the PLB.

Once SDRAM0_CFG[SRE] is cleared, the SDRAM controller performs the following:

1. Exits self-refresh mode.
2. Performs an auto-refresh cycle.
3. Clears SDRAM0_STATUS[SRSTATUS].

The SDRAM controller is then ready to service any memory request.

14.6 Power Management

The SDRAM controller provides a sleep mode where all SDRAM controller clocking is disabled with the exception of the SDRAM refresh logic and the power management wake-up logic. When the SDRAM controller is in sleep mode SDRAM refresh continues to preserve the contents of the memory and maintain the refresh interval.

14.6.1 Sleep Mode Entry

Sleep mode is enabled by setting SDRAM0_CFG[PME] and CPM0_ER[SDRAM]. Once sleep mode is enabled and the SDRAM controller has been idle for the number of cycles programmed in SDRAM0_PMIT, the SDRAM controller goes to sleep.

14.6.2 Power Management Idle Timer (SDRAM0_PMIT)

The SDRAM0_PMIT register determines the number for SDRAM clock (MemClkOut1:0) cycles that the controller must be idle before it asserts a sleep request when power management is enabled (SDRAM0_CFG[PME]=1). At system reset, SDRAM0_PMIT[CNT] is set to zero. This corresponds to a sleep request after 32 idle cycles.

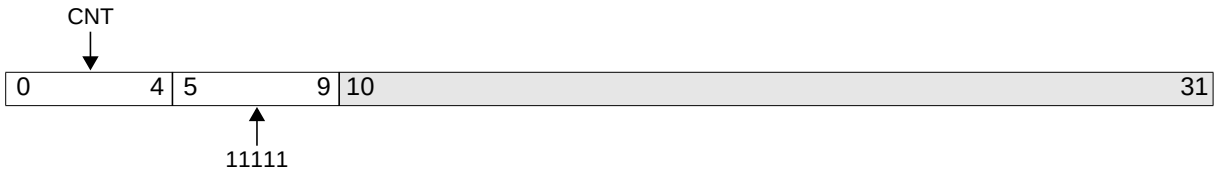


Figure 14-17. Power Management Idle Timer (SDRAM0_PMIT)

0:4	CNT	Count 0-31
5:9		Always 0b11111
10:31		Reserved

14.6.3 Sleep Mode Exit

The power management wake-up logic monitors the PLB for SDRAM reads or writes from other masters. In addition, the wakeup logic also monitor the DCR bus for accesses to SDRAM configuration and status registers. If either a PLB or DCR operation targeting the SDRAM controller is detected the SDRAM controller wakes up. The wakeup process results in a two cycle additional latency to the pending operation.

Chapter 15. External Bus Controller

The PPC405CR External Bus controller (EBC) provides direct attachment for most SRAM/Flash type memory and peripheral devices. The interface minimizes the amount of external glue logic needed to communicate with memory and peripheral devices. This reduces the embedded system device count, circuit board area, and cost.

To eliminate off-chip address decoding, the EBC provides eight programmable chip selects that enable system designers to locate memory and peripherals within the PPC405CR memory map. Chip select, data bus, and associated control signal timings are programmable for both single and burst transfers. For peripherals with variable timing requirements the EBC supports device-paced transfers with optional bus-timeout. System design is further simplified through dynamic bus sizing which supports seamlessly attaching 8-, 16-, and 32-bit wide memories and peripherals. Whenever a size mismatch exists between a read or write operation and the externally attached device, the EBC automatically packs or unpacks data as appropriate.

In addition to the peripheral and memory interface, the EBC includes an external bus master (EBM) interface. Using the EBM, external masters arbitrate and gain access to the peripheral interface. Once an external master owns the peripheral interface it can read and write all PLB- and OPB-addressable memory, with the exception of devices controlled by the EBC. SDRAM memory is the typical destination for an EBM transaction. For EBC-attached peripherals and memory, the external master is required to directly control the target.

15.1 Interface Signals

Figure 15-1 on page 15-2 illustrates the signal I/O between the EBC and the external peripheral bus.

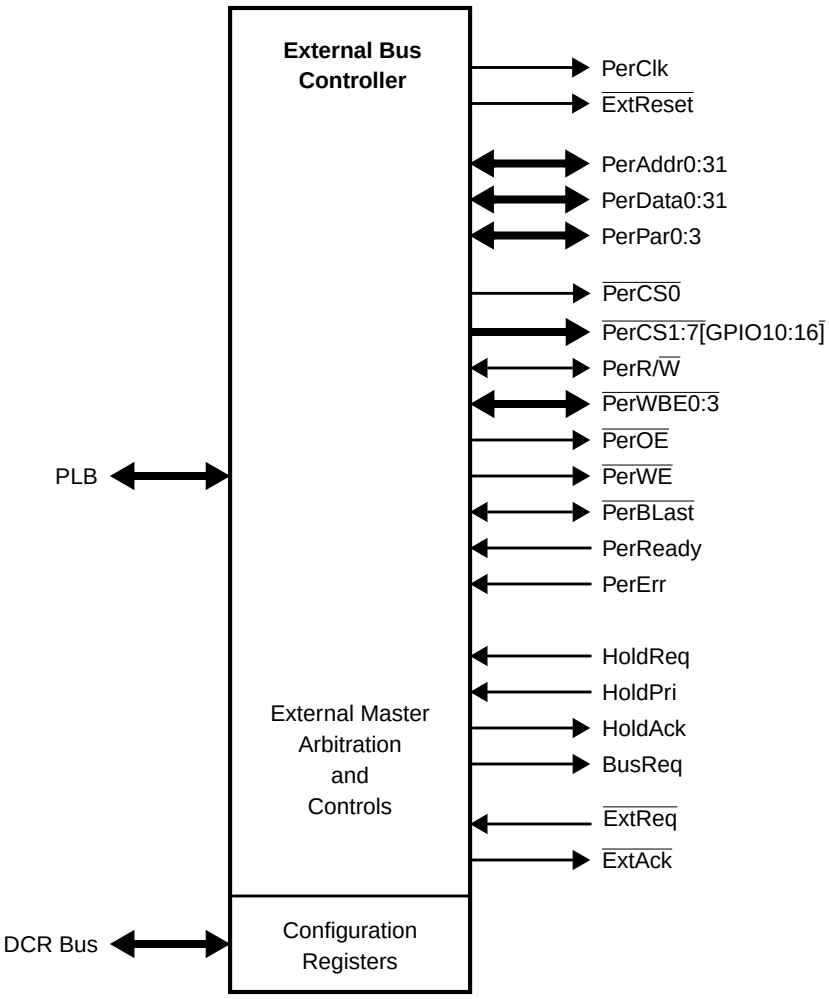


Figure 15-1. External Bus Controller Signals

Table 15-1 describes signal usage and state during and after chip and system resets.

Table 15-1. EBC Signal Usage and State During and After Chip and System Resets

Signal	$\overline{\text{ExtReset}}$ = 0	$\overline{\text{ExtReset}}$ = 1	Usage
PerClk	See Note 1	Toggling	Peripheral bus clock. During an EBC transfer all EBC signal transitions and data sampling occurs synchronous to PerClk.
$\overline{\text{ExtReset}}$	0	1	Peripheral reset for use by slaves and external bus masters.
PerAddr0:31	High-Z	Last Address	Peripheral address bus. PerAddr0 is the most significant bit.
PerData0:31	High-Z	High-Z	Peripheral data bus. PerData0 is the most significant bit.
PerPar0:3	High-Z	High-Z	Peripheral parity bus. The EBC implements odd parity.
$\overline{\text{PerCS0:7}}$	High-Z	1	Chip selects. $\overline{\text{PerCS1:7}}$ are multiplexed with GPIO10:16 and power up as actively driven chip selects. See “Pin Sharing” on page 8-2 for additional details.
PerR/$\overline{\text{W}}$	High-Z	1	Read not write.
$\overline{\text{PerWBE0:3}}$	High-Z	1	Write byte enables or read/write byte enables.

Table 15-1. EBC Signal Usage and State During and After Chip and System Resets

Signal	$\overline{\text{ExtReset}}$ = 0	$\overline{\text{ExtReset}}$ = 1	Usage
$\overline{\text{PerOE}}$	High-Z	1	Output enable.
$\overline{\text{PerWE}}$	High-Z	High-Z	Write enable. $\overline{\text{PerWE}}$ is low whenever any bit in $\overline{\text{PerWBE0:3}}$ is low and PerR/$\overline{\text{W}}$ = 0.
$\overline{\text{PerBLast}}$	High-Z	1	Burst Last. Active during non-burst operations and the last transfer of a burst access.
PerReady	Input	Input	An input to allow external peripherals to perform device-paced transfers.
PerErr	Input	Input	Peripheral data error input.
HoldReq	Input	Input	Hold request, used by an external master to request ownership of the peripheral bus.
HoldPri	Input	Input	Hold priority, used by an external master to indicate the priority in effect for an external master bus tenure.
HoldAck	High-Z	0	Hold acknowledge, informs the external master that it has been granted ownership of the peripheral bus.
BusReq	0	0	Bus request, active when the EBC needs to regain control of the peripheral interface from an external master.
$\overline{\text{ExtReq}}$	Input	Input	External request, used by an external master to indicate that it is prepared to transfer data.
$\overline{\text{ExtAck}}$	High-Z	1	External acknowledge, used by the PPC405CR to indicate that an external master data transfer occurred.

Note 1: During a chip or system reset PerClk begins clocking 64 SysClk cycles prior to the time when the $\overline{\text{ExtReset}}$ output switches from 0 to 1.

15.1.1 Interfacing to Byte, Halfword and Word Devices

Figure 15-2 on page 15-4 illustrates how to interface byte, halfword, and word devices to the peripheral data bus. When devices are connected in this way the EBC supports burst transfers and automatically converts read and write operations to the data width of the external device. As shown in Figure 15-2, halfword devices should not connect to PerAddr31. Similarly, a 32-bit device does not require either PerAddr30 or PerAddr31. Instead, the active byte lanes should be inferred from $\overline{\text{PerWBE0:3}}$, the read/write byte enables.

When a large number of byte and halfword devices are attached to the peripheral data bus, the capacitive loading on byte lane 0 (and byte lane 1, if many halfword devices are used) will be much larger than the loading on byte 3, possibly resulting in unacceptable timing performance on byte 0 or byte 1.

If a bank register is configured as word-wide, then byte-wide devices may be attached to the bus in any byte lane (and accessed using byte loads and stores). Similarly, if a bank register is configured as word-wide, then halfword-wide devices may be attached to the bus in the byte 0/byte 1 lane, or in the

byte 2/byte 3 lane, and accessed using halfword loads and stores. External logic may be required to develop additional control signals if the data bus is utilized in this manner.

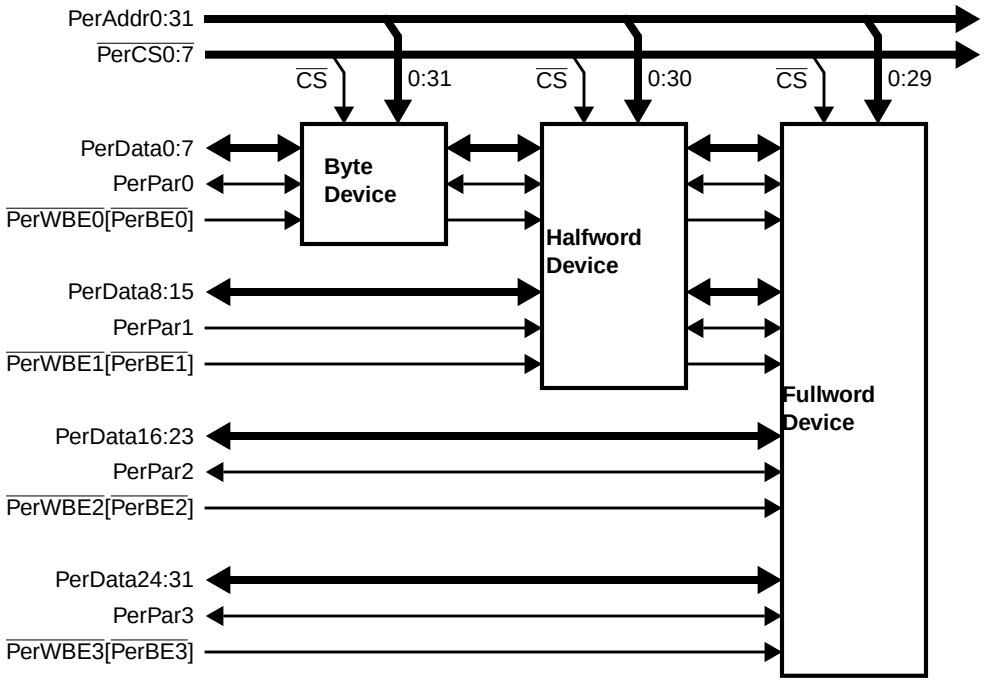


Figure 15-2. Attachment of Devices of Various Widths to the Peripheral Data Bus

15.1.2 Multiplexed I/Os

Seven of the chip select outputs, PerCS1:7, are multiplexed with GPIOs. The multiplexing of these I/Os occurs outside of the EBC logic, see “Pin Sharing” on page 8-2 for additional details. As a result, software can still configure and attempt to use a peripheral memory bank whose output is set up as a GPIO. Doing so causes an EBC transaction without an active chip select and the results are therefore undefined.

In the remainder of this chapter PerCS0:7 is assumed to be configured as EBC outputs.

15.1.3 Driver Enables

As shown in Table 15-2, “Effect of Driver Enable Programming on EBC Signal States,” on page 15-5, the output enables for the peripheral address, data and most of the EBC control signals are configurable. For systems that do not use an external master or where the external master does not directly control devices on the peripheral bus, setting EBC0_CFG[CSTC]=1 eliminates the need for pull-up resistors on PerCS0:7. Pullups are also unnecessary on the remainder of the EBC control signals when EBC0_CFG[EBTC]=1.

Both chip and system resets set EBC0_CFG[EBTC]=1 and EBC0_CFG[CSTC]=1. In most applications, clearing EBC0_CFG[CSTC] is not recommended. If EBC0_CFG[EBTC]=0, EBC control

signals can transition from the active state to high-Z without first being driven inactive. To prevent this, all peripheral banks must be configured with at least one hold cycle, EBC0_BnAP[TH] > 0.

Table 15-2. Effect of Driver Enable Programming on EBC Signal States

EBC Operation	$\overline{\text{ExtReset}}$ **PerClk** $\overline{\text{PerWE}}$ **HoldAck BusReq** $\overline{\text{ExtAck}}$	$\overline{\text{PerCS0:7}}$	**PerAddr0:31 PerR/**$\overline{\text{W}}$ $\overline{\text{PerWBE0:3}}$ $\overline{\text{PerOE}}$ $\overline{\text{PerBLast}}$	**PerData0:31 PerPar0:3**
Reset	High-Z	High-Z	High-Z	High-Z
Idle	Driven	EBC0_CFG[CSTC]	EBC0_CFG[EBTC]	EBC0_CFG[EBTC]
Read	Driven	Driven	Driven	High-Z
Write	Driven	Driven	Driven	Driven
External Master	Driven	EBC0_CFG[CSTC]	High-Z	High-Z

Note 1: If the EBC0_CFG bit is set, the signal is driven to the appropriate state during the indicated EBC operation. Otherwise, the I/O is High-Z.

15.2 Non-Burst Peripheral Bus Transactions

The timing of the $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$, and $\overline{\text{PerWBE0:3}}$ signals is programmable via the Peripheral Bank Access Parameter (EBC0_BnAP) registers. For non-burst transfers, the access parameter registers control the peripheral bus timing as follows:

- $\overline{\text{PerCSn}}$ becomes active 0-3 PerClk cycles (EBC0_BnAP[CSN]) after the address is driven.
- $\overline{\text{PerOE}}$ is driven low 0-3 PerClk cycles (EBC0_BnAP[OEN]) after $\overline{\text{PerCSn}}$ is active.
- $\overline{\text{PerBLast}}$ is active throughout the entire transfer and is driven high during the programmed hold time (EBC0_BnAP[TH]).
- $\overline{\text{PerWBE0:3}}$ can be either write byte enables or read and write enables.

  If EBC0_BnAP[BEM]=0, $\overline{\text{PerWBE0:3}}$ are write byte enables and:

  – $\overline{\text{PerWBE0:3}}$ goes active 0-3 (EBC0_BnAP[WBN]) PerClk cycles after $\overline{\text{PerCSn}}$ becomes active.

  – $\overline{\text{PerWBE0:3}}$ becomes inactive 0-3 (EBC0_BnAP[WBF]) PerClk cycles before $\overline{\text{PerCSn}}$ becomes inactive.

  If EBC0_BnAP[BEM]=1, $\overline{\text{PerWBE0:3}}$ are read/write byte enables and have timing identical to the peripheral address bus. In this case the EBC0_BnAP[WBN] and EBC0_BnAP[WBF] parameters are ignored.
- 1-256 PerClk cycles (EBC0_BnAP[TWT] + 1) after the address became valid:

  – If EBC0_CFG[CSTC]=1 or EBC0_BnAP[TH]>0, $\overline{\text{PerCSn}}$ is driven high.

  – If EBC0_CFG[CSTC]=0 and EBC0_BnAP[TH]=0, $\overline{\text{PerCSn}}$ transitions directly from logic 0 to the high-impedance state.
- The parameters TWT, CSN, OEN, WBN, and WBF in EBC0_BnAP are not independent. For non-burst configured banks it is required that TWT ≥ CSN + MAX(OEN,WBN) + WBF.

- The hold time, EBC0_BnAP[TH], is programmable from 0 to 7 PerClk cycles. During the hold time the peripheral address bus remains driven with the last address and all control signals are actively driven high. If the operation was a write, the peripheral data bus continues driving the last data value.
- There is no guarantee of dead cycles between transfers on the peripheral interface. If there are back-to-back transfers to the same memory bank and the number of hold cycles is programmed to zero (EBC0_BnAP[TH]=0) and EBC0_BnAP[CSN]=0, then:
  - $\overline{\text{PerCSn}}$ may not go inactive between the back-to-back transfers.
  - If EBC0_BnAP[OEN]=0, $\overline{\text{PerOE}}$ may not become inactive between the two transfers.
  - If EBC0_BnAP[WBN]=0 and EBC0_BnAP[WBF]=0, $\overline{\text{PerWBE0:3}}$ may not go inactive between the back-to-back transfers.

15.2.1 Single Read Transfer

Figure 15-3 shows the peripheral interface timing for a single read transfer from a non-burst enabled (EBC0_BnAP[BME]=0) bank. The transaction begins with the address being driven. Since this is a single transfer, $\overline{\text{PerBLast}}$ is also driven active along with the address. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on $\overline{\text{PerWBE0:3}}$. $\overline{\text{PerCSn}}$ then becomes active EBC0_BnAP[CSN] cycles after the address, while $\overline{\text{PerOE}}$ goes low EBC0_BnAP[OEN] cycles after $\overline{\text{PerCSn}}$. The EBC then waits until EBC0_BnAP[TWT]+1 cycles have elapsed since the start of the transaction and then reads the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PAR]=1) the parity is also read at this time. The EBC then drives $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles.

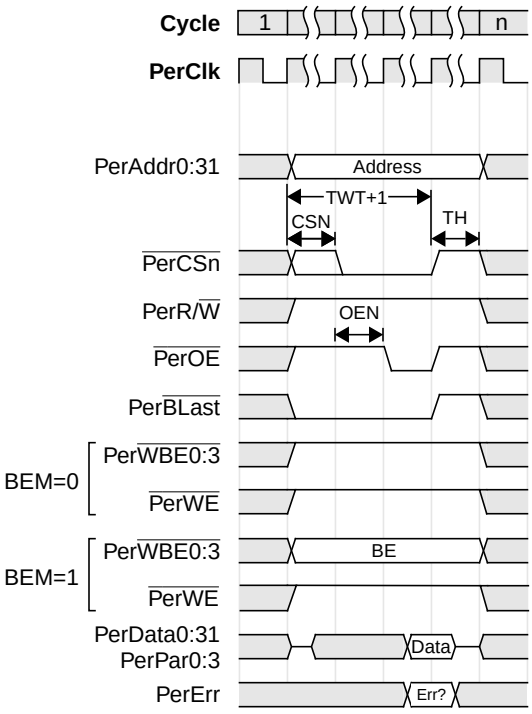


Figure 15-3. Single Read Transfer

15.2.2 Single Write Transfer

Figure 15-4 shows the peripheral interface timing for a single write transfer to a non-burst enabled (EBC0_BnAP[BME]=0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address. At this point the signalling sequence depends on whether or not byte enable mode is enabled for the bank.

- If EBC0_BnAP[BEM]=0, byte enable mode is disabled and the PerWBE0:3 are write byte enables. The appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn. The EBC then waits until (EBC0_BnAP[TWT] - EBC0_BnAP[WBF] + 1) cycles have elapsed since the start of the transaction, then drives all the PerWBE0:3 inactive.
- If EBC0_BnAP[BEM]=1, the PerWBE0:3 lines are byte enables and have the same timing as the peripheral address bus.

After EBC0_BnAP[TWT+1] cycles elapse from the start of transfer, PerCSn and PerBLast are driven high. The EBC then waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

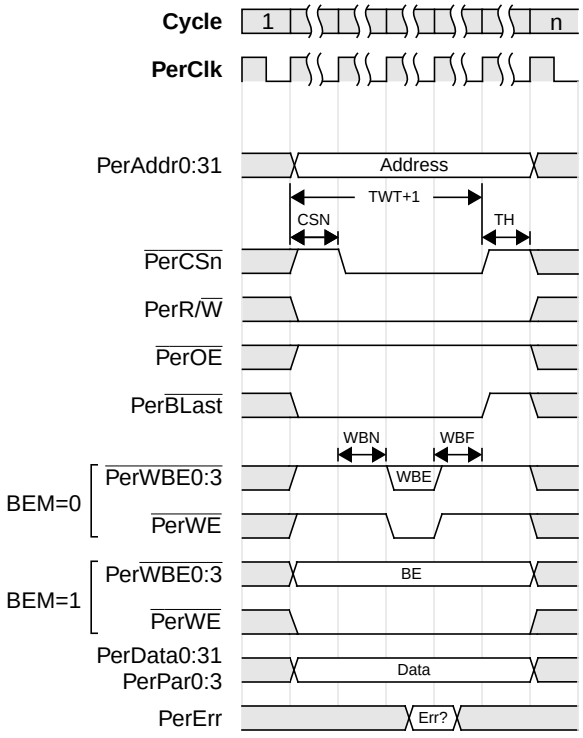


Figure 15-4. Single Write Transfer

15.3 Burst Transactions

Bursting is controlled on a per-bank basis by the Burst Mode Enable bit in the EBC0_BnAP registers. When enabled (EBC0_BnAP[BME]=1) this mode activates bursting for all cache line fills and flushes, PLB burst transfers to the EBC, and all packing and unpacking operations. When bursting is enabled:

- PerCSn becomes active 0-3 (EBC0_BnAP[CSN]) PerClk cycles after the address becomes valid.
- PerCSn is no longer actively driven:
  - 1-32 (EBC0_BnAP[FWT]+1) PerClk cycles after the address becomes valid when a single transfer occurs to a burst-enabled bank.
  - 1-8 (EBC0_BnAP[BWT]+1) PerClk cycles after the last address becomes valid during a burst:
    - If EBC0_CFG[CSTC]=1 or EBC0_BnAP[TH]>0, PerCSn is driven high.
    - If EBC0_CFG[CSTC]=0 and EBC0_BnAP[TH]=0, PerCSn transitions directly from logic 0 to the high-impedance state.
- During read operations PerOE is driven low 0-3 (EBC0_BnAP[OEN]) PerClk cycles after PerCSn is active. PerOE goes inactive when PerCSn goes inactive.
- For bursts, the EBC drives a new address (EBC0_BnAP[FWT]+1) + N*(EBC0_BnAP[BWT]+1) cycles after the start of the transaction, where N = 0, 1, 2, ...
- Addresses during a burst may “wrap.” For example, cache line fills are processed critical word first.
- During write operations, the write data is driven concurrent with each address.
- PerWBE0:3 can be either write byte enables or read and write enables.

  If EBC0_BnAP[BEM]=0, PerWBE0:3 are write byte enables and:
  - For the first transfer of a burst, or a single transfer to a burst enabled bank, the appropriate write byte enables go low 0-3 (EBC0_BnAP[WBN]) cycles after PerCSn becomes active. The EBC then waits until EBC0_BnAP[FWT] - EBC0_BnAP[WBF] + 1 cycles have elapsed since the start of the transaction and drives PerWBE0:3 inactive.
  - The remaining transfers of the burst are similar, except that PerWBE0:3 becomes active EBC0_BnAP[WBN] cycles after each new address is driven on the interface. The PerWBE0:3 remain low for (EBC0_BnAP[BWT] + 1) - EBC0_BnAP[WBN] - EBC0_BnAP[WBF] cycles.

  If EBC0_BnAP[BEM]=1, PerWBE0:3 are byte enables that have timing identical to the peripheral address bus. In this case the EBC0_BnAP[WBN] and EBC0_BnAP[WBF] parameters are ignored.
- PerBLast is active throughout the entire last (or only) transfer of a burst operation and is deactivated during the programmed hold time (EBC0_BnAP[TH]).
- Access bank parameters CSN and OEN apply to the first (or only) transfer of a burst, while WBN and WBF apply to all transfers. It is required that FWT ≥ CSN + MAX(OEN,WBN) + WBF and BWT ≥ WBN+WBF.
- Hold time (EBC0_BnAP[TH]) is programmable from 0 to 7 cycles. During the hold time, the peripheral address bus remains driven and all control signals are driven inactive. If the operation was a write, the peripheral data bus continues driving the write data.
- There is no guarantee of dead cycles between transfers on the peripheral interface. If there are back-to-back transfers to the same memory bank and the number of hold cycles is programmed to zero (EBC0_BnAP[TH]=0) and EBC0_BnAP[CSN]=0, then:

– $\overline{\text{PerCSn}}$ may not go inactive between the back-to-back transfers.

– If EBC0_BnAP[OEN]=0, $\overline{\text{PerOE}}$ may not become inactive between the two transfers.

– If EBC0_BnAP[WBN]=0 and EBC0_BnAP[WBF]=0, $\overline{\text{PerWBE0:3}}$ may not go inactive between the back-to-back transfers.

15.3.1 Burst Read Transfer

Figure 15-5 shows the peripheral interface timing for a burst read transfer from a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on $\overline{\text{PerWBE0:3}}$. $\overline{\text{PerCSn}}$ then becomes active EBC0_BnAP[CSN] cycles after the address, while $\overline{\text{PerOE}}$ goes low EBC0_BnAP[OEN] cycles after $\overline{\text{PerCSn}}$. The EBC then waits until EBC0_BnAP[FWT]+1 cycles have elapsed since the start of the transaction and then reads the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PEN]=1) the parity is also read at this same time.

The next address of the burst is then driven and after EBC0_BnAP[BWT]+1 cycles the EBC performs the next read. The remaining items in the burst are read in the same manner, except that $\overline{\text{PerBLast}}$ is active during the last data element. The EBC then drives $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

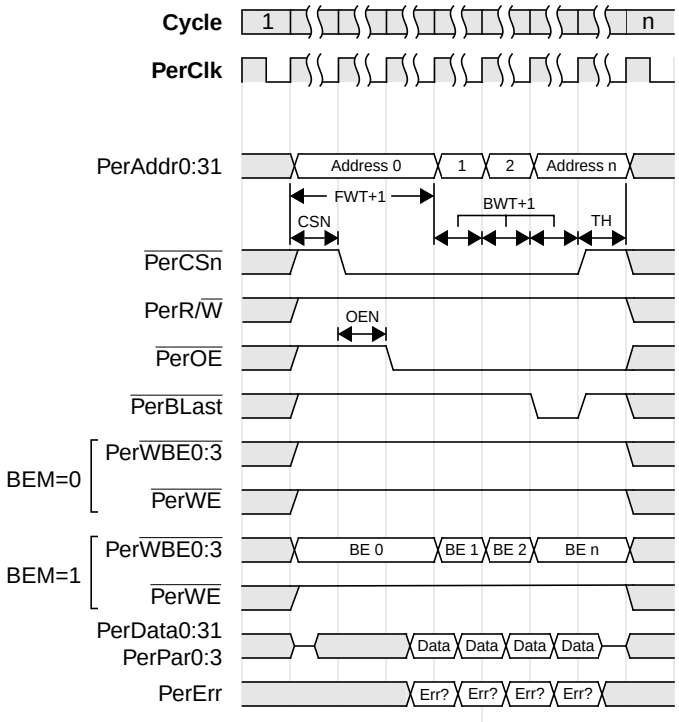


Figure 15-5. Burst Read Transfer

15.3.2 Burst Write Transfer

Figure 15-6 shows the peripheral interface timing for a burst write transfer to burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. At this point the signalling sequence depends on whether byte enable mode is enabled for the bank.

- If EBC0_BnAP[BEM]=0, byte enable mode is disabled and $\overline{\text{PerWBE0:3}}$ are write byte enables. In this case, the appropriate write byte enables go low EBC0_BnAP[WBN] cycles after $\overline{\text{PerCSn}}$. The EBC then waits until (EBC0_BnAP[FWT] +1) - EBC0_BnAP[WBF] cycles have elapsed since the start of the transaction and drives $\overline{\text{PerWBE0:3}}$ inactive. EBC0_BnAP[WBF] cycles are then allowed to elapse after which the address and data are output for the second element in the burst. As shown in Figure 15-6, the EBC transfers the subsequent data items in a similar manner.
- If EBC0_BnAP[BEM]=1, the $\overline{\text{PerWBE0:3}}$ lines are byte enables and have the same timing as the peripheral address bus. In this configuration external logic may be necessary to latch write data at the appropriate times.

$\overline{\text{PerBLast}}$ goes low at the beginning of the last transfer to indicate that the burst is ending. The EBC then drives $\overline{\text{PerCSn}}$ and $\overline{\text{PerBLast}}$ high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

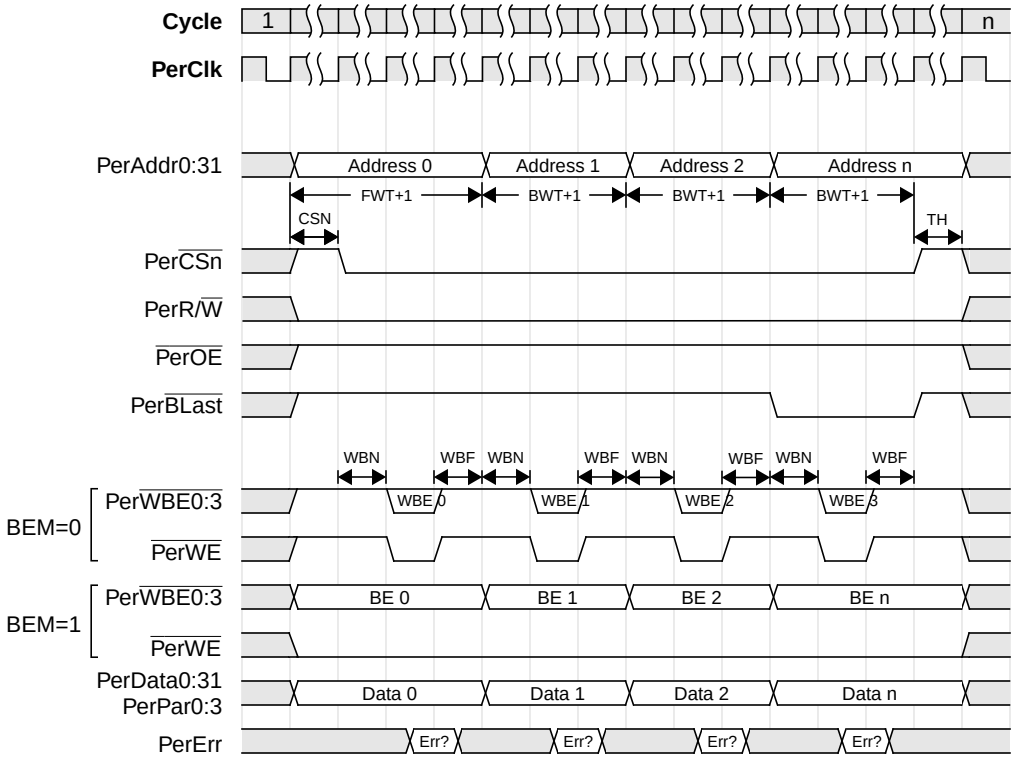


Figure 15-6. Burst Write Transfer

15.4 Device-Paced Transfers

For device-paced transfers, the EBC provides two distinct modes: Sample On Ready enabled and Sample On Ready disabled. The selection of these modes is controlled on a per-bank basis by EBC0_BnAP[SOR]. When Sample On Ready is enabled (EBC0_BnAP[SOR] = 1) data is transferred on the PerClk rising edge where PerReady is sampled active. When Sampling On Ready is disabled (EBC0_BnAP[SOR] = 0), PerReady sampled active causes the data transfer to occur in the next cycle, which results in an additional cycle of wait time.

The ready signal (PerReady) is an input which allows the insertion of externally generated (device-paced) wait states. PerReady is monitored only when EBC0_BnAP[RE]=1.

- For burst disabled banks (EBC0_BnAP[BME] = 0) sampling of the PerReady input starts EBC0_BnAP[TWT] cycles after the beginning of the transfer. Wait states are inserted and sampling continues once per cycle until either PerReady is high when sampled or a timeout occurs.
- For burst enabled banks (EBC0_BnAP[BME] = 1) sampling of the PerReady input starts EBC0_BnAP[FWT] PerClk cycles after the beginning of the first transfer of a burst, and EBC0_BnAP[BWT] cycles after the beginning of subsequent transfers of the burst. Sampling continues once per cycle until either PerReady is sampled high or a timeout occurs.
- When EBC0_BnAP[SOR] = 1 data transfer occurs in the same cycle where PerReady is sampled active. In contrast, if EBC0_BnAP[SOR]=0 the data transfer occurs in the next cycle.
- When EBC0_BnAP[SOR] = 1, if the hold time is set to zero, EBC0_BnAP[TH] = 0, the programmed hold time is ignored and the EBC performs the transaction with one hold cycle.
- When EBC0_BnAP[RE] = 1, the Write Byte Enable Off parameter must be programmed to zero, EBC0_BnAP[WBF] = 0.

The EBC may be programmed to wait only a limited time for PerReady to become active, or it may be programmed for unlimited wait. If EBC0_CFG[PTD] = 1, timeouts are disabled and the EBC waits indefinitely for an active PerReady.

If EBC0_CFG[PTD] = 0, device-paced timeouts are enabled and the EBC only waits for the number of PerClk cycles programmed in EBC0_CFG[RTC]. The timeout counter is reset whenever the peripheral address changes. In this manner each data element within a device-paced burst transaction is treated separately for the purposes of determining whether a timeout error occurs. If PerReady does not become active before the timeout counter reaches the value programmed into EBC0_CFG[RTC], the transfer is aborted and an error is signalled. See “Error Reporting” on page 15-28 for details on how timeout errors are logged.

15.4.1 Device-Paced Single Read Transfer

Figure 15-7 shows the peripheral interface timing for a device-paced single read transfer from a burst disabled (EBC0_BnAP[BME]=0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on PerWBE0:3. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address, while PerOE goes low EBC0_BnAP[OEN] cycles after PerCSn.

The EBC then waits until EBC0_BnAP[TWT]+1 cycles have elapsed since the start of the transaction and then begins sampling PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady to become active. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

Once PerReady is sampled active if Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle. The EBC then samples the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PEN]=1) the parity is also read at this time. The EBC then drives PerCSn, PerOE and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending EBC transfers to occur.

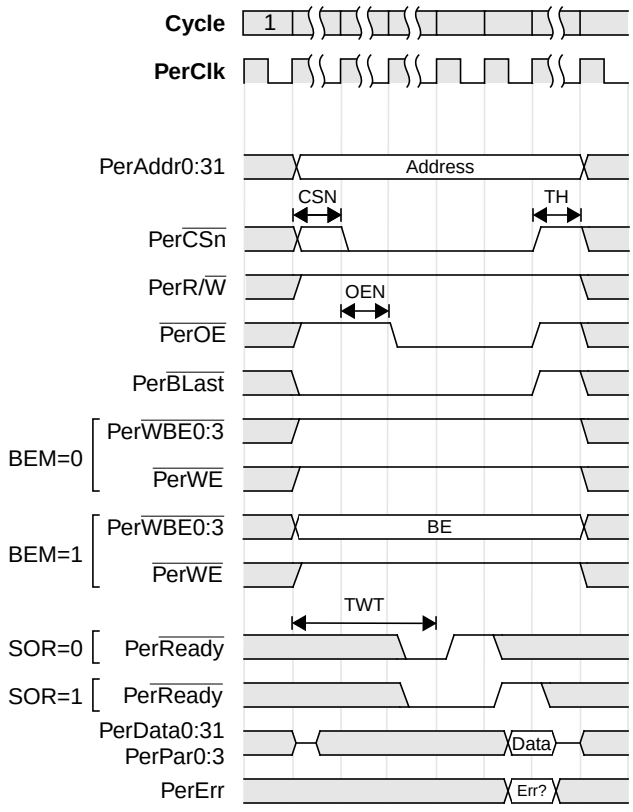


Figure 15-7. Device-Paced Single Read Transfer

15.4.2 Device-Paced Single Write Transfer

Figure 15-8 shows the peripheral interface timing for a device-paced single write transfer from a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. At this point the signalling sequence depends on whether byte enable mode is enabled for the particular bank.

- If EBC0_BnAP[BEM]=0, byte enable mode is disabled and the PerWBE0:3 are write byte enables. The appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn went low. PerWBE0:3 return high on the same PerClk edge that the write data is transferred (see below).
- If EBC0_BnAP[BEM]=1, the PerWBE0:3 lines are byte enables and have the same timing as the peripheral address bus.

The EBC then waits until EBC0_BnAP[TWT]+1 cycles have elapsed since the start of the transaction and then begins sample PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady to become active. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle. At this point, the write transfer occurs and the EBC reads the peripheral error input, PerErr. The EBC then drives PerCSn, PerOE and PerBLast high and waits EBC0_BnAP[TH] cycles.

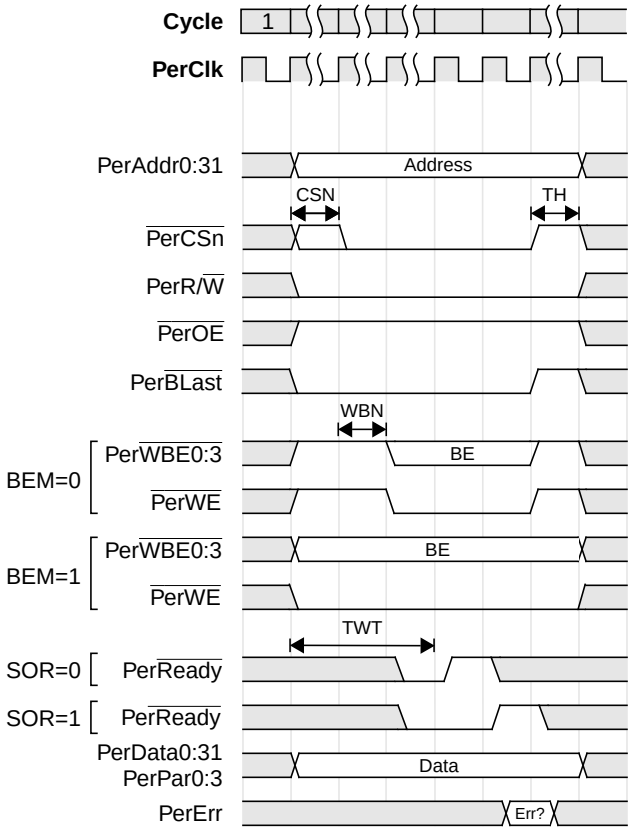


Figure 15-8. Device-Paced Single Write Transfer

15.4.3 Device-Paced Burst Read Transfer

Figure 15-9 shows the peripheral interface timing for a device-paced burst read transfer from a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM]=1) the byte enables are also output concurrently on PerWBE0:3. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address, while PerOE goes low EBC0_BnAP[OEN] cycles after PerCSn. The EBC then waits until EBC0_BnAP[FWT]+1 cycles have elapsed since the start of the transaction and begins sampling PerReady.

If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady to become active. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle before sampling read data. The EBC then reads the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PEN]=1) the parity is also read.

The next address of the burst is then driven and after EBC0_BnAP[BWT]+1 cycles the EBC waits for PerReady as described above. The remaining items in the burst are read in this same manner, except that PerBLast is active during the last data element. The EBC then drives PerCSn, PerOE and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

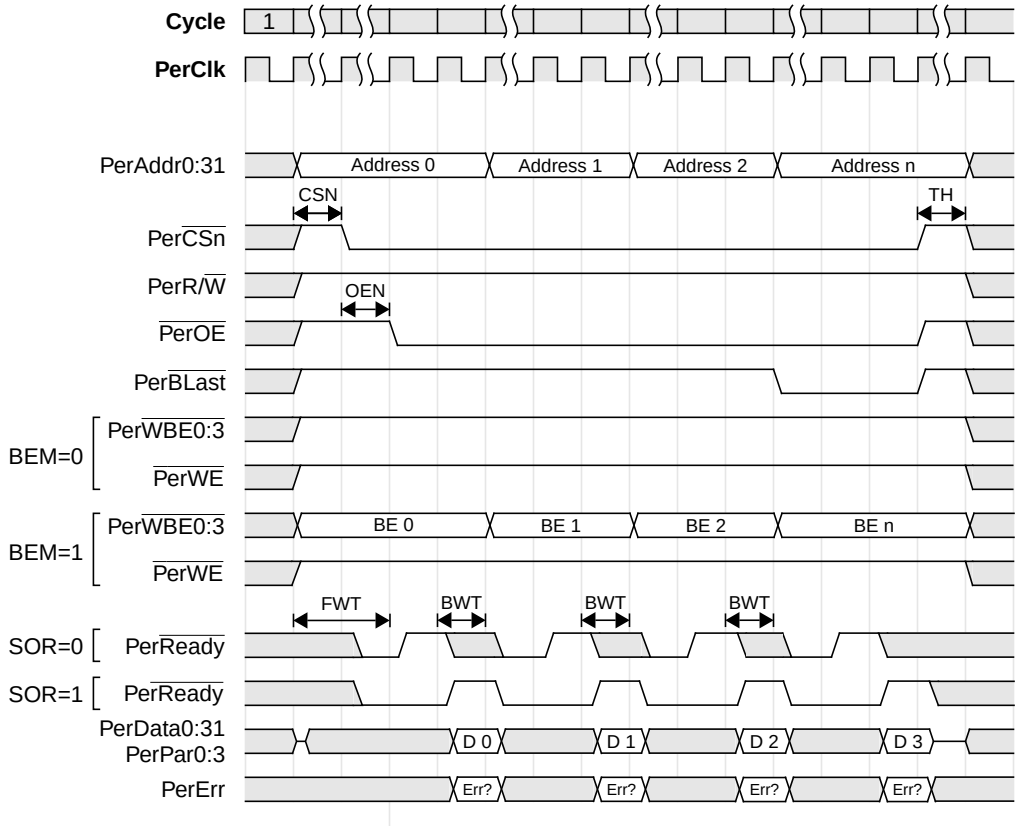


Figure 15-9. Device-Paced Burst Read Transfer

15.4.4 Device-Paced Burst Write Transfer

Figure 15-10 shows the peripheral interface timing for a device-paced burst write transfer to a burst enabled (EBC0_BnAP[BME]=1) bank. The transaction begins with the address being driven. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address. At this point the signalling sequence depends on whether or not byte enable mode is enabled for the bank.

- If byte enable mode is disabled (EBC0_BnAP[BEM]=0) PerWBE0:3 are write byte enables. In this case the appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn becomes active for the first element in a burst and EBC0_BnAP[WBN] cycles after each new address for the remainder of the burst. If EBC0_BnAP[WBN]<>0, PerWBE0:3 is driven inactive on the same PerClk edge that write data is transferred (see below). Otherwise, PerWBE0:3 remains low for all data elements in the burst.
- If EBC0_BnAP[BEM]=1, PerWBE0:3 are byte enables and have the same timing as PerAddr0:31.

The EBC then waits until EBC0_BnAP[FWT]+1 cycles have elapsed since the start of the transaction and begins sampling PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD]=0) the EBC waits indefinitely for PerReady. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR]=0) the EBC waits one more cycle. At this point the write transfer occurs and the EBC reads the peripheral error input, PerErr.

The next address of the burst is then driven and after EBC0_BnAP[BWT]+1 cycles the EBC waits for PerReady as described above. The remaining items in the burst are transferred in this same manner, except that PerBLast is active for the last data element. The EBC then drives PerCSn, PerOE and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

.

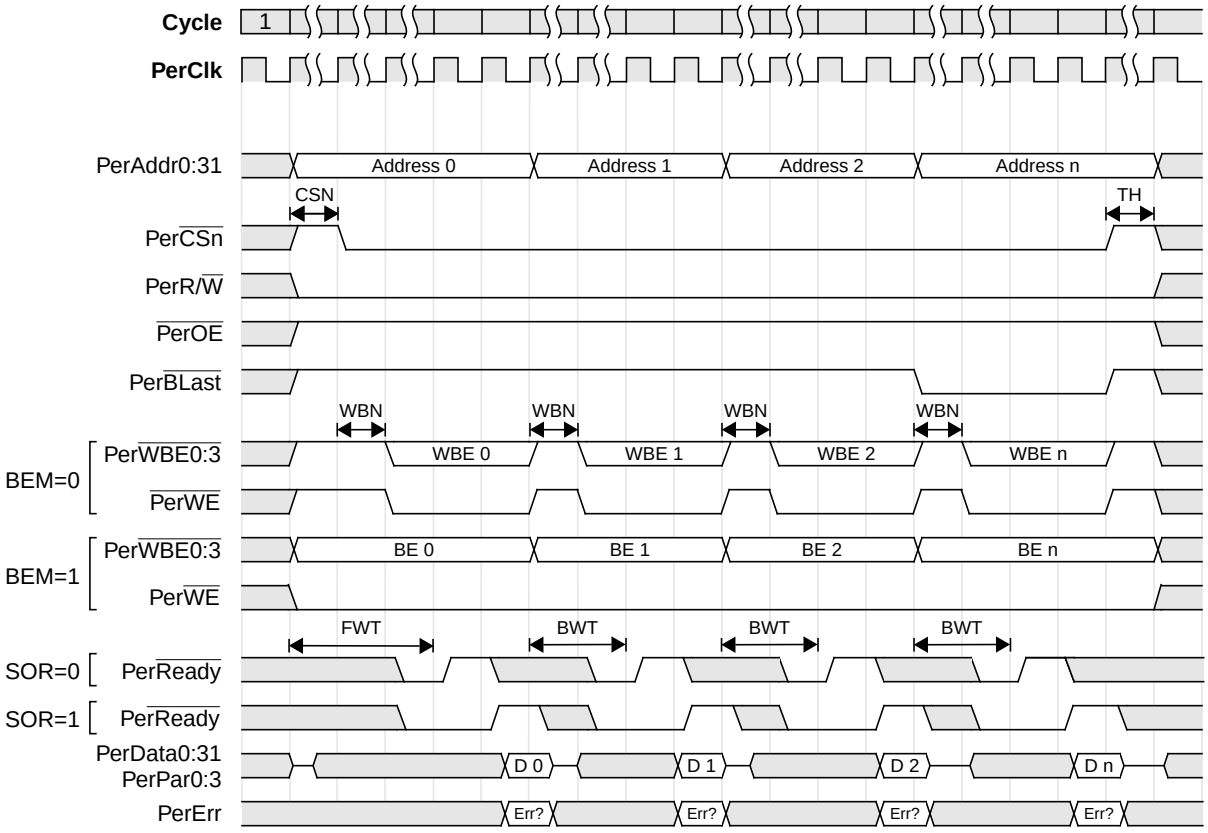


Figure 15-10. Device-Paced Burst Write Transfer

15.5 External Bus Master Interface

The EBC includes an External Bus Master (EBM) interface supporting a shared bus protocol which allows an EBM to gain control of the peripheral bus. Once an external master has been granted access to the peripheral interface it can read and write all PLB- and OPB-addressable memory, with the exception of devices controlled by the EBC. SDRAM memory is the typical destination for an EBM transaction. For EBC-attached peripherals and memory, the external master is required to directly control the target.

Figure 15-11 shows a sample interconnection of the EBC, one SRAM bank, and one external bus master. While only one SRAM bank is shown, the bus master could access all eight of the SRAM banks. Also, with the appropriate arbitration logic, multiple bus masters may be used in a system.

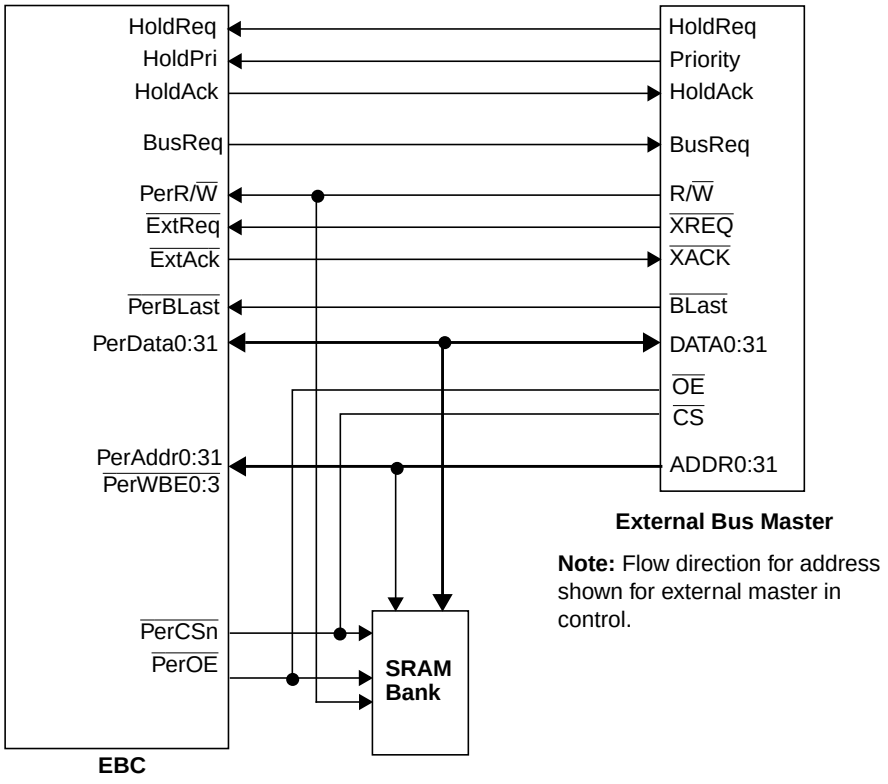


Figure 15-11. Sample External Bus Master System

The EBM interface includes both arbitration and data transfer functions. The arbiter grants the peripheral bus to either the EBC or an external master, while the datapath logic implements an external master to PLB bridge function. This bridge function translates the EBM interface protocol to that required by the on-chip PLB bus.

15.5.1 Arbitration

To gain control of the peripheral bus, the external bus master places an active level on the HoldReq input and drives the HoldPri signal to the desired value (see Figure 15-12 on page 15-20 for an illustrative waveform). The HoldPri input selects the priority of the external master's transactions

relative to other EBC operations. Internal to the PPC405CR all reads and writes that target the EBC are assigned the priority 0b10.

If HoldPri=0 the external master priority is set to EBC0_CFG[EMPL], whereas HoldPri=1 sets the priority to EBC0_CFG[EMPH]. Table 15-3 details the only unique ways of programming the external master priority fields in EBC0_CFG, along with the arbiter's response.

Table 15-3. External Master Arbitration.

EMPL	EMPH	HoldPri	When HoldReq=1 HoldAck Becomes Active	When HoldAck=1 BusReq Goes Active
0b11	0b11	X	After any active EBC operation completes.	Never.
0b00	0b11	0	If no EBC transfers are active or pending.	If an EBC transfer is pending.
		1	After any active EBC operation completes.	Never.
0b00	0b00	X	If no EBC transfers are active or pending.	If an EBC transfer is pending.

When an external bus master requests the bus by driving HoldAck=1, the EBC finishes any transfer in progress (except for bursts) before arbitrating between any pending PLB request and the external master request. If the external master presents a higher priority request during a PLB burst, the burst is terminated. Note that processor cache reads and writes are PLB line operations, not PLB bursts, and are never interrupted.

As detailed in Table 15-3, the arbitration logic drives HoldAck active either when the current EBC transaction completes or when there are no EBC transfers pending. Once the external master has been granted the bus, it may either directly control devices on the peripheral bus or read and write PLB- and OPB-addressable memory. If the EBC detects a pending PLB request when the external master owns the peripheral bus (HoldAck=1), BusReq may be asserted to signal the external master that the PPC405CR wants to regain ownership of the peripheral bus. Table 15-3 lists the cases where a PLB request causes BusReq to go active. To ensure fairness and optimize bus utilization, an external master that receives an active BusReq should complete its current transaction and then relinquish ownership of the peripheral bus by driving HoldReq inactive. This is important since the EBC cannot reclaim ownership of the external bus until the external master negates HoldReq.

Table 15-4, "Signal States During Hold Acknowledge (HoldAck=1)," on page 15-18 details the usage of the EBC I/Os when an external master has been granted ownership of the peripheral bus. The usage statements in this table for signals other than HoldPri, HoldReq, HoldAck and BusReq do not apply if an external master directly controls a device on the peripheral bus.

Table 15-4. Signal States During Hold Acknowledge (HoldAck=1)

Signal Name	State	Usage
HoldPri	Input	Requested priority for external master tenure. HoldPri must not change state when HoldReq=1.
HoldReq	Input (=1)	External master must maintain its request when it owns the bus.
HoldAck	Output (=1)	External master is receiving an active bus grant.
BusReq	Output	See Table 15-3 on Page 15-18.
ExtReq	Input	Indicates the external master is ready to transfer data.
ExtAck	Output	Indicates to the external master that a data transfer occurred.

Table 15-4. Signal States During Hold Acknowledge (HoldAck=1)

Signal Name	State	Usage
PerAddr0:31	Input	Requested address from external master.
$\overline{\text{PerWBE0:3}}$	Input	Selects the requested byte(s) for reads and writes.
PerR/$\overline{\text{W}}$	Input	Determines if the operation is a read or write.
$\overline{\text{PerBLast}}$	Input	Active during single transfers and the last transfer of a burst.
PerData0:31	I/O	Read and write data.
$\overline{\text{PerOE}}$	High-Z	Unused.
$\overline{\text{PerWE}}$	1	Unused.
PerErr	High-Z	Unused.
PerPar0:3	High-Z	Unused.

15.5.2 Transaction Overview

The EBM interface supports direct attachment of 8-, 16-, and 32-bit masters. By programming the width of the external master into EBC0_CFG[EMS] the interface accepts write data and provides read data at the appropriate width for the master. The EBM interface includes a 32-byte data buffer, used for both read and write operations between the EBM and PLB- and OPB-mapped memory locations. While write operations only use the buffer during bursts, all reads prefetch one doubleword and burst reads prefetch EBC0_CFG[BPR] doublewords from the source memory into the buffer. This prefetched data remains in the buffer until either a write operation is performed or a read is requested to a different 32-byte block of memory.

To provide the best possible performance, the EBM interface supports both single and burst transactions. Single read transfers result in the EBM reading and buffering a 64-bit doubleword from the requested memory address. The requested read data is then serviced from within this doubleword. If the next operation on the EBM interface is a read and targets this same doubleword, it is serviced directly from the buffer. Burst reads are similar, except that the EBM prefetches four doublewords beginning with the requested word.

Single write transfers result in a separate PLB transaction for each data item. To improve performance, burst writes are gathered in the 32-byte buffer and forwarded in a single PLB transaction to the target memory.

15.5.3 Single Read and Single Write Transfers

Figure 15-12 illustrates external master bus arbitration along with a single read and signal write transfer. An external master requests ownership of the peripheral bus by driving HoldReq active along with the desired priority on HoldPri. Observe that HoldPri must be held at constant value throughout the entire external master tenure. After two or more PerClk cycles the arbiter will grant the peripheral bus to the external master. The delay from when the external master asserts HoldReq to when HoldAck becomes active is variable and depends on any EBC transaction that may be in progress or pending, the level on HoldPri, and the programming in EBC0_CFG[EMPL] and EBC0_CFG[EMPH].

Once the external master is granted the peripheral bus (HoldAck=1), it may either directly control a device on the EBC or issue read and write transactions to the external master interface. This waveform and the ones that follow apply only to the later case. Additionally, cycles shown with breaks in the clock may not be present or may extend for multiple cycles.

To perform a single read operation the external master must:

- Place the desired address on PerAddr0:31.
- Indicate the requested data byte(s) on $\overline{\text{PerWBE0:3}}$.
- Drive PerR/$\overline{\text{W}}$ high.
- Assert $\overline{\text{PerBLast}}$ to mark this as a single transfer.
- Request the transfer by driving and holding $\overline{\text{ExtReq}}$ low.

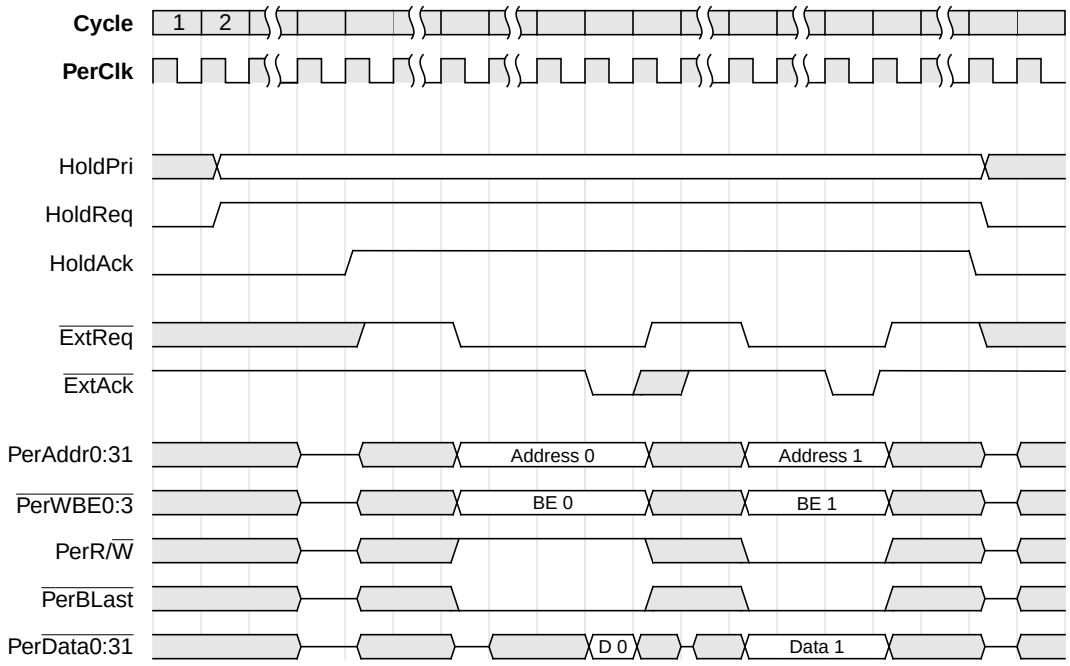


Figure 15-12. External Master Arbitration, Single Read and Single Write

The EBM interface then converts this read request into a PLB transaction and reads the target memory location. $\overline{\text{ExtAck}}$ then goes low when read data is available on PerData. Note that the external master must not remove $\overline{\text{ExtReq}}$ until the cycle after $\overline{\text{ExtAck}}$ becomes active. In addition, $\overline{\text{ExtReq}}$ must be high for at least one cycle between all external master transactions.

Write transfers are similar to reads except that the write data must be provided along with the address and PerR/$\overline{\text{W}}$ is low to indicate a write. As with reads, $\overline{\text{ExtReq}}$ must be held until the cycle after $\overline{\text{ExtAck}}$ is received.

15.5.4 Burst Read Transfer

Burst reads are preferred when accessing sequential addresses as they provide much better performance. Figure 15-13 illustrates an external master burst read transaction. A burst read differs from a single read in that $\overline{\text{PerBLast}}$ is held inactive for all but the last transfer of the burst. In addition, the EBM only requires the address of the first burst element. Following each $\overline{\text{ExtAck}}$ the EBM uses

the size of the master configured in EBC0_CFG[EMS] to increment its internal address counter as appropriate.

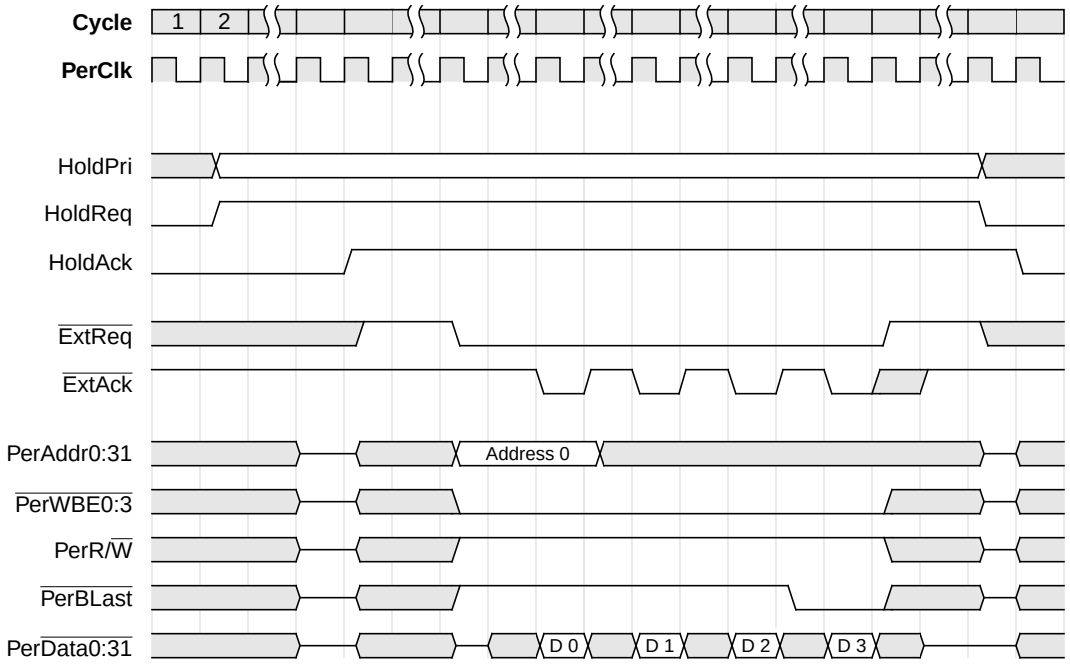


Figure 15-13. External Master Burst Read

15.5.5 Burst Write Transfer

Burst writes are preferred when accessing sequential addresses as they provide much better performance. Figure 15-14 illustrates an external master burst write transaction. A burst write differs from a single write in that PerBLast is held inactive for all but the last transfer of the burst. In addition,

the EBM only requires the address of the first burst element. Following each $\overline{\text{ExtAck}}$ the EBM uses the size of the master in EBC0_CFG[EMS] to increment its internal address counter as appropriate.

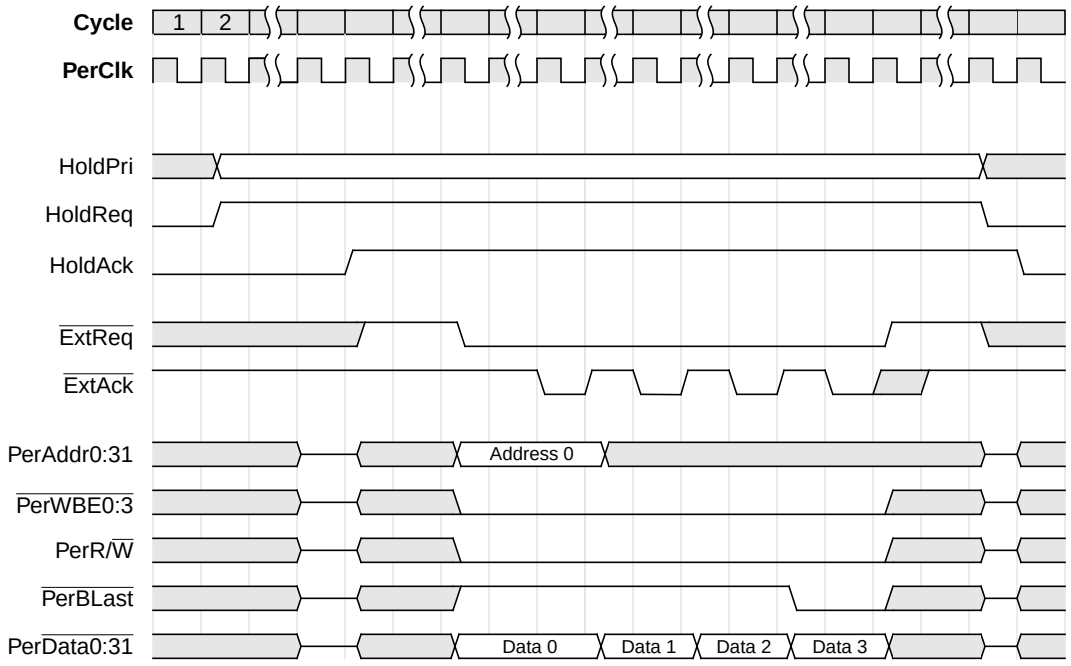


Figure 15-14. External Master Burst Write

15.5.6 External Master Error Interrupts

The EBM can generate an interrupt if a PLB error is encountered while read or writing data. As example, an SDRAM uncorrectable ECC error will cause and EBM interrupt. Please refer to the UIC Chapter for additional information regarding interrupts.

15.6 EBC Registers

All EBC configuration and status registers are accessed using the PowerPC **mtdcr** and **mfdcr** instructions. Access to these registers is performed using an indirect addressing method through the EBC0_CFGADDR and EBC0_CFGDATA registers.

Table 15-5. EBC DCR Addresses

Register	DCR Address	Access	Description
EBC0_CFGADDR	0x012	R/W	External Bus Controller Address Register
EBC0_CFGDATA	0x013	R/W	External Bus Controller Data Register

Table 15-6 lists the indirectly accessed EBC configuration and status registers.

Table 15-6. External Bus Configuration and Status Registers

Mnemonic	Address Offset	Access	Description	Page
EBC0_B0CR–EBC0_B7CR	0x00–0x07	R/W	Peripheral Bank Configuration Registers	15-25
EBC0_B0AP–EBC0_B7AP	0x10–0x17	R/W	Peripheral Bank Access Parameters	15-26
EBC0_BEAR	0x20	R	Peripheral Bus Error Address Register	15-29
EBC0_BESR0	0x21	R/W	Peripheral Bus Error Status Register 0	15-29
EBC0_BESR1	0x22	R/W	Peripheral Bus Error Status Register 1	15-30
EBC0_CFG	0x23	R/W	EBC Configuration Register	15-23

To access one of these registers, software must first write the address offset into the EBC0_CFGADDR register. The target register can then be read or written through the EBC0_CFGDATA DCR address. The following PowerPC code illustrates this procedure by writing the EBC0_B0CR register and then reading back the written value.

```
li        r3,EBC0_B0CR              ! address offset
lis       r4,<config upper>         ! upper 16-bits of configuration data
ori       r4,r4,<config lower>      ! lower 16-bits of configuration data
mtdcr     EBC0_CFGADDR,r3           ! set offset addr
mtdcr     EBC0_CFGDATA,r4           ! write config data
mfdcr     r5,EBC0_CFGDATA           ! read back config data
```

15.6.1 EBC Configuration Register (EBC0_CFG)

The contents of EBC0_CFG are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

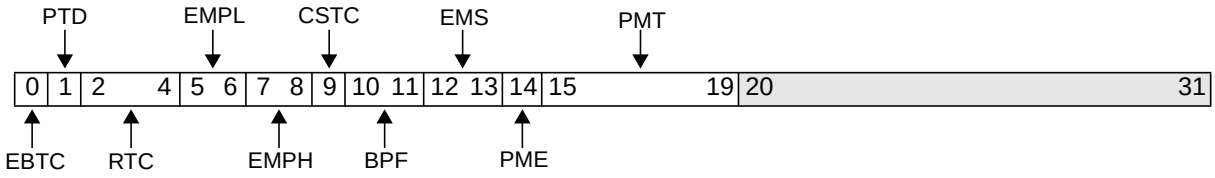


Figure 15-15. EBC Configuration Register (EBC0_CFG)

0	EBTC	External Bus Three-State Control 0 Address, data and control signals are high-Z between EBC transfers. 1 Between EBC transfers the peripheral data bus, address bus and control signals are driven.	Default after reset is EBTC=1. See “Effect of Driver Enable Programming on EBC Signal States” on page 15-5.
1	PTD	Device-Paced Time-out Disable 0 Enabled time-outs 1 Disable time-outs	If PTD=1, the EBC waits indefinitely for assertion of PerReady during device-paced accesses.
2:4	RTC	Ready Timeout Count 000 16 PerClk cycles 001 32 PerClk cycles 010 64 PerClk cycles 011 128 PerClk cycles 100 256 PerClk cycles 101 512 PerClk cycles 110 1024 PerClk cycles 111 2048 PerClk cycles	When PTD=0, the number of cycles from PerAddr0:31 changing until a time-out error occurs.
5:6	EMPL	External Master Priority Low 00 Low 01 Medium low 10 Medium high 11 High	The PLB priority for external master initiated transfers when the external master hold priority input is low (HoldPri=0).
7:8	EMPH	External Master Priority High 00 Low 01 Medium low 10 Medium high 11 High	The PLB priority for external master initiated transfers when the external master hold priority input is high (HoldPri=1).
9	CSTC	Chip Select Three-state Control 0 $\overline{\text{PerCS0:7}}$ are high-Z between EBC transfers and when an external master is active (HoldAck=1) 1 $\overline{\text{PerCS0:7}}$ are always driven.	Default after reset is CSTC=1. See “Effect of Driver Enable Programming on EBC Signal States” on page 15-5.
10:11	BPF	Burst Prefetch 00 Prefetch 1 doubleword 01 Prefetch 2 doublewords 10 Prefetch 4 doublewords 11 Reserved	Controls the amount of data prefetching when the EBC is servicing a PLB burst read. For most applications set this field to 0b00.
12:13	EMS	External Master Size 00 8-bit 01 16-bit 10 32-bit 11 No external master attached	Width of the attached external master.

14	PME	Power Management Enable 0 Disabled 1 Enabled	
15:19	PMT	Power Management Timer 0-31	The EBC makes a sleep request to the Clock and Power Management unit when PME=1 and the EBC has been idle for 32*PMT PerClk cycles.
20:31		Reserved	

15.6.2 Peripheral Bank Configuration Registers (EBC0_BnCR)

These registers must be configured to enable memory in each respective bank. Boot ROM must be attached to bank 0 if installed.

If a boot ROM is present, the bank 0 starting address register is loaded with a value of 0xFFE, and the bank 0 size register is loaded with a value of 0b001 (2MB) immediately following $\overline{\text{SysReset}}$ inactive.

ROM presence and width are controlled by strapping pins in the PPC405CR.

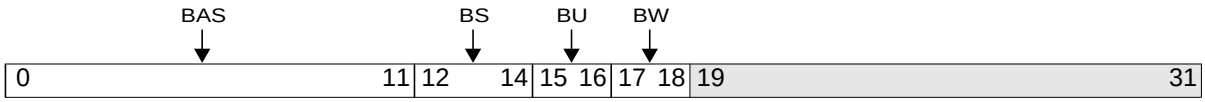


Figure 15-16. Peripheral Bank Configuration Registers (EBC0_BnCR)

0:11	BAS	Base Address Select	Specifies the bank starting address, which must be a multiple of the bank size.
12:14	BS	Bank Size 000 1 MB bank 001 2 MB bank 010 4 MB bank 011 8 MB bank 100 16 MB bank 101 32 MB bank 110 64 MB bank 111 128 MB bank	
15:16	BU	Bank Usage 00 Disabled 01 Read-only 10 Write-only 11 Read/Write	Specifies the type of accesses allowed for the bank. A protect error occurs if a write is attempted to a read-only bank or a read from a write-only bank.
17:18	BW	Bus Width 00 8-bit bus 01 16-bit bus 10 32-bit bus 11 Reserved	
19:31		Reserved	

- **BAS (Base Address Select, bits 0:11)** – Sets the base address for a peripheral device. The bank starting address must be a multiple of the bank size programmed in the BS field. The BAS field is compared to bits 0:11 of the address. If the address is within the range of a BAS field, the associated bank is enabled for the transaction.

  Multiple bank registers may be inadvertently programmed with the same base address or as overlapping banks. An attempt to use such overlapping banks is recorded in EBC0_BESR0 or EBC0_BESR1 as a configuration error and no external access occurs. This error may result in a machine check exception if the requesting master is the CPU. If the error occurred during a DMA access, the DMA may signal an interrupt to the PPC405CR through the UIC.

- **BS (Bank Size, bits 12:14)** – Sets the number of bytes which the bank may access, beginning with the base address set in the BAS field.

- BU (Bank Usage, bits 15:16) – Protects banks of physical devices from read or write accesses.

  When a write access is attempted to an address within the range of the BAS field, and the bank is designated as read-only, a protection error occurs. Also, when a read access is attempted to an address within the range of the BAS field, and the bank is designated as write-only, a protection error occurs. The address of the attempted access is logged in EBC0_BEAR and type of error is logged in either EBC0_BESR0 or EBC0_BESR1.

- BW (Bus Width, bits 17:18) – Controls the width of region accesses. If the BW field is 0b00, the region is configured for an 8-bit data bus; 0b01 indicates a 16-bit data bus and 0b10 indicates a 32-bit data bus. If devices are attached to the data bus as shown in Figure 15.1.1 on page 15-3, the EBC automatically packs read data and unpacks write data when a data transfer size mismatch exists.

15.6.3 Peripheral Bank Access Parameters (EBC0_BnAP)

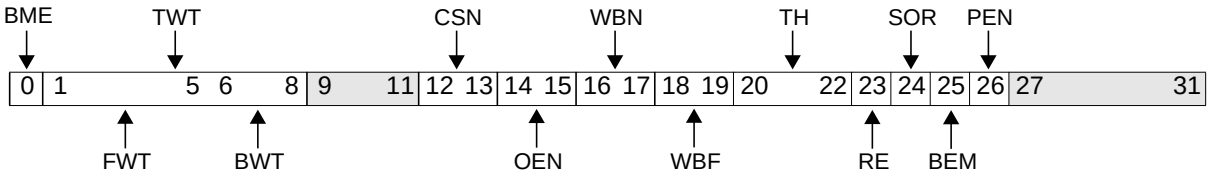


Figure 15-17. Peripheral Bank Access Parameters (EBC0_BnAP)

0	BME	Burst Mode Enable 0 Bursting is disabled 1 Bursting is enabled	
1:8	TWT	Transfer Wait 0-255 PerClk cycles	Wait states on all transfers when BME=0.
1:5	FWT	First Wait 0-31 PerClk cycles	If BME=1, number of wait states on the first transfer of a burst.
6:8	BWT	Burst Wait 0-7 PerClk cycles	If BME=1, number of wait states on non-first transfers of a burst.
9:11		Reserved	
12:13	CSN	Chip Select On Timing 0-3 PerClk cycles	Number of cycles from peripheral address driven to $\overline{\text{PerCSn}}$ low.

14:15	OEN	Output Enable On Timing 0-3 PerClk cycles	Number of cycles from $\overline{\text{PerCSn}}$ low to $\overline{\text{PerOE}}$ low.
16:17	WBN	Write Byte Enable On Timing 0-3 PerClk cycles	If BEM=0, number of cycles from $\overline{\text{PerCSn}}$ low to $\overline{\text{PerWBE0:3}}$ active.
18:19	WBF	Write Byte Enable Off Timing 0-3 PerClk cycles	If BEM=0 and RE=0, number of cycles $\overline{\text{PerWBEn}}$ becomes inactive prior to $\overline{\text{PerCSn}}$ inactive.
20:22	TH	Transfer Hold 0-7 PerClk cycles	Contains the number of hold cycles inserted at the end of a transfer.
23	RE	Ready Enable 0 PerReady is disabled 1 PerReady is enabled	
24	SOR	Sample on Ready 0 Data transfer occurs one PerClk cycle after PerReady is sampled active 1 Data transfer occurs in the same PerClk cycle that PerReady becomes active	
25	BEM	Byte Enable Mode 0 $\overline{\text{PerWBE0:3}}$ are only active for write cycles 1 $\overline{\text{PerWBE0:3}}$ are active for read and write cycles	If BEM=0, $\overline{\text{PerWBE0:3}}$ timing is controlled by WBN and WBF. If BEM=1, $\overline{\text{PerWBE0:3}}$ has the same timing as PerAddr0:31.
26	PEN	Parity Enable 0 Disable parity checking 1 Enable parity checking	The EBC implements odd parity.
27:31		Reserved	

- BME (Burst Mode Enable, bit 0) – Controls bursting for cache line fills and flushes, PLB burst transfers and all packing and unpacking operations. If BME=1, bursting is enabled. When bursting is enabled the parameters Chip Select On (CSN), Output Enable On (OEN), and First Wait (FWT) apply only to the first transfer, while Burst Wait (BWT) and Write Byte Enable On (WBN) apply during all remaining transfers of the burst.
- TWT (Transfer Wait, bits 1:8) – Specifies the number of wait states taken by each transfer to the bank. The number of cycles from address valid to the deassertion of $\overline{\text{PerCSn}}$ is (1 + TWT), where $0 \leq \text{TWT} \leq 255$. This field is used for non-burst transfers (field BME = 0).
- FWT (First Wait, bits 1:5) – Specifies the number of wait states to be taken by the first access to the bank during a burst transfer (field BME = 1). During a burst the number of cycles from the first address valid to the second address is (1 + FWT), where $0 \leq \text{FWT} \leq 31$.
- BWT (Burst Wait, bits 6:8) – Specifies the number of wait states to be taken by accesses beyond the first during a burst transfer (field BME = 1). On burst accesses except for the last, the number of cycles from address valid to the next valid address on each burst access is (1 + BWT), where $0 \leq \text{BWT} \leq 7$. On the last burst access, the number of cycles from address valid to the deassertion of $\overline{\text{PerCSn}}$ is (1 + BWT), where $0 \leq \text{BWT} \leq 7$.
- CSN (Chip Select On Timing, bits 12:13) – Specifies the chip select turn on delay relative to the address. $\overline{\text{PerCSn}}$ may turn on coincident with the address or be delayed by 1, 2, or 3 PerClk cycles.

- OEN (Output Enable On Timing, bits 14:15) – Specifies when the output enable signal, $\overline{\text{PerOE}}$, is asserted for read operations relative to the chip select signal. If 0, $\overline{\text{PerOE}}$ is asserted coincident with the chip select. If 1, 2 or 3, $\overline{\text{PerOE}}$ is delayed by 1, 2, or 3 PerClk cycles.
- WBN (Write Byte Enable On Timing, bits 16:17) – Specifies when the write byte enables, $\overline{\text{PerWBE0:3}}$, are asserted relative to the chip select signal. If 0, then $\overline{\text{PerWBE0:3}}$ turns on coincident with the chip select. If 1, 2, or 3, then $\overline{\text{PerWBE0:3}}$ is delayed 1, 2, or 3 PerClk cycles from the chip select.
- WBF (Write Byte Enable Off Timing, bits 18:19) – Specifies when the write byte enables are deasserted, relative to the deassertion of the chip select signal. If WBF=0, $\overline{\text{PerWBE0:3}}$ goes high coincident with the chip select signal. If WBF is 1, 2, or 3, then $\overline{\text{PerWBE0:3}}$ turns off 1, 2, or 3 PerClk cycles before the turn-off of the chip select signal.

  Programming Note: It is an error to set WBF > BWT. Moreover, for device-paced transfers (EBC0_BnAP[RE]=1) WBF must be set to zero.
- TH (Transfer Hold, bits 20:22) – Specifies the number of PerClk cycles (0 through 7) that the peripheral bus is held idle after the deassertion of $\overline{\text{PerCSn}}$. During these cycles, the address bus and data bus are active and PerR/$\overline{\text{W}}$ is valid. During the hold time, chip select, output enable, and write byte enables are inactive. If Ready Mode is used (RE=1) along with Sample on Ready (SOR=1) TH must be set to at least 1.
- RE (Ready Enable, bit 23) – Controls the use of the PerReady input signal. If RE=0, the PerReady input is ignored and no additional wait states are inserted into bus transactions. If RE=1, the PerReady input is examined after the wait period expires; additional wait states are inserted if the PerReady input is 0. The maximum number of wait states in each transaction is determined by the settings in the Device-Paced Timeout Disable (PTD) and Ready Timeout Counter (RTC) fields in EBC0_CFG. If EBC0_CFG[PTD]=0, the ready timeout function is disabled and the PPC405CR waits indefinitely until PerReady=1. If EBC0_CFG[PTD]=1, the PPC405CR waits the number of cycle indicated by EBC0_CFG[RTC] cycles for PerReady to become active. If PerReady does not become active in the allotted time, the address of the error is logged in EBC0_BEAR and the type of error is captured in either EBC0_BESR0 or EBC0_BESR1.
- **SOR (Sample Ready, bit 24)** – Controls the location of the data transfer cycle with respect to the PerReady input. If SOR=1 the data transfer occurs on the same PerClk edge that PerReady is sampled active, whereas if SOR=0 the data transfer occurs one cycle later.
- **BEM (Byte Enable Mode, bit 25)** – Controls whether $\overline{\text{PerWBE0:3}}$ is active during writes or for both reads and writes.
- **PEN (Parity Enable Mode, bit 26)** – Enables parity generation and checking.

15.7 Error Reporting

The EBC monitors four kinds of errors when performing read and write transfers. Of these four, bank protect and external bus errors are always checked, while timeout and read parity error checking must be enabled via DCR-mapped configuration registers.

- **Protect Error** – Requested read or write operation violates the bank usage programmed in EBC0_BnCR[BU]. For example, write attempt to read-only bank. In all cases, no external bus activity occurs.

- **External Bus Error** – The PerErr input was sampled active during the data transfer cycle of a read or write operation. The associated data is read or written as usual.
- **Timeout Error** – This error is possible during memory operations when both PerReady sampling is enabled, EBC0_BnAP[RE]=1, and device paced timeouts are enabled, EBC0_CFG[PTD]=0. Whenever the peripheral address bus changes the EBC begins counting PerClk cycles. If the count reaches the value represented by EBC0_CFG[RTC] a timeout error occurs. Note that timeout errors are not possible during the peripheral portion of DMA transfers.
- **Parity Error** – Indicates that the parity calculated for the read data did not match the parity read. Parity generation and checking is enabled for memory operations by setting EBC0_BnAP[PEN]=1 and for DMA peripheral transfers by programming DMA0_CRn[PCE]=1.

  When the EBC slave detects one of the above errors it reports the error condition to the PLB master that initiated the transfer. The EBC also logs the type of error into EBC0_BESR0 or EBC0_BESR1 and the address of the error in EBC0_BEAR.

15.7.1 Peripheral Bus Error Address Register (EBC0_BEAR)

The Peripheral Bus Error Address Register (EBC0_BEAR) is a 32-bit register containing the address of the access where a data bus error occurred. The contents of the EBC0_BEAR are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

Precise address capture in the EBC0_BEAR when a parity error occurs only applies to devices with at least one cycle of hold time: EBC0_BnAP[TH] > 0. This is because parity errors are not calculated until the cycle after the data was valid on the external bus, and the EBC0_BEAR is loaded with the address on the bus during that cycle. If the device timings do not include at least one hold cycle, the address may transition to the next address in a burst or the address of a new transaction before the EBC0_BEAR latches the value.

Figure 15-18. Peripheral Bus Error Address Register (EBC0_BEAR)

0:31		Address of Bus Error (asynchronous)

15.7.2 Peripheral Bus Error Status Register 0 (EBC0_BESR0)

The Peripheral Bus Error Status Register 0 (EBC0_BESR0) records the occurrence and type of errors for transactions attempted on behalf of the CPU and External Bus Master. The contents of EBC0_BESR0 are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

It is possible to have both a parity error and a bus error during the same data transfer. If this occurs, the bus error is detected first and EBC0_BESR0 and EBC0_BEAR are updated. In the next cycle the parity error is detected and, if error locking is not enabled, logged in EBC0_BESR0.

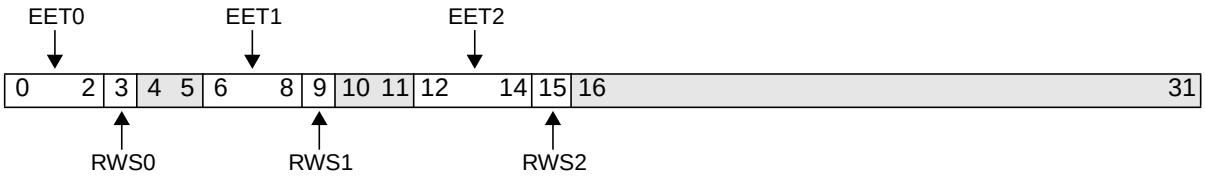


Figure 15-19. Peripheral Bus Error Status Register 0 (EBC0_BESR0)

0:2	EET0	Error type for master 0 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 0 is the processor instruction fetcher.
3	RWS0	Read/write status for master 0 0 Error operation was a write operation 1 Error operation was a read operation	
4:5		Reserved	
6:8	EET1	Error type for master 1 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 1 is the processor data side.
9	RWS1	Read/write status for master 1 0 Error operation was a write operation 1 Error operation was a read operation	
10:11		Reserved	
12:14	EET2	Error type for master 2 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 2 is the external bus master.
15	RWS2	Read/write status for master 2 0 Error operation was a write operation 1 Error operation was a read operation	
16:31		Reserved	

15.7.3 Peripheral Bus Error Status Register 1 (EBC0_BESR1)

The Peripheral Bus Error Status Register 1 (EBC0_BESR1) records the occurrence and type of errors for transactions attempted on behalf of the DMA controller. The contents of EBC0_BESR1 are

accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

It is possible to have both a parity error and a bus error during the same data transfer. If this occurs the bus error is detected first and EBC0_BESR1 and EBC0_BEAR are updated. In the next cycle the parity error is detected and, if error locking is not enabled, logged in EBC0_BESR1.

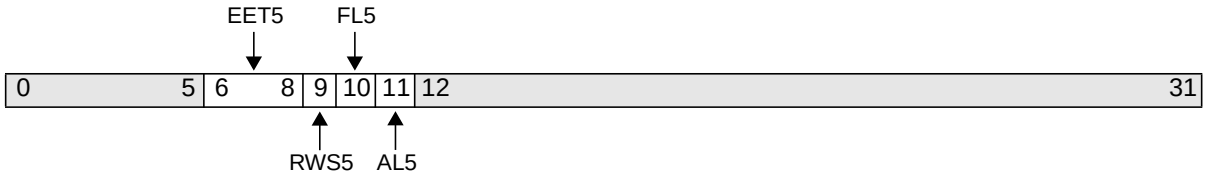


Figure 15-20. Peripheral Bus Error Status Register 1 (EBC0_BESR1)

0:5		Reserved
6:8	EET5	Error type for master 5 — Master 5 is the DMA controller. 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error
9	RWS5	Read/write status for master 5 0 Error operation was a write operation 1 Error operation was a read operation
10	FL5	Field lock for master 5 0 EET5 and RWS5 fields are unlocked 1 EET5 and RWS5 fields are locked
11	AL5	EBC0_BEAR address lock for master 5 0 EBC0_BEAR address unlocked 1 EBC0_BEAR address locked
12:31		Reserved

Chapter 16. Direct Memory Access Controller

The Direct Memory Access (DMA) controller is a Processor Local Bus (PLB) and On-chip Peripheral Bus (OPB) master which supports the autonomous transfer of data between memory and peripherals and from memory-to-memory. The controller provides four DMA channels, each of which has an independent set of configuration registers. Each channel has its own control, source address, destination address, count, and scatter/gather address registers. Once these registers are programmed by the PPC405 processor, the DMA controller performs the requested data transfer without the need for processor intervention.

The four DMA channels also support scatter/gather transfers. During a scatter/gather transfer the configuration registers for a particular DMA channel are automatically loaded from a data structure in memory instead of being individually programmed. Since the scatter/gather address register is updated in this process, the channel can optionally reconfigure itself for another transfer when the current one completes.

As master on both the PLB and OPB the DMA controller can read and write any address accessible by the PPC405 processor. This includes memory and memory-mapped peripherals on the EBC interface and SDRAM memory. The DMA controller can also service DMA peripherals attached to the EBC via the DMAReqn, DMAAckn and EOTn[TCn] I/Os, along with the OPB-attached UART0.

16.1 External Interface Signals

Figure 16-1 illustrates the external I/Os associated with the DMA controller and the EBC I/Os used during external peripheral transfers. External peripheral and EBC device-paced memory transfers request service from the DMA controller by driving a DMA request line (DMAReq0-3) active. For peripheral mode transfers the DMA controller acknowledges the request and transfers data by asserting a DMA acknowledge signal (DMAAck0-3). In contrast, an EBC device-paced memory-to-memory transfer occurs when the chip select ($\overline{\text{PerCSn}}$) associated with the memory location is driven active. The timing of $\overline{\text{PerCSn}}$ and the other EBC I/Os is determined by the Bank Access Parameter Register (EBC0_BnAP) for the particular memory location. See Chapter 15, “External Bus Controller” for details on EBC configuration and timings.

Table 16-1. DMA Controller External I/Os

Signal	Usage
DMAReqn	DMA Request, used to request either a peripheral mode transfer or an EBC device-paced memory-to-memory transfer.
DMAAckn	DMA Acknowledge, instructs an EBC-attached DMA peripheral to transfer data.
EOTn[TCn]	End of Transfer or Terminal Count. Used to stop the channel when programmed as EOT. When configured as TC, goes active when the channel transfer count register (DMA0_CTn) reaches zero.

Note 1: The active level (polarity) of DMAReqn, DMAAckn, and EOTn[TCn] are individually programmable. See “DMA Polarity Configuration Register (DMA0_POL)” on page 16-5.

.

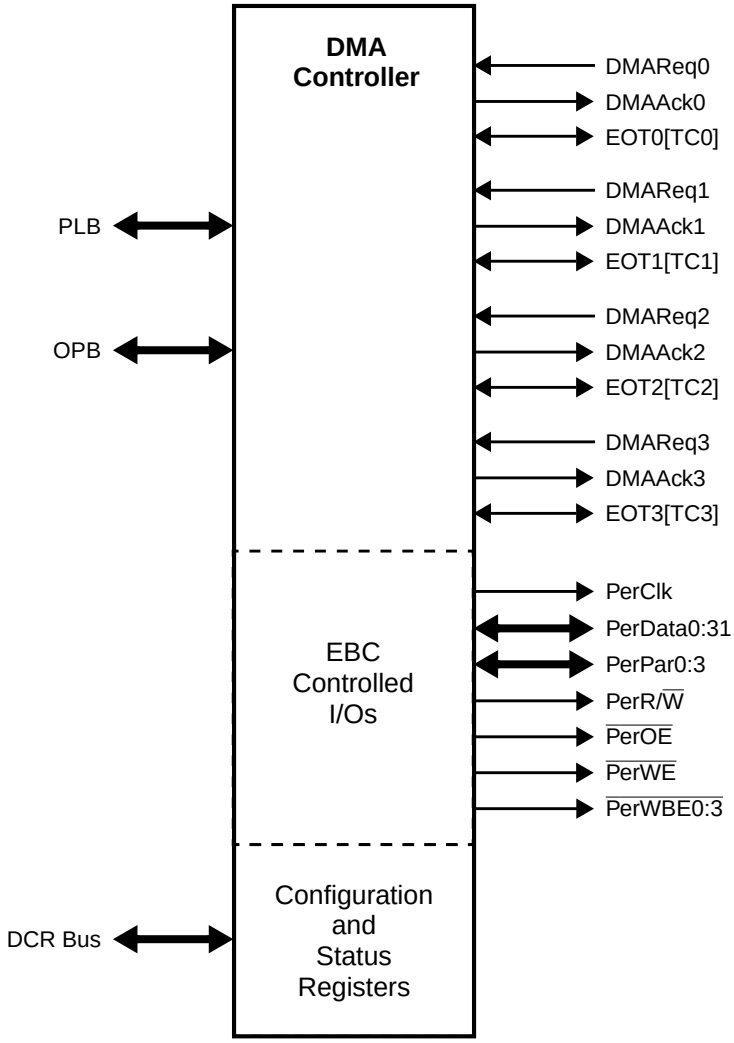


Figure 16-1. DMA Controller External Bus Control Signals

16.2 Functional Overview

As a specialized controller, the DMA unit provides system designers and programmers with a highly efficient method of moving data. During any DMA transfer the controller always buffers data read from the source prior to writing the data to the destination. Since many buses, including the internal PLB and the SDRAM interface, provide substantially better performance when bursting data, the DMA controller includes a 32-byte (4 doubleword) buffer. This buffer is enabled on a per-channel basis by setting DMA0_CRn[BEN] and serves to minimize the number of discrete memory transactions. Each of the four DMA channels is configurable for either peripheral or memory-to-memory transfers.

16.2.1 Peripheral Mode Transfers

Peripherals are either devices attached to the EBC interface via the DMAReqn and DMAAckn lines, or the internal serial port (UART0). Memory is any address accessible from the PLB, including SDRAM

memory, and memory and memory-mapped devices on the peripheral bus. During a peripheral mode transfer the peripheral requests a DMA transfer by asserting a DMA request line. For UART0, this signal is internal to the PPC405CR, while external peripherals use one of the DMAReqn lines. When the requesting channel has the highest priority of any active channel, the requesting device receives a DMA acknowledge. In the case of external peripherals, the appropriate DMAAckn line is driven to the active state, with the timing specified in the DMA Control Register for the channel (DMA0_CRn).

There are two types of peripheral mode transfers: peripheral-to-memory and memory-to-peripheral. A peripheral-to-memory transfer reads data from a DMA device, while a memory-to-peripheral transfer writes data. In both cases, the peripheral interface never bursts and data is transferred at the width of the peripheral. If the DMA buffer is disabled for the active channel (DMA0_CRn[BEN]=0), each peripheral transfer causes a corresponding memory operation.

When buffering is enabled during a peripheral-to-memory transfer, data is collected until the 32-byte buffer is full, the peripheral deasserts DMAReqn, a higher priority DMA request becomes pending, or the channel completes. The buffer contents are then written to the target memory as efficiently as possible. If the initial programming of the channel's destination address register (DMA0_DAn) is 32-byte aligned, the buffer is emptied in one burst operation to the target memory.

Memory-to-peripheral transfers differ since the amount of data that will be requested by the peripheral is unknown. If the DMA buffer is disabled (DMA0_CRn[BEN]=0) a discrete source memory read occurs for each element in the DMA transfer. Since this is inefficient, the buffer should only be disabled for low data rate transfers or when the source memory is FIFO-like, and reads are therefore destructive. When the 32-byte buffer is enabled the controller uses the setting in DMA0_CRn[PF] to prefetch 1, 2, or 4 64-bit doublewords from the source memory. The DMA controller provides data from the buffer until the peripheral deasserts its request, the channel is interrupted by one of higher priority, or the transfer completes. Whenever any of these conditions occurs any unused data in the DMA buffer is discarded.

16.2.2 Memory-to-Memory Transfers

The DMA controller can perform either device-paced (hardware-initiated) or software-initiated memory-to-memory transfers. Device-paced memory transfers function identically to peripheral mode transfers, except that a chip select ($\overline{\text{PerCSn}}$) serves as the DMA acknowledge instead of a DMAAckn output. As with peripheral mode transfers, bursts never occur on the device-paced side of the transaction.

Software-initiated memory-to-memory transfers between memories with fixed timings provide the best overall performance. During a software-initiated transfer the DMA controller knows the exact amount of data to be transferred. As a result, when the 32-byte DMA buffer is enabled (DMA0_CRn[BEN]=1) the controller uses bursts as much as possible. To ensure the highest bandwidth, source and destination addresses should be aligned on 32-byte boundaries.

There are three cases that limit the ability to burst during software-initiated memory-to-memory transfers. Bursting is not possible from the source memory when the source address increment is zero (DMA0_CRn[SAI]=0). Similarly, the DMA controller does not burst to the destination when the destination address increment is zero (DMA0_CRn[DAI]=0). Finally, the DMA controller cannot burst to or from any address that maps to a device-paced (EBC0_BnAP[RE]=1) EBC chip select ($\overline{\text{PerCSn}}$).

16.2.3 Scatter/Gather Transfers

Each of the four DMA channels supports scatter/gather transfers. This scatter/gather capability allows the chaining of multiple DMA controller operations within a channel. During a normal DMA operation software must program the control, source address, destination address, and count registers for each transfer. Scatter/gather transfers differ in that these registers are automatically loaded from a linked list data structure in system memory. When a channel completes one transfer the DMA controller loads the next set of configuration values into the channel's registers and the channel continues with the new programmings.

16.3 Configuration and Status Registers

Table 16-2 on page 16-4 lists the DMA configuration and status registers, each of which is accessed using the PowerPC **mtdcr** and **mfdcr** instructions. As example, the following PowerPC assembly code writes the control register for DMA channel 0 and then reads the DMA status register:

```
#define DMA0_CR0  0x100
#define DMA0_SR   0x120

    mtdcr     DMA0_CR0,r3              ! write r3 to channel 0 control register
    mfdcr     r4,DMA0_SR               ! read contents of status register into r4
```

The DMA configuration and status registers are readable at any time. However, since each register read requires a separate operation, it is not possible to guarantee that the values read from multiple registers correspond to a state that ever existed in the DMA controller. To illustrate, consider software that reads the destination address for channel 0 (DMA0_DA0) and then the count for channel 0 (DMA0_CT0). If the DMA controller updates the count between these two operations, the values read differ from what is expected.

While reads can occur at any time, software must not write the configuration registers for any channel that is currently enabled (DMA0_CRn[CE]=1). The only exception is that a channel may be disabled by reading the channel control register, clearing the channel enable bit, and then writing the new value to the control register. Once a channel is disabled, all of its configuration registers may be reprogrammed as desired.

Table 16-2. DMA Controller Configuration and Status Registers

Mnemonic	DCR Address	Access	Description	Page
DMA0_CR0	0x100	R/W	DMA Channel Control Register 0	16-8
DMA0_CT0	0x101	R/W	DMA Count Register 0	16-11
DMA0_DA0	0x102	R/W	DMA Destination Address Register 0	16-10
DMA0_SA0	0x103	R/W	DMA Source Address Register 0	16-10
DMA0_SG0	0x104	R/W	DMA Scatter/Gather Descriptor Address Register 0	16-12
DMA0_CR1	0x108	R/W	DMA Channel Control Register 1	16-8
DMA0_CT1	0x109	R/W	DMA Count Register 1	16-8
DMA0_DA1	0x10A	R/W	DMA Destination Address Register 1	16-10
DMA0_SA1	0x10B	R/W	DMA Source Address Register 1	16-10
DMA0_SG1	0x10C	R/W	DMA Scatter/Gather Descriptor Address Register 1	16-12
DMA0_CR2	0x110	R/W	DMA Channel Control Register 2	16-8
DMA0_CT2	0x111	R/W	DMA Count Register 2	16-11

Table 16-2. DMA Controller Configuration and Status Registers (continued)

Mnemonic	DCR Address	Access	Description	Page
DMA0_DA2	0x112	R/W	DMA Destination Address Register 2	16-10
DMA0_SA2	0x113	R/W	DMA Source Address Register 2	16-10
DMA0_SG2	0x114	R/W	DMA Scatter/Gather Descriptor Address Register 2	16-12
DMA0_CR3	0x118	R/W	DMA Channel Control Register 3	16-8
DMA0_CT3	0x119	R/W	DMA Count Register 3	16-11
DMA0_DA3	0x11A	R/W	DMA Destination Address Register 3	16-10
DMA0_SA3	0x11B	R/W	DMA Source Address Register 3	16-10
DMA0_SG3	0x11C	R/W	DMA Scatter/Gather Descriptor Address Register 3	16-12
DMA0_SR	0x120	R/Clear	DMA Status Register	16-7
DMA0_SGC	0x123	R/W	DMA Scatter/Gather Command Register	16-12
DMA0_SLP	0x125	R/W	DMA Sleep Mode Register	16-6
DMA0_POL	0x126	R/W	DMA Polarity Configuration Register	16-5

16.3.1 DMA Polarity Configuration Register (DMA0_POL)

The Polarity Configuration Register (DMA0_POL) is used to set the polarity (active state) of the external DMA I/O signals: DMAReqn, DMAAckn, and EOTn[TCn]. As shown in Figure 16-2, if a bit in DMA0_POL is zero, the corresponding signal is active high, otherwise the signal is active low.

Whenever any of the EOT polarities are changed (DMA0_POL[EnP), software must subsequently clear the corresponding EOT status bits in the DMA Status Register (DMA0_SR[TS0:3]) prior to enabling the associated DMA channel. This is necessary to prevent a channel from being disabled because of an incorrect EOT status stored in the status bits.

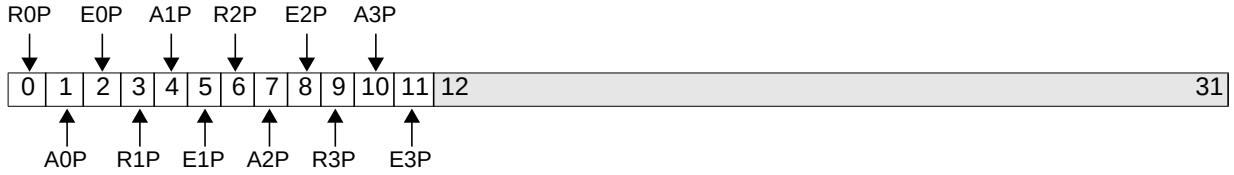


Figure 16-2. DMA Polarity Configuration Register (DMA0_POL)

0	R0P	DMAReq0 Polarity 0 DMAReq0 is active high 1 DMAReq0 is active low
1	A0P	DMAAck0 Polarity 0 DMAAck0 is active high 1 DMAAck0 is active low
2	E0P	EOT0[TC0] Polarity 0 EOT0[TC0] is active high 1 EOT0[TC0] is active low
3	R1P	DMAReq1 Polarity 0 DMAReq1 is active high 1 DMAReq1 is active low
4	A1P	DMAAck1 Polarity 0 DMAAck1 is active high 1 DMAAck1 is active low

5	E1P	EOT1[TC1] Polarity 0 EOT1[TC1] is active high 1 EOT1[TC1] is active low
6	R2P	DMAReq2 Polarity 0 DMAReq2 is active high 1 DMAReq2 is active low
7	A2P	DMAAck2 Polarity 0 DMAAck2 is active high 1 DMAAck2 is active low
8	E2P	EOT2[TC2] Polarity 0 EOT2[TC2] is active high 1 EOT2[TC2] is active low
9	R3P	DMAReq3 Polarity 0 DMAReq3 is active high 1 DMAReq3 is active low
10	A3P	DMAAck3 Polarity 0 DMAAck3 is active high 1 DMAAck3 is active low
11	E3P	EOT3[TC3] Polarity 0 EOT3[TC3] is active high 1 EOT3[TC3] is active low
12:31		Reserved

16.3.2 DMA Sleep Mode Register (DMA0_SLP)

The Sleep Mode Register (DMA0_SLP) enables the DMA controller to enter sleep (low-power) mode and programs the number of PLB clock cycles to wait when the controller is idle before going to sleep. The DMA controller only goes to sleep when no DMA channels are enabled and no configuration or status (DCR) register operations are in progress. Reading or writing any of the DMA control or status register awakens the DMA controller.

To enable sleep mode set DMA0_SLP[SME] and CPM0_ER[DMA]. When sleep mode is enabled and the DMA controller becomes idle the 10-bit idle timer begins counting down from the programmed value. Only the upper 5 bits of the idle counter are programmable; the lower 5 bits are hardcoded to 0b11111. Therefore, the minimum granularity of the idle timer is 32 PLB clock cycles. When the counter reaches zero, the controller is placed in sleep mode.

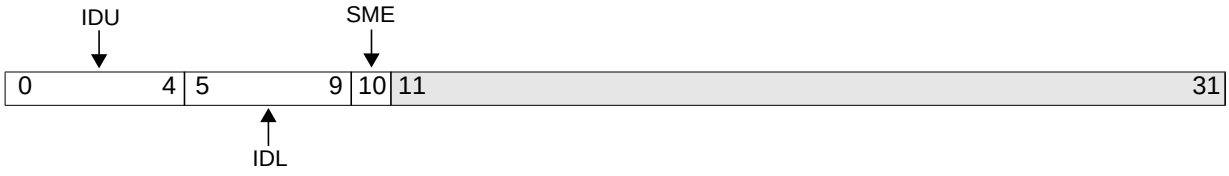


Figure 16-3. DMA Sleep Mode Register (DMA0_SLP)

0:4	IDU	Idle Timer Upper 0-31	Upper 5-bits of the idle timer.

5:9	IDL	Idle Timer Lower Hardcoded to 0b11111	Lower 5-bit portion of the idle timer. Writing this field has no effect.
10	SME	Sleep Mode Enable 0 Sleep disabled 1 Sleep enabled	If SME=1, also set CPM0_ER[DMA] to enable the Clock and Power Management macro to put the DMA controller to sleep.
11:31		Reserved	

16.3.3 DMA Status Register (DMA0_SR)

As shown in Figure 16-4, the DMA Status Register (DMA0_SR) provides status information for each of the DMA channels. Bits in DMA0_SR are set in hardware, and can be either read or cleared by software. Clearing is performed by writing a word to DMA0_SR containing a 1 in any bit position to be cleared and 0 in all other bit positions.

The terminal count status (DMA0_SR[CSn]), end of transfer status (DMA0_SR[TSn]) and error status (DMA0_SR[RIn]) must be cleared for a DMA channel to operate. If a scatter/gather operation generates an interrupt for any of the above conditions, the channel pauses until software clears the associated status field(s) in DMA0_SR.

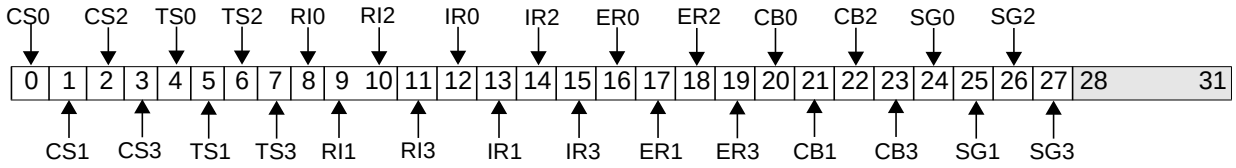


Figure 16-4. DMA Status Register (DMA0_SR)

0:3	CS[0:3]	Channel 0-3 Terminal Count Status 0 Terminal count has not occurred 1 Terminal count has been reached	Set when the transfer count reaches 0.
4:7	TS[0:3]	Channel 0-3 End of Transfer Status 0 End of transfer has not been requested 1 End of transfer has been requested	Only valid for channels with DMA0_CRn[ETD]=0.
8:11	RI[0:3]	Channel 0-3 Error Status 0 No error 1 Error occurred	See “Errors” on page 16-14 for more information.
12:15	IR[0:3]	Internal DMA Request 0 No internal DMA request pending 1 Internal DMA request is pending	
16:19	ER[0:3]	External DMA Request 0 No external DMA request pending 1 External DMA request is pending	
20:23	CB[0:3]	Channel Busy 0 Channel is idle 1 Channel currently active	

24:27	SG[0:3]	Scatter/Gather Status 0 No scatter/gather operation in progress 1 Scatter/gather operation in progress
28:31		Reserved

16.3.4 DMA Channel Control Registers (DMA0_CR0–DMA0_CR3)

The DMA Channel Control Registers (DMA0_CR0-DMA0_CR3) are used to configure and enable their respective DMA channels. Before a DMA channel can transfer data, the channel control, source address, destination address, and transfer count registers must be programmed. If a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1) the DMA channel control register is automatically loaded from memory. For additional details see “Scatter/Gather Transfers” on page 16-15.

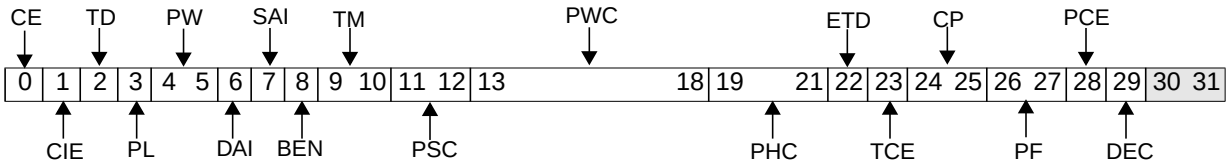


Figure 16-5. DMA Channel Control Registers (DMA0_CR0-DMA0_CR3)

0	CE	Channel Enable 0 Channel is disabled 1 Channel is enabled	This field is automatically cleared when the transfer completes or an error occurs.
1	CIE	Channel Interrupt Enable 0 Disable interrupts from this channel 1 Enable interrupts from this channel	When enabled, interrupts are generated for terminal count, end of transfer, and errors conditions. See “DMA Interrupts” on page 16-15.
2	TD	In peripheral mode: 0 Transfers are from memory-to-peripheral 1 Transfers are from peripheral-to-memory In device-paced memory-to-memory mode: 0 Peripheral is at the destination address 1 Peripheral is at the source address	TD is not used (don’t care) for software-initiated memory-to-memory transfers.
3	PL	Peripheral Location 0 External peripheral (EBC) bus 1 OPB (UART0)	
4:5	PW	Peripheral Width/Memory alignment 00 Byte (8 bits) 01 Halfword (16 bits) 10 Word (32 bits) 11 Doubleword (64 bits) memory-to-memory transfers only	Transfer width equals peripheral width for peripherals.

6	DAI	Destination Address Increment 0 Do not increment destination address 1 After each data transfer increment the destination address by: 1, if the transfer width is a byte (8 bits) 2, if the transfer width is a halfword (16 bits) 4, if the transfer width is a word (32 bits) 8, if the transfer width is a doubleword (64 bit)	
7	SAI	Source Address Increment 0 Do not increment source address 1 After each data transfer increment the source address by: 1, if the transfer width is a byte (8 bits) 2, if the transfer width is a halfword (16 bits) 4, if the transfer width is a word (32 bits) 8, if the transfer width is a doubleword (64 bit)	
8	BEN	Buffer Enable 0 Disable DMA 32-byte buffer 1 Enable DMA 32-byte buffer	If BEN=0 discrete read and write operations occur for each data transfer.
9:10	TM	Transfer mode 00 Peripheral 01 Reserved 10 Software-initiated memory-to-memory 11 Device-paced memory-to-memory	
11:12	PSC	Peripheral Setup Cycles 0-3	Number of PerClk cycles that the EBC peripheral bus is idle from the last peripheral bus transaction until DMAAckn is active. Used only for the peripheral side of peripheral mode transfers.
13:18	PWC	Peripheral Wait Cycles 0-63	DMAAckn remains active for PWC + 1 PerClk cycles. Used only for the peripheral side of peripheral mode transfers.
19:21	PHC	Peripheral Hold Cycles 0-7	The number of PerClk cycles from the time DMAAckn becomes inactive until the peripheral bus is available for the next bus access. Used only for the peripheral side of peripheral mode transfers.
22	ETD	End-of-Transfer/Terminal Count (EOTn[TCn]) Pin Direction 0 EOTn[TCn] is an EOT input 1 EOTn[TCn] is a TC output	ETD must be set to 1 if the channel is configured for software-initiated memory-to-memory transfers.
23	TCE	Terminal Count (TC) Enable 0 Channel does not stop when TC is reached 1 Channel stops when TC is reached	If TCE=1, it is required that ETD=1.
24:25	CP	Channel Priority 00 Low priority 01 Medium low priority 10 Medium high priority 11 High priority	Actively requesting channels of the same priority are prioritorized by channel number; channel 0 has the highest priority. See “Channel Priorities” on page 16-13 for more information.

26:27	PF	Memory Read Prefetch Transfer 00 Prefetch 1 doubleword 01 Prefetch 2 doublewords 10 Prefetch 4 doublewords 11 Reserved	Used only during memory-to-peripheral and deviced-paced memory-to-memory transfers. To enable prefetching it is required that BEN=1.
28	PCE	Parity Check Enable 0 Disable parity checking 1 Enable parity checking	Enables parity checking for peripheral mode transfers. See “Data Parity During DMA Peripheral Transfers” on page 16-13.
29	DEC	Address Decrement 0 SAI and DAI fields control memory address incrementing. 1 After each data transfer the memory address is decremented by the transfer width.	If DEC=1, it is required that BEN=0. This field is valid only for peripheral mode transfers (TM=00).
30:31		Reserved	

16.3.5 DMA Source Address Registers (DMA0_SA0–DMA0_SA3)

The DMA Source Address Registers (DMA0_SA0–DMA0_SA3) contain the source address for memory-to-memory and memory-to-peripheral transfers. If a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1) the source address register is automatically loaded from memory. For additional details see “Scatter/Gather Transfers” on page 16-15.

The source address must be aligned at the transfer width programmed in DMA0_CRn[TW], otherwise the error bit (DMA0_SR[RIn]) is set for the channel and no transfer occurs. If the source address increment bit in the channel’s control register is set (DMA0_CRn[SAI]) the address is incremented by the transfer width after each data transfer. In contrast, if the channel is performing a memory-to-peripheral transfer and the address decrement bit is set (DMA0_CRn[DEC]=1), the address is decremented by the transfer width after each transfer.

Figure 16-6. DMA Source Address Registers (DMA0_SA0–DMA0_SA3)

0:31		Source address for memory-to-memory and memory-to-peripheral transfers.

16.3.6 DMA Destination Address Registers (DMA0_DA0–DMA0_DA3)

The DMA Destination Address Registers (DMA0_DA0-DMA0_DA3) contain the destination address for memory-to-memory and peripheral-to-memory transfers. When a DMA channel is configured for

scatter/gather transfers (DMA_SGC[SSGn]=1) the destination address register is automatically loaded from memory. For additional details see “Scatter/Gather Transfers” on page 16-15.

The destination address must be aligned at the transfer width programmed in DMA0_CRn[TW], otherwise the error bit (DMA0_SR[RIn]) is set for the channel and no transfer occurs. If the destination address increment bit in the channel’s control register is set (DMA0_CRn[DAI]) the address is incremented by the transfer width after each data transfer. However, if the channel is performing a peripheral-to-memory transfer and the address decrement bit is set (DMA0_CRn[DEC]=1), the destination address is decremented by the transfer width after each transfer.

Figure 16-7. DMA Destination Address Registers (DMA0_DA0-DMA0_DA3)

0:31		Destination address for memory-to-memory and peripheral-to-memory transfers.

16.3.7 DMA Count Registers (DMA0_CT0–DMA0_CT3)

The DMA Count Registers (DMA0_CT0-DMA0_CT3) contain the number of transfers left in the DMA transaction for their respective channels when EOTn[TCn] is programmed as a terminal count output. When EOTn[TCn] is programmed as an end-of-transfer input (DMA0_CRn[ETD]=1), DMA0_CTn continues to count down past zero until EOTn is asserted.

If a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1) the count register is automatically loaded from memory. For additional details see “Scatter/Gather Transfers” on page 16-15.

The value in the DMA count register is interpreted as the number of **transfers** of the width specified in DMA0_CRn[PW], **not** the total number of bytes. The maximum number of transfers is 64K, and each transfer can be either 1, 2, 4, or 8 bytes as programmed in DMA0_CR[PW]. The maximum count of 64K transfers is programmed by writing zero to DMA0_CTn.

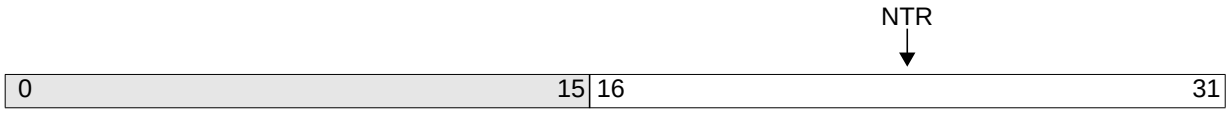


Figure 16-8. DMA Count Registers (DMA0_CT0-DMA0_CT3)

0:15		Reserved
16:31	NTR	Number of transfers remaining

16.3.8 DMA Scatter/Gather Descriptor Address Registers (DMA0_SG0–DMA0_SG3)

When a DMA channel is setup for scatter/gather transfers (DMA_SGC[SSGn]=1), the Scatter/Gather Descriptor Address Register (DMA0_SGn) contains the memory address of the next scatter/gather descriptor table. Prior to starting a scatter/gather transfer, software must write the address of the channel's descriptor table to DMA0_SGn. Once the scatter/gather transfer starts, DMA0_SGn is automatically updated from the descriptor table. For additional details see "Scatter/Gather Transfers" on page 16-15.

Figure 16-9. DMA Scatter/Gather Descriptor Address Registers (DMA0_SG0–DMA0_SG3)

0:31		Address of next scatter/gather descriptor table.

16.3.9 DMA Scatter/Gather Command Register (DMA0_SGC)

The DMA Scatter/Gather Command Register (DMA0_SGC) is a 32-bit register, of which 8-bits are implemented. Bits 0:3 are the Start Scatter/Gather Enable bits for channels 0 to 3, and bits 16:19 are the corresponding Enable Mask bits for the Start Scatter/Gather Enable bits. Setting a Start Scatter/Gather Enable bit causes the selected channel to begin a scatter/gather operation, while writing a 0 stops the Scatter/Gather operation. To start or stop a specific Scatter/Gather channel, the corresponding Enable Mask bit must be set to 1; otherwise, the register holds the previous value. Note that halting a scatter/gather transfer does not stop the transfer currently in progress.

Upon completion of a scatter/gather sequence of transfers the DMA controller clears DMA0_SGC[SSGn].

If an error occurs when the DMA controller is reading the Scatter/Gather descriptor table, DMA0_SGC[SSGn] is cleared for the affected channel, and the channel's error status bit (DMA0_SR[RIn]) is set.

For additional details see "Scatter/Gather Transfers" on page 16-15.

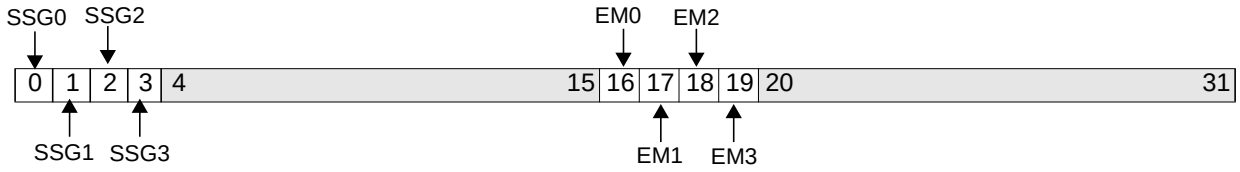


Figure 16-10. DMA Scatter/Gather Command Register (DMA0_SGC)

0:3	SSG[0:3]	Start Scatter/Gather for channels 0-3. 0 Scatter/gather support is disabled 1 Scatter/gather support is enabled	To start a scatter/gather operation for channel n, EM[n] must also be set.
4:15		Reserved	
16:19	EM[0:3]	Enable Mask for channels 0-3. 0 Writes to SSG[n] are ignored 1 Allow writing to SSG[n]	To write SSG[n], EM[n] must be set. Otherwise, writing SSG[n] has no effect.
20:31		Reserved	

16.4 Channel Priorities

The priority of DMA transfers is controlled on a per-channel basis by the channel priority field in the channel control register. Table 16-3 shows the different priority settings for DMA0_CRn[PW].

Table 16-3. DMA Transfer Priorities

DMA0_CRn[CP]	Priority Level
0b00	Low
0b01	Medium Low
0b10	Medium High
0b11	High

These priorities serve two purposes. First, the DMA controller arbitrates among all actively requesting channels and selects the highest priority one for service. If multiple channels request at the same priority, the arbiter selects the lowest numbered channel for service. Secondly, DMA0_CRn[CP] determines the priority of the internal PLB transactions that the DMA controller uses to read and write data.

16.5 Data Parity During DMA Peripheral Transfers

The DMA controller works in conjunction with the peripheral bus controller (EBC) to generate and check parity during peripheral mode DMA transfers. When DMA0_CRn[PCE]=1 parity checking is enabled for peripheral mode transfers on channel n.

During memory-to-memory transfers any data error checking and/or correction is dependent on the configuration of the relevant memory controller. For example, if ECC is enabled on the SDRAM controller, only uncorrectable errors are reported to the DMA controller. Similarly, an EBC memory bank with parity enabled will report an error if a parity error is encountered during a memory read

operation. See “Peripheral Bank Access Parameters (EBC0_BnAP)” on page 15-26 for more information on peripheral bus parity checking.

16.6 Errors

The DMA controller detects and reports three types of errors: address alignment, PLB timeout and slave errors. The DMA controller reports errors through the channel error status bit in the DMA status register (DMA0_SR[RIn]). If the error status bit for a channel is set, the channel enable bit (DMA0_CRn[CE]) is cleared, disabling the channel. An interrupt signal is also presented to the interrupt controller if DMA0_CRn[CIE] is set. See “DMA Interrupts” on page 16-15 for more information on interrupt processing.

When the DMA controller has multiple channels active, an error may be reported on the current channel which was in actuality caused by a previously active channel. This causes the current channel to have its error status bit set. Therefore, for deterministic error analysis with multiple DMA channels active the PLB slave bus controller’s error status registers (the bus error address register in particular), must be queried to isolate the actual channel which encountered the error. In any case, the channel causing the errors will eventually cause all active channels, including itself, to be disabled.

16.6.1 Address Alignment Error

The source address (DMA0_SAn) and destination address (DMA0_DAn) registers must be aligned to the programmed transfer width (DMA0_CRn[TW]).The address alignment rules are outlined in Table 16-4. In addition, when a channel is configured for scatter/gather transfers, the scatter/gather table must be word-aligned. If the source, destination and scatter/gather address registers are not appropriately aligned an error occurs immediately after the channel is enabled.

Table 16-4. Address Alignment Requirements

DMA0_CRn[PW] Setting	Required Alignment for: Source Address Register (DMA0_SAn) and Destination Address (DMA0_DAn)
0b00	Byte (8-bit)
0b01	Halfword (16-bit)
0b10	Word (32-bit)
0b11	Doubleword (64-bit)

16.6.2 PLB Timeout

The DMA controller uses PLB operations to read and write memory. A PLB timeout results if the DMA controller attempts to access a non-existent memory location. This will occur if the source, destination or scatter/gather address registers do not map to valid memory locations.

16.6.3 Slave Errors

If the DMA controller detects an error from a PLB slave, it finishes any active read/write pair transfer on the channel and then reports an error. An SDRAM uncorrectable ECC error and an EBC bank protection error are examples of PLB slave errors.

16.7 DMA Interrupts

Each DMA channel can generate interrupts for end of transfer, terminal count and error conditions. Interrupts from a particular DMA channel are enabled by setting the channel enable bit in the channel's control register (DMA0_CRn[CIE]=1). When an interrupt occurs for a given channel, the DMA controller sends a signal to the Universal Interrupt Controller. For the PPC405CR's CPU to take an exception, interrupts from the particular DMA channel must be enabled in the interrupt controller's interrupt enable register (UIC0_ER). Also, the CPU's machine state register's interrupt enable bit must be enabled for the appropriate interrupt type (critical or non-critical), MSR[EE,CE]. See Chapter 9, "Interrupt Controller Operations" for more information on interrupt controller processing.

For DMA channels with interrupts enabled (DMA0_CRn[CIE]=1) and not performing a scatter/gather transfer an interrupt is generated while any of the following are true:

DMA0_CRn[TCE]=1 and DMA0_SR[CSn]=1
DMA0_SR[TSn]=1
DMA0_SR[RIn]=1

When a channel is performing a scatter/gather transfer, interrupt generation is further qualified by the TCI, ETI and ERI bits loaded from the descriptor table (see Table 16-5, "Scatter/Gather Descriptor Table," on page 16-15). Any of the following conditions cause an interrupt during a scatter/gather transfer when interrupts are enabled for the channel (DMA0_CRn[CIE=1]):

TCI=1 and DMA0_CRn[TCE]=1 and DMA0_SR[CSn]=1
ETI=1 and DMA0_SR[TSn]=1
ERI=1 and DMA0_SR[RIn]=1

For both normal DMA and scatter/gather transfers the interrupt remains active until the appropriate bits are cleared in the DMA Status Register (DMA0_SR). In addition, interrupts from a channel performing a scatter/gather transfer cause the channel to pause until the interrupt is cleared.

16.8 Scatter/Gather Transfers

With a normal DMA transfer it is necessary to program a channel's control, source, destination, and count registers for each transfer. The scatter/gather capability of the DMA controller provides a more efficient solution for applications that require multiple transactions on a single DMA channel. Instead of individually programming a channel's registers, software creates a set (linked list) of descriptor tables in system memory. Table 16-5 illustrates the required table format.

Table 16-5. Scatter/Gather Descriptor Table

Memory Address	Byte 0 (MSB)						Byte 1	Byte 2	Byte 3 (LSB)
x (word aligned)	DMA Channel Control Word								
x + 4	Source Address								
x + 8	Destination Address								
x + 12	LK		TCI	ETI	ERI			Count	
x + 16	Next Scatter/Gather Descriptor Table Address								

Table 16-6 details the usage of the bit fields in the scatter/gather table.

Table 16-6. Bit Fields in the Scatter/Gather Descriptor Table

Bit	Mnemonic	Description
0	LK	Link 0 This is the last descriptor. 1 Fetch next descriptor from address DMA0_SGn when the channel completes
2	TCI	Enable Terminal Count Interrupt 0 Do not interrupt when terminal count occurs 1 Allow an interrupt when terminal count occurs
3	ETI	Enable End of Transfer Interrupt 0 Do not interrupt when end of transfer occurs 1 Allow an interrupt when end of transfer occurs.
4	ERI	Enable Error Interrupt 0 Do not interrupt if an error occurs 1 Allow an interrupt if an error occurs

To configure a channel for a scatter/gather transfer the DMA Scatter/Gather Descriptor Address Register (DMA0_SGn) for the channel is set to the address of the first descriptor table, which must be word-aligned. To begin the scatter/gather transfer, software then writes a start scatter/gather and enable mask to the Scatter/Gather Command Register (DMA0_SGC). The DMA controller then reads the descriptor table at address DMA0_SGn and updates the DMA controller registers as shown in Table 16-7. Upon receiving the data from the scatter/gather descriptor table, the channel's terminal count status bit (DMA0_SR[TCn]) is automatically cleared.

Table 16-7. DMA Registers Loaded from Scatter/Gather Descriptor Table

Descriptor Table Entry	Register Loaded
Channel Control Word	DMA0_CRn
Source Address	DMA0_SAn
Destination Address	DMA0_DAn
Count	DMA0_CTn
Next Descriptor Address	DMA0_SGn

After loading the channel's registers from the descriptor table, the transfer functions as a normal non-scatter/gather operation.

If the channel control word loaded from the descriptor table enables interrupts for the channel (DMA0_CRn[CIE]=1), the TCI, ETI, and ERI bits further qualify the generation of interrupts. See "DMA Interrupts" on page 16-15 for more information on scatter/gather interrupts.

If the LK (link) bit was not set the scatter/gather process stops when the current transfer completes. Otherwise, the DMA controller reads the descriptor table at address DMA0_SGn and the process repeats.

16.9 Programming the DMA Controller

Before the DMA controller can transfer data it must be configured, both globally and on a per-channel basis. Global settings include the DMA Polarity Register (DMA0_POL) and DMA Sleep Mode Register (DMA0_SLP). For most applications, these registers should be configured when the DMA

controller is first initialized. To prevent spurious activity resulting from changing the active level for DMAReqn, DMAAckn, or EOTn[TCn], a channel's configuration in the Polarity Register should not be altered when the channel is enabled (DMA0_CRn[CE]=1).

Each channel has a Control (DMA0_CRn), Source Address (DMA0_SAn), Destination Address (DMA0_DAn), Count (DMA0_CTn), and Scatter/Gather Descriptor Address (DMA0_SGn) register. The type of DMA transfer determines which of these registers must be programmed and what causes the channel to start. In all cases, the terminal count (CSn), end of transfer (TSn) and error status (RIn) bits in the DMA Status Register (DMA0_SR) must be cleared or the channel will not start.

The programming information that follows assumes that the DMA controller is operating in non-scatter/gather mode. To use scatter/gather transfers the channel configuration data must be written into a set of descriptor tables in system memory. See "Scatter/Gather Transfers" on page 16-15 for additional details.

16.9.1 Peripheral Mode Transfers

DMA peripherals are either devices attached to the EBC interface via the DMAReqn and DMAAckn lines, or the internal serial port (UART0). During a peripheral mode transfer an external peripheral asserts DMAReqn to request a DMA transfer. For metastability protection DMAReqn is double latched in the DMA upon assertion, and sampled with a single latch on the deassertion.

Timings on the memory-access portion of peripheral mode DMA transfers are governed by the configuration of the associated memory controller. In contrast, timing during the peripheral portion of the transfer is controlled by the PSC, PWC and PHC fields in the DMA Channel Control Registers. The effect of these parameters on peripheral timings is illustrated in Figure 16-11 on page 16-18 and

Figure 16-12 on page 16-19. Although shown as active high, the polarity (active state) of DMAReqn, DMAAckn, and EOTn[TCn] are programmable via the DMA Polarity Register (DMA0_POL).

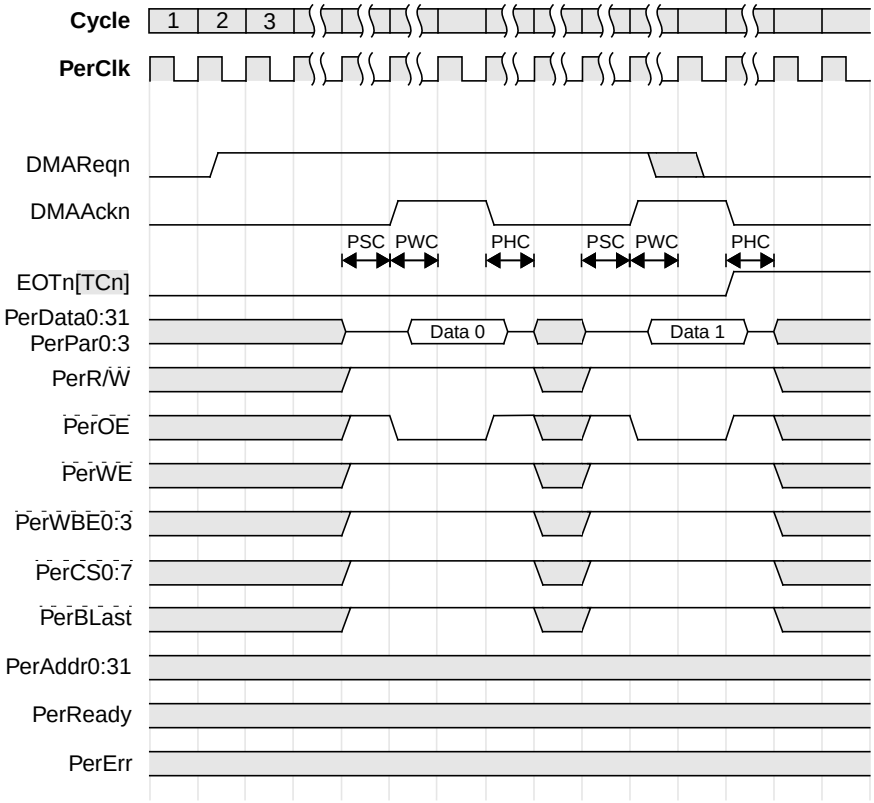


Figure 16-11. Peripheral-to-Memory DMA Transfer

Peripheral setup cycles (DMA0_CRn[PSC]) provide a delay between any previous operation on the peripheral bus and DMAAckn becoming active. Following the setup time DMAAckn is driven active for (DMA0_CRn[PWC] + 1) PerClk cycles. During peripheral-to-memory transfers read data from the peripheral is sampled on the PerClk edge where DMAAckn rises. After DMAAckn becomes inactive the peripheral bus is held idle for DMA0_CRn[PHC] PerClk cycles.

The second transfer in Figure 16-11 on page 16-18 illustrates the required DMAReqn timing to prevent a subsequent DMA transfer. For all peripheral mode transfers DMAReqn must be sampled inactive at the end of the last PerClk cycle where DMAAckn is active.

The EOTn[TCn] I/O can be configured either as an end of transfer input (DMA0_CRn[ETD]=0) or a terminal count output (DMA0_CRn[ETD]=1). When programmed as a terminal count output, EOTn[TCn] is asserted in the cycle after DMAAckn became inactive and the channel's count register (DMA0_CTn) reached zero. EOTn[TCn] remains active until the terminal count status bit is cleared in the DMA status register (DMA0_SR[CSn]).

If EOTn[TCn] is configured as an end of transfer input (DMA0_CRn[ETD]=0), EOTn[TCn] must be sampled active during the last DMAAckn cycle. If the channel is configured for scatter/gather transfers

EOTn[TCn] should be immediately deasserted to prevent the subsequent transfer from ending prematurely. Figure 16-12 on page 16-19 shows the required timing.

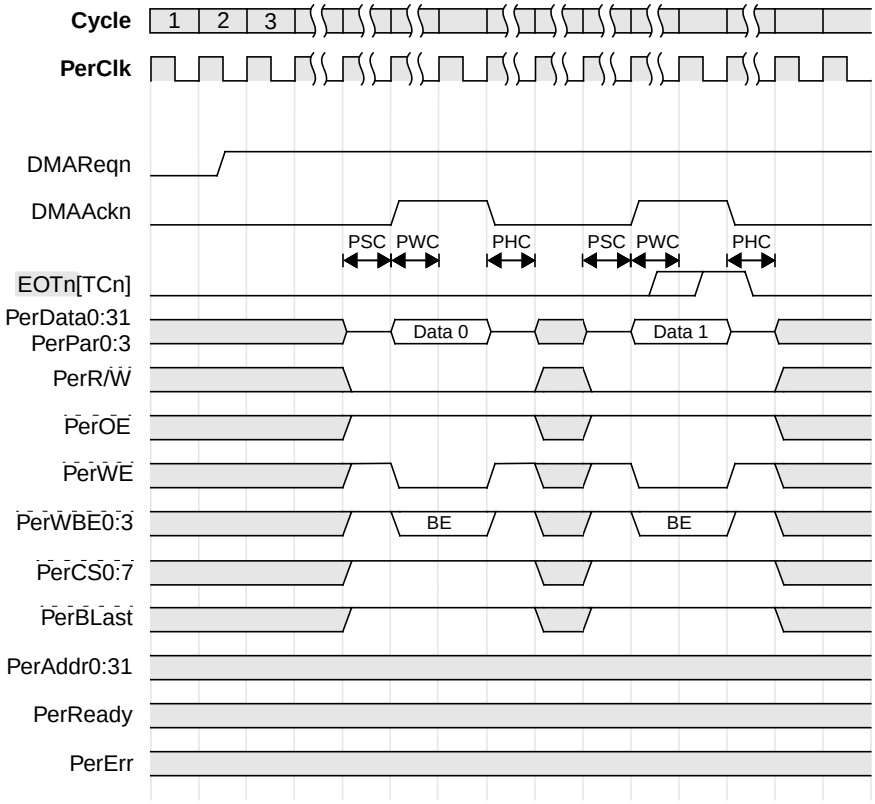


Figure 16-12. Memory to Peripheral DMA Transfer

For both peripheral-to-memory and memory-to-peripheral transfers the transfer width (DMA0_CR[PW]) must be set to the data bus width of the peripheral. This is because the DMA controller does not pack or unpack data on the peripheral side of transaction.

Peripheral-to-Memory Transfer

To perform a peripheral-to-memory DMA transfer from an EBC-attached DMA peripheral:

1. Set destination address register (DMA0_DAn) to the desired memory location. The address must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
2. Program the count register (DMA0_CTn) for the number of transfers.
3. Clear the channel's status bits in the DMA status register (DMA0_SR).
4. In the channel control register (DMA0_CRn):
   a. Optionally enable the DMA buffer, BEN=1.
   b. Optionally enable parity checking, PCE=1.
   c. Set the destination address increment, DAI=1.
   d. Set the transfer mode to peripheral, TM=0b00.
   e. Set the peripheral location to external, PL=0.
   f. Set the transfer direction to peripheral-to-memory, TD=1.

g. Enable the channel CE=1.

Once the DMA channel is active, the peripheral initiates a transfer by activating the DMAReqn pin for the channel. The PPC405CR then activates the DMAAckn pin to read data from the peripheral. This continues until either a terminal count or end of transfer condition occurs.

Memory-to-Peripheral Transfer

To perform a memory-to-peripheral DMA transfer to an EBC-attached DMA peripheral:

1. Set source address register (DMA0_SAn) to the desired memory location. The address must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
2. Program the count register (DMA0_CTn) for the number of transfers.
3. Clear the channel's status bits in the DMA status register (DMA0_SR).
4. In the channel control register (DMA0_CRn):
   a. Optionally enable the DMA buffer, BEN=1, and set the desired prefetch count, PF.
   b. Optionally enable parity generation, PCE=1.
   c. Set the source address increment, SAI=1.
   d. Set the transfer mode to peripheral, TM=0b00.
   e. Set the peripheral location to external, PL=0.
   f. Set the transfer direction to memory-to-peripheral TD=0.
   g. Enable the channel, CE=1.

Once the DMA channel is active, the peripheral initiates a transfer by activating the DMAReqn pin for the channel. The PPC405CR then reads the source memory and subsequently activates the DMAAckn pin to write data to the peripheral. This continues until either a terminal count or end of transfer condition occurs.

16.9.2 Memory-to-Memory Transfers

Memory-to-memory transfers can be initiated either by software or by an external device. If initiated via software, the transfer begins as soon as the channel is configured and enabled. When initiated by hardware (also known as a device-paced memory-to-memory transfer), software configures the channel for a memory-to-memory move and transfers begin when an external device places an active request on the channel request line, DMAReqn.

Hardware-Initiated (Device-Paced) Memory-to-Memory Transfers

To perform a device-paced memory-to-memory DMA transfer:

1. Set the transfer width (DMA0_CRn[PW]) to the width of the device-paced memory.
2. Set the source (DMA0_SAn) and destination (DMA0_DAn) address registers to the desired memory locations. These addresses must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
3. Program the count register (DMA0_CTn) for the number of transfers.
4. Clear the channel's status bits in the DMA status register (DMA0_SR).
5. In the channel control register (DMA0_CRn):
   a. Optionally enable the DMA buffer, BEN=1, and set the desired prefetch count, PF.

b. If the device-paced memory is at the source memory location set PL=1.

c. Set the source address increment, SAI, and destination address increment, DAI, as desired.

d. Set the transfer mode to device-paced memory-to-memory, TM=0b11.

e. Enable the channel, CE=1.

Once the DMA channel is configured for this mode, the external device initiates a transfer by activating the DMAReqn input. The PPC405CR then reads the source memory, buffers the data in the DMA controller and then outputs the data to the destination memory address. Transfers continue as long as the controlling device maintains an active signal on DMAReqn and the channel count register (DMA0_CTn) is non-zero. To pause a device paced memory-to-memory transfer, the controlling device must deassert DMAReqn one PerClk cycle before the last cycle in the device-paced memory access.

16.9.3 Software-Initiated Memory-to-Memory Transfers (Non-Deviced Paced)

To perform a software-initiated memory-to-memory DMA transfer:

1. Set the transfer width (DMA0_CRn[PW]) as desired.
2. Set the source (DMA0_SAn) and destination (DMA0_DAn) address registers to the desired memory locations. These addresses must be aligned to the programmed transfer width (DMA0_CRn[PW]), otherwise an alignment error will occur.
3. Program the count register (DMA0_CTn) for the number of transfers.
4. Clear the channel's status bits in the DMA status register (DMA0_SR).
5. In the channel control register (DMA0_CRn):
   a. Optionally enable the DMA buffer, BEN=1.
   b. Set the source address increment, SAI, and destination address increment, DAI, as desired.
   c. Set the transfer mode to software-initiated memory-to-memory, TM=0b10.
   d. Enable the channel, CE=1.

Once the channel is enable the DMA controller transfers data from source to destination until the channel count reaches zero. Note that memory-to-memory transfers initiated by software do not use DMAReqn or DMAAckn.

Chapter 17. Serial Port Operations

The PPC405CR contains two universal asynchronous receiver/transmitters (UARTs) which provide two full-duplex serial interfaces to support communications with serial peripheral devices. Each UART is compatible with the National Semiconductor (NS) 16550 chip, and includes a 16-byte send and a 16-byte receive FIFO.

Features of the UART include:

- Compatible with the NS 16550
- 16-byte send FIFO, 16-byte receive FIFO
- Full duplex operation
- Programmable baud rate generator
- Supports 5- to 8-bit word size, 1 or 2 stop bits, even, odd, or no parity
- One 8-wire interface (UART0) and one 4-wire interface (UART1)

The UART performs serial-to-parallel conversion on data characters received from a peripheral device, and parallel-to-serial conversion on data characters received from the processor. The processor can read the complete status of the UART at any time during the functional operation. Status information reported includes the type and condition of the transfer operations being performed by the UART, as well as any error conditions, such as parity, overrun, framing, and break interrupt.

This UART is functionally identical to NS16450 in character mode (on power up it will be in this mode), and can be put into FIFO mode to relieve the processor of excessive software overhead. Here, internal FIFOs are activated allowing 16 bytes (plus 3 bits per byte of error data in the RCVR FIFO) to be stored in both receive and transmit modes.

The source of the UART serial clock input is selected in Chip Control Register 0 (CPC0_CR0[U0EC, U1EC]) bits 24 and 25. Either the internal serial clock or an external serial clock can be selected. A programmable baud rate generator is included that is capable of dividing the UART serial clock input by a divisor of 1 to ($2^{16} - 1$) and producing the 16× clock required for driving the UART internal transmitter/receiver logic. The internal serial clock input is derived from the CPU clock by a divisor specified in CPC0_CR0[UDIV].

The UART has an interrupt system that can be programmed to the user's requirements, helping to minimize the computing required to handle the communications link. UART interrupts are capable of triggering an interrupt request to the PPC405CR interrupt controller.

17.1 Functional Description

- Runs NS 16550 software
- Registers are identical to the NS16550 register set
- After reset, all registers are identical to the NS16450 register set
- Complete status reporting capability
- Transmitter and receiver are each buffered with 16-byte FIFOs when FIFO mode selected

- Can add/delete standard asynchronous communication bits such as start, stop, and parity to/from the serial data
- When in character mode, holding and shift registers eliminate the need for precise synchronization between the processor and serial data
- Full prioritized interrupt system controls
- Independently controlled transmit, receive, line status, and data set interrupts
- Programmable baud rate generator divides the UART serial clock input by 1 to (2^{16}–1) and generates the 16x clock:

  Baud rate (bps) = (Serial Clock Input) / (16 x Decimal Divisor)
- Receiver uses 5-way oversampling as follows: it samples each serial bit five times, and if at least three of the samples are 1's, the bit is determined to be a 1, otherwise it is a 0
- Fully programmable serial-interface characteristics:
  - 5-, 6-, 7-, or 8-bit characters
  - Even, odd, or no parity bit generation and detection
  - 1-,1.5-, or 2-stop bit generation
  - Variable baud rate
- Line break generation and detection, and false start bit detection
- Internal diagnostic capability:
  - Loopback controls for communications link fault isolation
  - Break, parity, overrun, framing error simulation

17.2 Serial Input Clocking

The two PPC405CR UARTs can be clocked individually from an external serial clock or from an internally generated serial clock. The internally generated serial clock is derived from the CPU clock, and is CPU/n, where n ranges from 1 to 32. The divisor n is programmed by setting a value of 0 to 31 in CPC0_CR0[UDIV] (see Chapter 6, "Clocking").

The choice of serial clock frequency affects the serial communications error rate. If an external clock of 1.8432 MHz (or some multiple of this frequency) is used, the error rate approaches zero. However, when using the internally generated clock only certain clock frequencies are possible, which results in a small, non-zero error rate in all cases, unless SysClk is chosen as an integer multiple of 1.8432 MHz.

The optimum serial clock frequency is determined from the following relationship:

Serial Clock = Baud Rate $\times$ 16 $\times$ UART Divisor

Acceptable baud rates are always integral multiples of 300 (for example, 1200 = 4 $\times$ 300). Table 17-1 shows optimum UART divisor and CPU divide ratios for a range of possible baud rates. This information is provided for four different CPU clock frequencies. The UART divisor is programmed in

UARTx_DLM and UARTx_DLL (see “Divisor Latch LSB and MSB Registers (UARTx_DLL, UARTx_DLM)” on page 17-14). The value range is 1 to (2^16-1)=65535.

Table 17-1. Baud Rate Settings

Desired Baud Rate (bps)	CPU:UART Divide Ratio	Serial Clock Frequency (MHz)	UART Divisor	Actual Baud Rate (bps)	Error (%)
CPU Clock = 150 MHz					
1200	13	11.5380	601	1199.923205	0.0063996
2400	21	7.1429	186	2400.153610	0.0064004
4800	21	7.1429	93	4800.307220	0.0064004
9600	16	9.3750	61	9605.532787	0.0576332
19200	14	10.7140	35	19132.653061	0.3507653
28800	13	11.5380	25	28846.153846	0.1602564
33600	9	16.6670	31	33602.150538	0.0064004
38400	9	16.6670	27	38580.246914	0.4693930
57600	18	8.3333	9	57870.370370	0.4693930
115200	9	16.6670	9	115740.740741	0.4693930
307200	30	5.0000	1	312500.000000	1.7252604
CPU Clock = 166.66 MHz					
1200	31	5.3763	280	1200.076800	0.0064000
2400	31	5.3763	140	2400.153600	0.0064000
4800	31	5.3763	70	4800.307200	0.0064000
9600	31	5.3763	35	9600.614401	0.0064000
19200	32	5.2083	17	19148.284237	0.2693529
28800	19	8.7719	19	28855.032202	0.1910840
33600	31	5.3763	10	33602.150403	0.0064000
38400	17	9.8039	16	38296.568474	0.2693529
57600	30	5.5556	6	57870.370139	0.4693926
115200	30	5.5556	3	115740.740278	0.4693926
307200	17	9.8039	2	306372.547794	0.2693529
CPU Clock = 200 MHz					
1200	11	18.1820	947	1199.961601	0.0031999
2400	28	7.1429	186	2400.153610	0.0064004
4800	28	7.1429	93	4800.307220	0.0064004
9600	14	14.2860	93	9600.614439	0.0064004
19200	31	6.4516	21	19201.228879	0.0064004
28800	31	6.4516	14	28801.843318	0.0064004
33600	31	6.4516	12	33602.150538	0.0064004
38400	25	8.0000	13	38461.538462	0.1602564
57600	31	6.4516	7	57603.686636	0.0064004
115200	12	16.6670	9	115740.740741	0.4693930
307200	20	10.0000	2	312500.000000	1.7252604

17.3 UART Registers

UART registers are accessed via memory locations 0xEF60_0XYY where X=3 for UART 0 and X=4 for UART 1.

Table 17-2. UART Configuration Registers

UART Config Register	Address	R/W	Description	Reset
UARTx_RBR	EF60_0X00 [1]	R	UART x Receiver Buffer Register	
UARTx_THR	EF60_0X00 [1]	W	UART x Transmitter Holding Register	
UARTx_IER	EF60_0X01 [1]	R/W	UART x Interrupt Enable Register	0000 0000
UARTx_IIR	EF60_0X02	R	UART x Interrupt Identification Register	0000 0001
UARTx_FCR	EF60_0X02	W	UART x FIFO Control Register	0000 0000
UARTx_LCR	EF60_0X03	R/W	UART x Line Control Register	0000 0000
UARTx_MCR	EF60_0X04	R/W	UART x Modem Control Register	0000 0000
UARTx_LSR	EF60_0X05	R/W	UART x Line Status Register	0110 0000
UARTx_MSR	EF60_0X06	R/W	UART x Modem Status Register	xxxx 0000
UARTx_SCR	EF60_0X07	R/W	UART x Scratch Register	
UARTx_DLL	EF60_0X00 [1]	R/W	UART x Divisor Latch (LSB)	
UARTx_DLM	EF60_0X01 [1]	R/W	UART x Divisor Latch (MSB)	

1. UARTx_LCR[DLAB] controls the function accessed through registers EF60_0X00 and EF60_0X01. When UARTx_LCR[DLAB] is 0, access is enabled to the Receiver/Transmitter registers and the Interrupt Enable register. When UARTx_LCR[DLAB] is a 1, access is enabled to the Divisor Latch registers.

The system programmer may access any of the UART registers via the processor. These registers control all UART operations including transmission and reception of data. In PPC405CR there are two UARTs, designated 0 (8-wire interface) and 1 (4-wire interface). In the following sections, the registers are specified with a generic name where x represents 0 or 1. For example, the Line Control Register appears as a UARTx_LCR.

For UART1, two of the four wires are TX and RX. The remaining two wires can be programmed as a combination of DTR and DSR, or CTS and RTS in CPC0_CR0(DCS, RDS). DCD and RI are not available on the 4-wire interface.

17.3.1 Receiver Buffer Registers (UARTx_RBR)

Figure 17-1. UART Receiver Buffer Registers (UARTx_RBR)

0:7		Data bit

Note: UARTx_RBR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.2 Transmitter Holding Registers (UARTx_THR)

Figure 17-2. UART Transmitter Holding Registers (UARTx_THR)

0:7		Data bit

Note: UARTx_THR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.3 Interrupt Enable Registers (UARTx_IER)

Five UART interrupts on four priority levels are enabled via the Interrupt Enable Register, UARTx_IER. Any of the five interrupts can be used to surface a UART interrupt to the PPC405CR interrupt controller. Each interrupt can be enabled by setting its appropriate bit. Resetting UARTx_IER[4:7] totally disables the UART interrupt system. Disabling an interrupt prevents it from being shown as active in the UARTx_IIR and prevents it from signaling a UART interrupt to the PPC405CR interrupt controller. See Table 17-3, “Interrupt Priority Level,” on page 17-7.

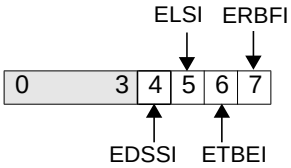


Figure 17-3. UART Interrupt Enable Registers (UARTx_IER)

0:3		Reserved	Always 0.

4	EDSSI	Enable Modem Status Interrupt 0 Disable modem status interrupt 1 Enable modem status interrupt	
5	ELSI	Enable Receiver Line Status Interrupt 0 Disable receiver line status interrupt 1 Enable receiver line status interrupt	
6	ETBEI	Enable Transmitter Holding Register Empty Interrupt 0 Disable transmitter holding register empty interrupt 1 Enable transmitter holding register empty interrupt	
7	ERBFI	Enable Received Data Available Interrupt 0 Disable received data available interrupt 1 Enable received data available interrupt	In FIFO mode, timeout interrupts follow the enable/disable state of ERDAI.

Note: UARTx_IER is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.4 Interrupt Identification Registers (UARTx_IIR)

The UART prioritizes interrupts into four levels which are recorded in the Interrupt Identification Register. The interrupt types in the order of their priority are as follows:

1. Receiver line status
2. Received data available and character timeout indication
3. Transmitter holding register empty
4. Modem status

Table 17-3 lists the interrupt priority levels.

Table 17-3. Interrupt Priority Level

IIR Bit 4	IIR Bit 5	IIR Bit 6	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	1	1	1	Receiver Line Status	Overrun, Parity or Framing Error, or Break Interrupt.	Read LSR.
0	1	0	2	Received Data Available	Receiver data available or trigger level reached.	Read RBR, or FIFO drops below trigger level.
1	1	0	2	Character Timeout Indication	No characters have been removed from or input to the receiver FIFO during the last four character times and it contains at least one character during this time.	Read RBR.
0	0	1	3	Transmitter Holding Register Empty	Transmitter Holding Register Empty.	Read IIR (if source of interrupt) or write THR.
0	0	0	4	Modem Status	Clear to Send, Data Set Ready, Ring Indicator or Data Carrier Detect.	Read MSR.

When the processor accesses UARTx_IIR, the UART records new interrupts, but does not change its current contents until the access by the processor is complete. The UART indicates the highest priority interrupt pending to the PPC405CR interrupt controller via the IIR.

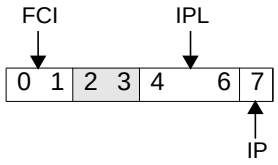


Figure 17-4. UART Interrupt Identification Registers (UARTx_IIR)

0:1	FCI	FIFO Control Indicator 00 FIFOs disabled (UARTx_FCR[FC] = 0) 01 Reserved 10 Reserved 11 FIFOs enabled (UARTx_FCR[FC] = 1)
2:3		Reserved

4:6	IPL	Interrupt Priority Level 000 Priority level 4 001 Priority level 3 010 Priority level 2 011 Priority level 1 100 Reserved 101 Reserved 110 Priority level 2 111 Reserved	See Table 17-3. Note: Priority 1 is highest priority.
7	IP	Interrupt Pending 0 Interrupt is pending 1 No interrupt pending	When set to 0, IIR contents can be used as a pointer to the appropriate interrupt service routine.

Note: UARTx_IIR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.5 FIFO Control Registers (UARTx_FCR)

The FIFO control register has the same address as the IIR and is a write-only register. This register is used to perform FIFO control operations such as selecting the type of DMA signaling, setting the receiver FIFO trigger levels, clearing the FIFOs, and enabling the FIFO.

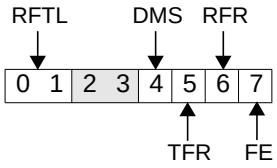


Figure 17-5. UART FIFO Control Registers (UARTx_FCR)

0:1	RFTL	Receiver FIFO Trigger Level 00 1 byte 01 4 bytes 10 8 bytes 11 14 bytes	
2:3		Reserved	
4	DMS	DMA Mode Select 0 Mode 0 = single transfer 1 Mode 1 = multiple transfers	Select single or multiple transfer mode if UARTx_FCR[7] = 1.
5	TFR	Transmitter FIFO Reset 0 Operation complete 1 Reset the transmitter FIFO	A 1 written to this bit clears all bytes in the transmitter FIFO and resets all of its counter logic to 0. The transmitter shift register is not cleared. This bit is self-clearing.
6	RFR	Receiver FIFO Reset 0 Operation complete 1 Reset the receiver FIFO	A 1 written to this bit clears all bytes in the receiver FIFO and resets all of its counter logic to 0. The receiver shift register is not cleared. This bit is self-clearing.

7	FE	FIFO Enable 0 Disable FIFOs 1 Enable FIFOs	When set to 1, both the receiver and transmitter FIFOs are enabled. When set to 0, both receiver and transmitter FIFOs are reset. Data is automatically cleared from both FIFOs when changing to and from FIFO and 16450 modes. Programming other bits will be ignored if this bit is not a 1.

Note: UARTx_FCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.6 Line Control Registers (UARTx_LCR)

The system programmer uses the line control register (LCR) to specify the format of the asynchronous data communications exchange and to set the Divisor Latch Access bit. The contents of the LCR can also be read by the processor. The read capability simplifies system programming, and eliminates the need for separate storage of the line characteristics in system memory.

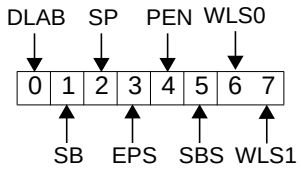


Figure 17-6. UART Line Control Registers (UARTx_LCR)

0	DLAB	Divisor Latch Access Bit 0 Address RBR, THR and IER with LTADR2–0 for read or write operation 1 Address Divisor Latches with LTADR2–0 for read or write operation	
1	SB	Set Break 0 Disable Break 1 Enable Break	Causes a break condition to be transmitted to the UART when the core is receiving. SOUT is forced to the spacing state (0). This bit acts only on SOUT and has no effect on the transmitter logic.
2	SP	Sticky Parity 0 Disable sticky parity 1 Enable sticky parity	If UARTx_LCR[EPS] = 1 and UARTx_LCR[PE] = 1, the parity bit is transmitted and checked as 0. If UARTx_LCR [EPS] = 0 and UARTx_LCR[PE] = 1,the parity bit is transmitted and checked as 1.
3	EPS	Even Parity Select 0 Generate odd parity 1 Generate even parity	This bit is significant only if UARTx_LCR[PE] = 1.
4	PEN	Parity Enable 0 Disable parity checking 1 Enable parity checking	

5	SBS	Stop Bit Select 0 Characters have 1 stop bit 1 Characters have 1.5 or 2 stop bits	If UARTx_LCR[CL] = 00, characters have 1.5 stop bits. For any other value of UARTx_LCR[CL], characters have 2 stop bits. The receiver checks the first stop bit only, regardless of how many stop bits are selected.
6:7	WLS0, WLS1	Word Length Select Bits 0,1 00 Use 5-bit characters 01 Use 6-bit characters 10 Use 7-bit characters 11 Use 8-bit characters	

Note: UARTx_LCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.7 Modem Control Registers (UARTx_MCR)

The interface between the modem, data set, or peripheral device emulating a modem, and the UART, is controlled by the Modem Control Register (UARTx_MCR).

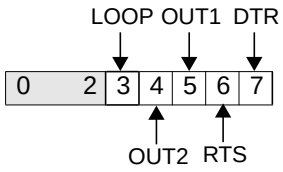


Figure 17-7. UART Modem Control Registers (UARTx_MCR)

0:2		Reserved	Always 0.
3	LOOP	Loopback Mode 0 Disabled 1 Enabled	Provides a local loopback feature for diagnostic testing of the UART. The following occurs: 1. SOUT is set to the marking state (logic 1) SIN is disconnected. 2. The output of the transmitter shift register feeds the input of the receiver shift register. 3. The four modem control inputs $\overline{DSR}$, $\overline{CTS}$, $\overline{RI}$, and $\overline{DCD}$ are disconnected. 4. The four modem control outputs $\overline{DTR}$, $\overline{RTS}$, $\overline{OUT1}$, and $\overline{OUT2}$ are set to a logic 1 (their inactive state). 5. The four modem control outputs are connected internally to the four modem control inputs. Transmitted data is immediately received to verify the UART transmit and receive data paths. Receiver and transmitter interrupts are operational. Their sources are external to the UART. Also operational are the modem control interrupts, but their source is the low-order 4 bits of UARTx_MCR instead of the modem control inputs to the UART. UARTx_IER still controls the interrupts.
4	OUT2	User Output 2 0 $\overline{OUT2}$ inactive (1) 1 $\overline{OUT2}$ active (0)	Auxiliary user designated output.

5	OUT1	User Output 1 0 $\overline{OUT1}$ inactive (1) 1 $\overline{OUT1}$ active (0)	Auxiliary user designated output.
6	RTS	Request To Send 0 $\overline{RTS}$ inactive (1) 1 $\overline{RTS}$ active (0)	
7	DTR	Data Terminal Ready 0 $\overline{DTR}$ inactive (1) 1 $\overline{DTR}$ active (0)	

Note: UARTx_MCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.8 Line Status Registers (UARTx_LSR)

Information concerning the data transfer is held for the processor in this register. Bits 3 through 6 are conditions that produce a receiver line status interrupt whenever the condition corresponding to the active bit is detected and the interrupt is enabled. This register is intended for read operations only and writing is not recommended.

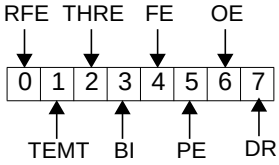


Figure 17-8. UART Line Status Registers (UARTx_LSR)

0	RFE	Receiver FIFO Error Indicator 0 In FIFO mode, reset to 0 when the processor reads the UARTx_LSR, provided there are no subsequent errors in the FIFO. 1 There are one or more instances of parity error, framing error or break indication in the FIFO.	Always 0 in 16450 mode.
1	TEMT	Transmitter Empty Indicator 0 Reset to 0 whenever the THR or the transmitter shift register contain a character. In FIFO mode, it is reset to 0 whenever the transmitter FIFO or the transmitter shift register contain a character. 1 Set to 1 when the THR and the Transmitter shift register are both empty. In FIFO mode, it is set to 1 when the transmitter FIFO and the transmitter shift register are both empty.	

2	THRE	Transmitter Holding Register Empty Indicator 0 Concurrent reset to 0 with the loading of the THR by the processor. In FIFO mode it is reset to 0 when at least one byte is written to the transmitter FIFO. 1 Set to 1 when the UART is ready to accept a new character for transmission. In FIFO mode, this bit is set when the transmitter FIFO is empty.	When UARTx_IER[THRE] = 1, the UART issues an interrupt to the PPC405CR interrupt controller. This bit is set to 1 when a character is transferred from the THR to the transmitter shift register.
3	BI	Break Interrupt Indicator. 0 Reset to 0 whenever processor reads Line Status Register (LSR). 1 Set to 1 whenever the received data input is held at the spacing level (0) for longer than a full word transmission time.	The word transmission time is the time required for the start bit, data bits (can be 5–8 bits), parity and stop bits. In FIFO mode, this error is reported to the processor when the character associated with the error is at the top of the FIFO. Only one 0 character is loaded into the receiver FIFO when a break occurs. After the next valid start bit is received and is in the marking state, the next character transfer is enabled. The error causes a Receiver Line Status Interrupt.
4	FE	Framing Error Indicator. 0 Reset to 0 whenever processor reads LSR. 1 Set to 1 whenever stop bit following the last data bit or parity bit is detected as 0 (spacing level).Indicates that a valid stop bit was not found in the received character.	Error causes a Receiver Line Status Interrupt.
5	PE	Parity Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Indicates that the received data character does not have the correct parity as determined by the even parity select bit (UARTx_LCR.[EPS]). Set to 1 upon detection of a parity error.	In FIFO mode, this error is revealed to the processor when the character this error is associated with is at the top of the FIFO. Error causes a Receiver Line Status Interrupt.
6	OE	Overrun Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Data in the RBR was read by the processor before the next character was transferred into the UARTx_RBR, hence the original data was lost.	In FIFO mode, if the incoming data continues to fill the FIFO beyond the trigger level, an OE occurs only after the FIFO is completely full and the entire next character has been received in the receiver shift register. The processor is informed of the OE immediately upon occurrence. The character in the shift register will be overwritten and will not be transferred to the FIFO. Error causes a Receiver Line Status Interrupt.

7	DR	Receiver Data Ready Indicator. 0 Reset to 0 when all data has been read from the receiver FIFO or the UARTx_RBR. 1 An entire incoming character has been received into the UARTx_RBR or receiver FIFO.

Note: UARTx_LSR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.9 Modem Status Registers (UARTx_MSR)

The processor can monitor the present state of the modem (or peripheral device) control lines by reading the Modem Status Register (UARTx_MSR). In addition, the UARTx_MSR has four bits to indicate if any of the modem (or peripheral device) control lines have changed state.

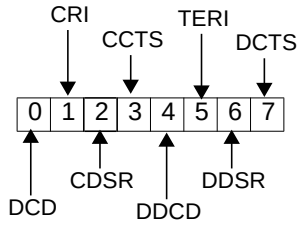


Figure 17-9. UART Modem Status Registers (UARTx_MSR)

0	DCD	Data Carrier Detect	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT2].
1	CRI	Complement of Ring Indicator	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT1].
2	CDSR	Complement of Data Set Ready	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[DTR].
3	CCTS	Complement of Clear To Send	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[RTS].
4	DDCD	Delta Data Carrier Detect 0 Set when processor reads the Modem Status Register 1 $\overline{\text{DCD}}$ input changed state	Indicates that the $\overline{\text{DCD}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
5	TERI	Trailing Edge of Ring Indicator 0 Set when processor reads the Modem Status Register 1 $\overline{\text{RI}}$ input changed from 0 to 1	Indicates that the $\overline{\text{RI}}$ input to the UART changed from 0 to 1 since the processor last read the Modem Status Register. A modem status interrupt is generated.
6	DDSR	Delta Data Set Ready 0 Set when processor reads the Modem Status Register 1 $\overline{\text{DSR}}$ input changed state	Indicates that the $\overline{\text{DSR}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
7	DCTS	Delta Clear To Send 0 Set when processor reads the Modem Status Register 1 $\overline{\text{CTS}}$ input changed state	Indicates that the $\overline{\text{CTS}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.

Note: UARTx_MSR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.10 Scratchpad Registers (UARTx_SCR)

A scratchpad register intended for use by the programmer as a temporary data location is provided in this UART. It does not control the UART operation in any way.

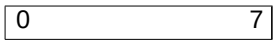


Figure 17-10. Scratchpad Registers (UARTx_SCR)

0:7		Data bits

Note: UARTx_SCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

17.3.11 Divisor Latch LSB and MSB Registers (UARTx_DLL, UARTx_DLM)

The divisor latches are used to program the UART divisor used in generating the baud clock. A 16-bit divisor may be programmed through these registers. Access to these registers is provided by setting UARTx_LCR[DLAB] = 1. These registers have a power-on reset value of 0.

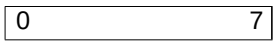


Figure 17-11. UART Baud-Rate Divisor Latch (MSB) Registers (UARTx_DLM)

0:7		Data bits

Note: UARTx_DLM is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

Figure 17-12. UART Baud-Rate Divisor Latch (LSB) Registers (UARTx_DLL)

8:15		Data bits

Note: UARTx_DLL is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

The UART divisor is calculated using the following formula:

UART Divisor = Serial Input Clock/(16 x Baud Rate)

For example, if the serial input clock= 11.0592MHz and a baud rate of 9600 bps is required:

UART Divisor = Serial Input Clock/(16 x Baud Rate)

= 11,059,200/(16x9600)

= 72 = 0x48

For this example, UARTx_DLM should be programmed to 0 and UARTx_DLL register should be programmed to 0x48. Due to the error introduced by rounding, some baud rates cannot be generated at certain serial input clock frequencies. Table 17-4 lists some common baud rates and their corresponding Divisor Latch register values with a serial input clock of 11.0592MHz.

Table 17-4. Divisor Latch Settings for Certain Baud Rates

Baud Rate (bps)	Divisor Latch MSB	Divisor Latch LSB
9600	0x00	0x48
19200	0x00	0x24
28800	0x00	0x18
38400	0x00	0x12
57600	0x00	0x0C

17.4 FIFO Operation

17.4.1 Interrupt Mode

17.4.1.1 Receiver

Receiver interrupts occur as described below when the receiver FIFO and receiver interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ERBFI] = 1.

The received data available interrupt is issued when the number of characters in the FIFO has reached the trigger level programmed into UARTx_FCR. This interrupt is reset to 0 when the FIFO character count drops below this trigger level.

The received data available indicator is issued when the number of characters in the FIFO has reached the trigger level programmed into UARTx_FCR. This indicator is reset to 0 when the FIFO character count drops below this trigger level.

The receiver line status interrupt (UARTx_IIR = 0xC6) is a top priority interrupt, whereas the received data available interrupt (UARTx_IIR = 0xC4) is a second priority interrupt.

Data Ready (UARTx_LSR[DR]) is set as soon as a character is transferred from the shift register to the receiver FIFO. This bit is reset when the FIFO is empty.

Receiver timeout interrupts will occur as described below when the receiver FIFO and receiver interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ERBFI] = 1.

A FIFO timeout will occur when:

At least one character is in the receiver FIFO, no serial characters have been received for four serial character time periods, and the processor has not read the FIFO for four serial character time periods. A serial character time period is as follows:

1/(baud rate) x (# start bits + word length + # parity bits + # stop bits)

For example, the serial character time period for an 8-bit word with one parity bit, two stop bits at 56K baud is as follows:

1/(56000) x (1 + 8 + 1 + 2) = 214.3µs

So the timeout would occur after 857.1µs, if the above conditions hold.

When a timeout interrupt has occurred, it is cleared and the timer is reset when the processor reads one character from the receiver FIFO.

When a timeout interrupt has not occurred, the timer is reset after a new serial character is received or the processor reads the receiver FIFO.

17.4.1.2 Transmitter

Transmitter interrupts occur, as described below, when the transmitter FIFO and transmitter interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ETBEI] = 1.

The transmitter holding register interrupt (UARTx_IIR = 0xC2) occurs when transmit FIFO is empty, and is cleared as soon as the transmitter holding register is written to or the IIR is read. One to 16 characters may be written to the transmitter FIFO while servicing this interrupt.

The transmitter FIFO empty indications are delayed by one character time minus the last stop bit time whenever the following event occurs: UARTx_LSR[THRE] = 1 and there were less than two bytes simultaneously present in the transmit FIFO since the last UARTx_LSR[THRE] = 1. If UARTx_FCR[FE] = 1 (FIFOs enabled), the first transmitter interrupt after changing UARTx_FCR[FE] is immediate.

Receiver FIFO trigger level interrupts, received data available interrupts, and character timeouts all have equivalent second interrupt priority. Current transmitter holding register empty interrupt and Transmit FIFO empty have equivalent third interrupt priority.

17.4.2 Polled Mode

When UARTx_FCR[FE] = 1 (FIFOs enabled), and UARTx_IER[5:7] are all set to 0 (interrupts disabled), the UART is in FIFO polled mode of operation. The receiver and transmitter are controlled separately, so either can be in polled mode of operation. In polled mode, the user program must check the UARTx_LSR to see the status of the receiver and/or transmitter.

UARTx_LSR3:6 specifies which errors (if any) have occurred. Character status errors are handled in the same way as in interrupt mode. Since UARTx_IER[ELSI] = 0, the IIR is not affected. UARTx_LSR[DR] is set as long as there is at least one character in the receiver FIFO. UARTx_LSR[THRE] indicates if the transmitter FIFO is empty. UARTx_LSR[TEMT] indicates if the transmitter FIFO and the transmitter shift register are empty. UARTx_LSR[RFE] indicates if there are any errors in the receiver FIFO.

In FIFO polled mode, there are no character timeout or trigger levels; however, the FIFOs are still capable of holding characters.

17.5 UART and Sleep Mode

Both UARTs can be placed in sleep mode via the UART sleep bits in the CPC0_ER register (CPC0_ER[UART0:UART1]). The most common usage would be to save a little power if one or both of the UARTs were not going to be used.

Using sleep mode dynamically requires careful software control to make sure the UARTs are idle before putting them to sleep.

17.6 DMA Operation

The DMA controller can be configured to perform DMA operations using UART0, which appears as an 8-bit peripheral to the DMA controller. When selected, the UART receiver is internally wired to the DMAReq and DMAAck signals of DMA channel 2, and the transmitter is internally wired to the DMAReq and DMAAck signals of DMA channel 3.

The UART can be operated in FIFO mode or non-FIFO mode. In FIFO mode, the transfers can be done as single transfers (DMA mode 0) or multiple transfers (DMA mode 1), depending on the setting of the DMS field in the FIFO Control Register (FCR). In non-FIFO mode, DMA transfers are performed using single transfers, using the UART's DMA mode 0. This section describes proper UART0 DMA programming. For more information on general DMA programming, see Chapter 16, "Direct Memory Access Controller," on page 16-1.

17.6.1 Chip Control Register 0 (CPC0_CR0)

Only CPC0_CR0 fields related to UART are shown in Figure 17-13. Other non-related functions are not shown here.

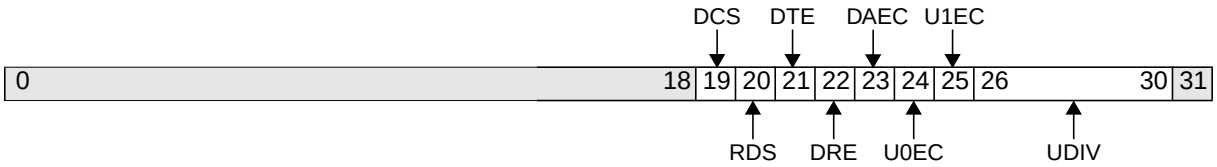


Figure 17-13. Chip Control Register 0 (CPC0_CR0)

0:18		Reserved.	
19	DCS	DSR/CTS select 0 DSR is selected. 1 CTS is selected.	
20	RDS	RTS/DTR select 0 RTS is selected. 1 DTR is selected.	
21	DTE	DMA Transmit Enable for UART0 0 DMA transmit channel is disabled. 1 DMA transmit channel is enabled.	
22	DRE	DMA Receive Enable for UART0 0 DMA receive channel is disabled. 1 DMA receive channel is enabled.	
23	DAEC	DMA Allow Enable Clear for UART0 0 DTE and DRE for UART0 are not cleared when the UART receives a corresponding terminal count. 1 DTE and DRE for UART0 are cleared when the UART receives a corresponding terminal count.	
24	U0EC	Select External Clock for UART0 0 UART0 uses the internally derived serial clock. 1 UART0 uses the UARTSerClk external serial clock input.	
25	U1EC	Select External Clock for UART1 0 UART1 uses the internally derived serial clock. 1 UART1 uses the UARTSerClk external serial clock input.	
26:30	UDIV	UART Divisor 00000 Divide by 1 00001 Divide by 2 00010 Divide by 3 . . . 11110 Divide by 31 11111 Divide by 32	UDIV specifies the divider ratio between the CPU and UART serial clock frequencies. UART0 and UART1 can use a serial clock frequency derived from the CPU clock frequency divided by UDIV, or use the UARTSerClk external serial clock input. For example, if the CPU runs at 200MHz, a UDIV of 20 sets the serial clock frequency at 10MHz. **Note:** Maximum serial clock frequency is slightly less than 1/2× OPB frequency.
31		Reserved.	

17.6.2 Transmitter DMA Mode

The UART0 Transmit Channel Enable field of the Chip Control Register 0, CPC0_CR0[DTE], controls the use of the serial port transmitter as a DMA destination. For the transmitter in DMA mode 0, when the FIFOs are disabled or the FIFOs are enabled and there are no characters in the TX FIFO or Transmit Holding Register (THR), the DMA request goes active. Once activated, the DMA request goes inactive after the first character is loaded into the TX FIFO or THR. For the transmitter in DMA mode 1, when FIFOs are enabled and there is at least one unfilled position in the TX FIFO, the DMA request goes active. This signal will become inactive when the TX FIFO is completely full. To operate in this mode, DMA Channel Control Register 3 (DMA0_CR3) must be configured to accept DMA requests from an internal source. Setting the Peripheral Location (PL) bit of DMA0_CR3 to a logic 1 configures DMA channel to accept DMA requests from UART0. Table 17-5 lists required register settings for UART0 transmit transfers. Other DMA registers and register fields must be programmed appropriately, see Chapter 16, “Direct Memory Access Controller,” on page 16-1 for more information.

Table 17-5. UART0 Transmitter DMA Mode Register Field Settings

Register [Field]	Meaning
CPC0_CR0[DTE]=1	UART0 DMA Transmit channel is enabled using DMA channel 3.
CPC0_CR0[DAEC]	Set to 0 to not clear CPC0_CR0[DTE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA0_CR3[TD]=0	DMA Channel 3 transfer direction is from memory to peripheral.
DMA0_CR3[PL]=1	DMA Channel 3 peripheral is on the OPB (UART0).
DMA0_CR3[PW]=00	Peripheral width is byte (8 bits).
DMA0_CR3[TM]=00	DMA Channel 3 is in peripheral mode.
DMA0_CR3[PWC]=000010	Peripheral Wait cycles, how long the internal DMAck is active. Three cycles are required.
DMA0_CR3[PHC]=000	Peripheral Hold Cycles are 0.
DMA0_CR3[ETD]=1	EOT/TC is programmed as terminal count output.
UART0_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.

Note: When using DMA Channel 3 for UART0 transmitter transfers, external DMA transfers cannot be performed on this channel.

17.6.3 Receiver DMA Mode

The UART0 Receive Enable field of the Chip Control Register 0, CPC0_CR0[DRE], controls the use of the serial port receiver as a DMA source. For the receiver in DMA mode 0, when there is at least one character in the RX FIFO or Receive Buffer Register, RBR, the DMA request goes active. Once activated, the DMA request goes inactive when there are no more characters in the FIFO or RBR. For the receiver in DMA mode 1, when the FIFOs are enabled and the trigger level or the timeout has been reached, the DMA request goes active. Once activated, it will go inactive when there are no more characters in the RX FIFO or RBR. To operate in this mode, DMA Channel Control Register 2 (DMA0_CR2) must be configured to accept DMA requests from an internal source. Setting the Peripheral Location (PL) bit of DMA0_CR2 to a logic 1 configures DMA channel to accept DMA requests from UART0. Table 17-6 lists required register settings for UART0 receiver transfers. Other DMA registers and register fields must be programmed appropriately, see Chapter 16, “Direct Memory Access Controller,” on page 16-1 for more information.

Table 17-6. UART0 Receiver DMA Mode Register Field Settings

Register [Field]	Meaning
CPC0_CR0[DRE]=1	UART0 DMA Receiver channel is enabled using DMA channel 2.
CPC0_CR0[DAEC]	Set to 0 to not clear CPC0_CR0[DRE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA0_CR2[TD]=1	DMA Channel 2 transfer direction is from peripheral to memory.
DMA0_CR2[PL]=1	DMA Channel 2 peripheral is on the OPB (UART0).
DMA0_CR2[PW]=00	Peripheral width is byte (8 bits).
DMA0_CR2[TM]=00	DMA Channel 2 is in peripheral mode.
DMA0_CR3[PWC]=000010	Peripheral Wait cycles, how long the internal DMAAck is active. Three cycles are required.
DMA0_CR2[PHC]=000	Peripheral Hold Cycles are 0.
DMA0_CR2[ETD]=1	EOT/TC is programmed as terminal count output.
UART0_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.

Note: When using DMA Channel 2 for UART0 receiver transfers, external DMA transfers cannot be performed on this channel.

Chapter 18. IIC Bus Interface

The PPC405CR provides an inter-integrated circuit (IIC) bus interface complying with specifications contained in the Philips® Semiconductors document *The I²C-bus and how to use it (including specifications)* (1995 update).

The IIC bus is a two-wire, bi-directional, open-drain, low-speed serial interface. The serial clock (IICSCL) and serial data (IICSDA) lines are bidirectional, to support multiple bus masters and to mix high- and low-speed devices on the same bus.

The IIC interface (referred to as IIC to distinguish it from the Philips I²C bus) supports the following standard and enhanced features:

- 100-kHz and 400-kHz operation
- 8-bit data transfers
- 7-bit and 10-bit addressing
- Slave transmitter and receiver
- Master transmitter and receiver
- Multiple bus masters

The IIC interface can switch between 7-bit and 10-bit addressing under program control.

18.1 Addressing

The IIC interface supports 7-bit and 10-bit addressing for master and slave transfers.

Addressing is described in detail in “IIC0 Low Master Address Register” on page 18-5, “IIC0 High Master Address Register” on page 18-6, “IIC0 Low Slave Address Register” on page 18-14, and “IIC0 High Slave Address Register” on page 18-14.

Descriptions of addressing modes and address formats follow.

18.1.1 Addressing Modes

For master transfers, the address mode (AMD) field of the IIC Control register (IIC0_CNTL) controls whether 7-bit or 10-bit addresses are used. If IIC0_CNTL[AMD] = 0, addresses contain 7 bits; if IIC0_CNTL[AMD] = 1, addresses contain 10 bits.

For slave transfers, the contents of the IIC0 High Slave Address register (IIC0_HSADR) determines whether 7-bit or 10-bit addressing is used. If IIC0_HSADR = 0b00000000, 7-bit addressing is used. If 10-bit addressing is to be used for slave transfers, IIC0_HSADR = 0b11110*yyx*, where *yy* contains the high-order bits of the 10-bit address, and x is a don’t care.

Programming Note: For slave transfers, IIC0_CNTL[AMD] does not control addressing mode.

18.1.2 Seven-Bit Addresses

Figure 18-1 illustrates a 7-bit address. For master transfers, the address bits 0 through 6 (A0:A6) are read from IIC0_LMADR. For slave transfers, A0:A6 are read from IIC0_LSADR. Bit 7 of address byte 0 contains a transfer type bit provided by the IIC interface.

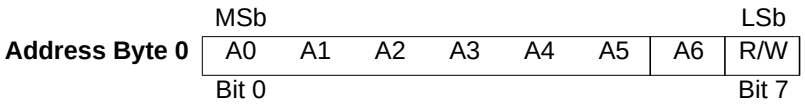


Figure 18-1. 7-Bit Addressing

18.1.3 Ten-Bit Addresses

Figure 18-2 illustrates a 10-bit address. A0:A1 of address byte 0 are read from IIC0_HMADR[A6:A7] (for master transfers) or IIC0_HSADR[A6:A7] (for slave transfers). These are the two highest-order address bits transmitted on the IIC bus. Bit 7 of address byte 0 contains a transfer type bit provided by the IIC interface.

For 10-bit addressing for master or slave transfers, respectively, IIC0_HMADR[A0:A4] and IIC0_HSADR [A0:A4] must contain 0b11110.

The low-order byte of the 10-bit address, contained in A0:A7 of address byte 1, are read from IIC0_LMADR or IIC0_LSADR for master or slave transfers, respectively.

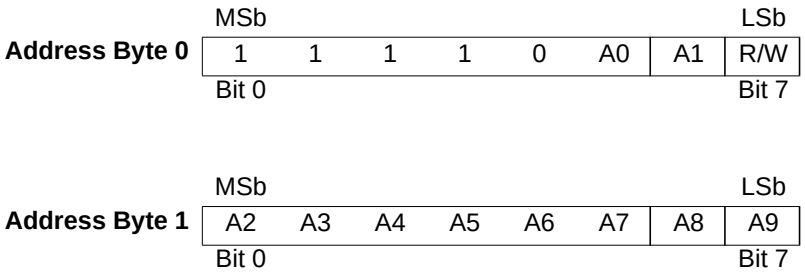


Figure 18-2. 10-Bit Addressing

18.2 IIC Registers

IIC registers are accessed at memory locations 0xEF600500–0xEF600510 in the PPC405CR.

Table 18-1 lists the IIC registers. Descriptions of the registers, in the listed order, follow in “IIC Register Descriptions” on page 18-3.

Table 18-1. IIC Registers

Register	Mnemonic	PPC405CR Memory Map	Address	Access	Effect of Reset	Bits
IIC0 Master Data Buffer	IIC0_MDBUF	0xEF60 0500	0x0	R/W	Cleared	8,16
Reserved		0xEF60 0501	0x1			

Table 18-1. IIC Registers

Register	Mnemonic	PPC405CR Memory Map	Address	Access	Effect of Reset	Bits
IIC0 Slave Data Buffer	IIC0_SDBUF	0xEF60 0502	0x2	R/W	Cleared	8,16
IIC0 Reserved		0xEF60 0503	0x3			
IIC0 Low Master Address	IIC0_LMADR	0xEF60 0504	0x4	R/W	No	8
IIC0 High Master Address	IIC0_HMADR	0xEF60 0505	0x5	R/W	No	8
IIC0 Control	IIC0_CNTL	0xEF60 0506	0x6	R/W	Cleared	8
IIC0 Mode Control	IIC0_MDCNTL	0xEF60 0507	0x7	R/W	Cleared	8
IIC0 Status	IIC0_STS	0xEF60 0508	0x8	R/W	Cleared	8
IIC0 Extended Status	IIC0_EXTSTS	0xEF60 0509	0x9	R/W	Cleared	8
IIC0 Low Slave Address	IIC0_LSADR	0xEF60 050A	0xA	R/W	No	8
IIC0 High Slave Address	IIC0_HSADR	0xEF60 050B	0xB	R/W	No	8
IIC0 Clock Divide	IIC0_CLKDIV	0xEF60 050C	0xC	R/W	Cleared	8
IIC0 Interrupt Mask	IIC0_INTRMSK	0xEF60 050D	0xD	R/W	Cleared	8
IIC0 Transfer Count	IIC0_XFRCNT	0xEF60 050E	0xE	R/W	Cleared	8
IIC0 Extended Control and Slave Status	IIC0_XTCNTLSS	0xEF60 050F	0xF	R/W	Cleared	8
IIC0 Direct Control	IIC0_DIRECTCNTL	0xEF60 0510	0x10	R/W	0x0f	4

18.3 IIC Register Descriptions

The following sections contains the bit definitions for the various registers in the IIC interface.

18.3.1 IIC0 Master Data Buffer

The IIC0 Master Data Buffer (IIC0_MDBUF) is a 1-byte × 4-byte first-in/first-out (FIFO) buffer. A byte written to IIC0_MDBUF is placed into the fourth FIFO stage. If the third FIFO stage is empty, the data is moved into the third stage at the next OPB clock. This process is repeated for the second and first FIFO stages at each successive OPB clock. The byte moves through the buffer until it reaches the deepest unoccupied stage of the FIFO. The buffer data is either written on the IIC bus when the IIC interface performs a write, or is received from the IIC bus when the IIC interface performs a read.

Figure 18-3 illustrates the IIC0_MDBUF.

Figure 18-3. IIC0 Master Data Buffer (IIC0_MDBUF)

0		Data bit
1		Data bit
2		Data bit

3		Data bit
4		Data bit
5		Data bit
6		Data bit
7		Data bit

IIC0_MDBUF is cleared (flushed and set to empty) whenever the IIC interface is reset, or IIC0_MDCNTL[FMB] = 1. Figure 18-4 shows the four FIFO stages.

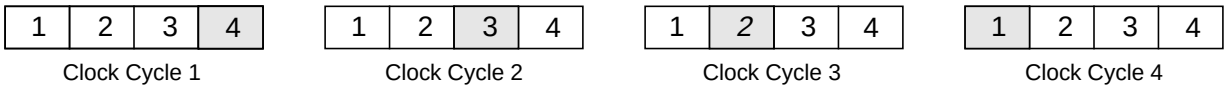


Figure 18-4. FIFO Stages

When IIC0_MDBUF is written with a byte, the byte is placed in the FIFO. The hardware pushes the byte into the deepest unoccupied stage in the FIFO and advances one FIFO stage per clock. Thus, if the FIFO is empty, four clocks are needed (one per stage) for the byte to walk to the first stage of the FIFO. This timing is important to consider when reading the IIC0_MDBUF immediately after data is written. When a master transfer is requested, the IIC interface handles this latency.

If a byte is written to IIC0_MDBUF while the FIFO is full, the byte is discarded and not placed into the FIFO.

If IIC0_MDBUF is written with two bytes in a halfword access, and there is space in the FIFO, byte 0 of the halfword is placed ahead of byte 1 in the FIFO. The MSB, byte 0, is written to the IIC bus first, followed by the LSB, byte 1.

IIC0_MDBUF receives data from the IIC bus when the requested master transfer is a read. The first byte received is the first byte read by software from IIC0_MDBUF.

For halfword reads the first byte received is MSB, byte 0, and the following byte is LSB, byte 1. When an empty FIFO is read, the byte (or halfword) most recently read is returned.

Care must be taken not to start a requested master operation while there is data in IIC0_MDBUF. If, for example, a master read transfer is requested and obsolete data is in IIC0_MDBUF, the obsolete data would be presented, to the requesting software, as data read by the requested transfer.

18.3.2 IIC0 Slave Data Buffer

The IIC0 Slave Data Buffer (IIC0_SDBUF) is a copy of IIC0_MDBUF. The data contained in the slave buffer is either received from the IIC bus when the IIC interface is addressed as a slave during a write, or is written on the IIC bus when the IIC interface is addressed as a slave during a read operation.

IIC0_SDBUF works in the same way as IIC0_MDBUF, except that IIC0_SDBUF is used only to store data sent or received in slave transfers on the IIC bus. This enables overlapping slave and master transfers on the IIC bus.

Bit assignments for the IIC0_MDBUF and IIC0_SDBUF are identical, as illustrated in Figure 18-5.

Figure 18-5. IIC0 Slave Data Buffer (IIC0_SDBUF)

0		Data bit
1		Data bit
2		Data bit
3		Data bit
4		Data bit
5		Data bit
6		Data bit
7		Data bit

IIC0_SDBUF is cleared (flushed and set to empty) whenever the IIC interface is reset, or IIC0_MDCNTL[FSBD] = 1.

18.3.3 IIC0 Low Master Address Register

The IIC0 Low Master Address (IIC0_LMADR) and IIC0 High Master Address Register (IIC0_HMADR) form addresses that the IIC interface transmits on the IIC bus.

Programming Note: IIC0_HMADR is used only for 10-bit addressing.

When IIC0_CNTL[AMD] = 0 (7-bit addressing), only IIC0_LMADR is written. IIC0_LMADR[A0:A6] form the address transmitted on the IIC bus; IIC0_LMADR[A7] is a don't care. When IIC0_CNTL[AMD] = 1 (10-bit addressing), IIC0_LMADR[A0:A7] form the second byte address transmitted on the IIC bus.

Figure 18-6 illustrates the IIC0_LMADR.

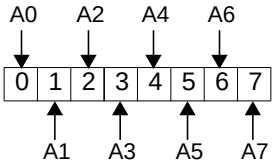


Figure 18-6. IIC0 Low Master Address Register (IIC0_LMADR)

0	A0	Address bit 0
1	A1	Address bit 1
2	A2	Address bit 2
3	A3	Address bit 3
4	A4	Address bit 4

5	A5	Address bit 5	
6	A6	Address bit 6	LSb for 7-bit addresses
7	A7	Address bit 7	LSb for 10-bit addresses; don't care for 7-bit addresses

18.3.4 IIC0 High Master Address Register

IIC0 High Master Address Register (IIC0_HMADR) is not used for 7-bit addressing.

When IIC0_CNTL[AMD] = 1 (10-bit addressing), IIC0_HMADR must be programmed to 0b1111 0*yyx*, where *yy* are the high-order bits of a 10-bit address and *x* is a don't care.

Thus, in 10-bit address mode, IIC0_HMADR[A5:A6] are the two highest-order bits of the 10-bit address and IIC0_HMADR[A7] is a don't care. IIC0_LMADR contains the low-order byte of the 10-bit address.

Figure 18-7 illustrates the IIC0_HMADR.

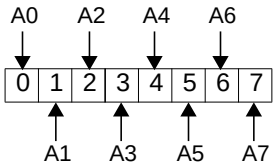


Figure 18-7. IIC0 High Master Address Register (IIC0_HMADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	MSb for 10-bit addresses
6	A6	Address bit 6	Next to MSb for 10-bit addresses
7	A7	Address bit 7	Don't care for 10-bit addresses

18.3.5 IIC0 Control Register

The IIC0 Control Register (IIC0_CNTL) starts and stops IIC interface master transfers on the IIC bus. When a transfer begins, the IIC interface uses the values in IIC0_CNTL to determine the type and size of the transfer.

> **Programming Note:** IIC0_CNTL *must* be the last register programmed. Whenever IIC0_CNTL[PT] = 1, the IIC interface attempts to perform the requested transfer, using values set in other registers. Note that not all IIC registers must be programmed before performing each transfer.

During transfers, and after transfers finish, software can read the IIC0_STS and IIC0_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

Only IIC0_CNTL[PT] is cleared when a requested master transfer is complete; the remaining bits are not affected.

Figure 18-8 illustrates the IIC0_CNTL.

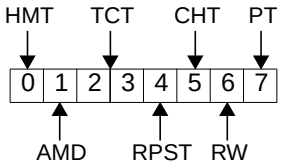


Figure 18-8. IIC0 Control Register (IIC0_CNTL)

0	HMT	Halt Master Transfer 0 Normal transfer operation. 1 Issue Stop signal on the IIC bus as soon as possible to halt master transfer.	If no transfer is in progress, no action is taken. IIC0_CNTL[PT] need not be set. If IIC0_MDCNTL[EINT] = 1, an interrupt is generated.
1	AMD	Addressing Mode 0 Use 7-bit addressing. 1 Use 10-bit addressing.	Does not affect slave transfers.
2:3	TCT	Transfer Count 00 Transfer one byte. 01 Transfer two bytes. 10 Transfer three bytes. 11 Transfer four bytes.	
4	RPST	Repeated Start 0 Normal start operation 1 Use repeated Start function to start transfer.	
5	CHT	Chain Transfer 0 Transfer is only or last transfer. 1 Transfer is one of a sequence of transfers (but not last in sequence).	Completion of a requested transfer causes a Stop signal to be issued on the IIC bus.
6	RW	Read/Write 0 Transfer is a write. 1 Transfer is a read.	
7	PT	Pending Transfer 0 Most recent requested transfer is complete. 1 Start transfer if bus is free.	

Table 18-2 summarizes IIC interface operation for settings of IIC0_CNTL[HMT, RPST, CHT, PT] x is a don't care.

Table 18-2. IIC Response to IIC0_CNTL Field Settings

IIC0_CNTL Fields				Resulting Action on IIC Bus and Inside IIC Interface
HMT	RPST	CHT	PT	
0	x	x	0	No action taken
0	0	1	1	Start, Transfer, ACK on last byte, Pause
0	0	0	1	Start, Transfer, NACK on last byte, Stop
1	x	x	x	NACK on current byte, Stop
0	1	x	1	Start, Transfer, NACK on last byte, Wait

Settings of IIC0_CNTL[HMT, RPST, CHT, PT] result in the following actions.

Start	IIC Start condition generated, if the IIC interface was stopped or waiting.
Stop	IIC Stop condition generated; IIC interface enters the Stop condition.
ACK	IIC Acknowledge condition generated.
NACK	IIC Not Acknowledge condition generated, if performing a read.
Transfer	Requested bytes are transferred.
Pause	IIC interface enters the Pause state.
Wait	IIC interface enters the Wait state.

IIC0_CNTL[HMT] overrides IIC0_CNTL[RPST, CHT].

IIC0_CNTL[RPST, PT] overrides IIC0_CNTL[CHT].

18.3.6 IIC0 Mode Control Register

The IIC0 Mode Control Register (IIC0_MDCNTL) sets the major modes of operation on the IIC bus. In addition, IIC0_MDCNTL can force the data buffers into the empty state.

In typical applications, IIC0_MDCNTL is configured once, during software initialization. Applications providing complex error handling may reconfigure this register more often.

> **Programming Note:** IIC0_CLKDIV must be initialized before IIC0_MDCNTL. IIC0_LSADR and IIC0_HSADR should also be configured before IIC0_MDCNTL.

Note that the IIC hardware does not implement time-out functions on the IIC bus. Such functions must be implemented, in software, by setting IIC0_CNTL[HMT] = 1, or setting IIC0_XTCNTLSS[SRST] = 1.

Regarding IIC0_MDCNTL[HSCL], a "slave not ready" condition occurs during a slave receive operation, if a slave has no free space in its slave data buffer at the start of a write operation, or if the slave data buffer fills during the write. In a slave transmit operation, a slave not ready condition occurs if a slave has no data in its slave data buffer at the start of a read operation, or if the slave data buffer becomes empty during the read.

Using IIC0_MDCNTL[HSCL] to handle slave not ready conditions can affect system performance. A slave holding the IICSCL signal low guarantees data delivery to or from the requesting master, but prevents other masters from performing transfers over the IIC bus. There is no general rule for handling slave not ready conditions; each system has its own requirements.

Figure 18-9 illustrates the IIC0_MDCNTL.

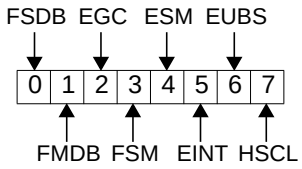


Figure 18-9. IIC0 Mode Control Register (IIC0_MDCNTL)

0	FSDB	Flush Slave Data Buffer 0 Normal operation 1 Set slave data buffer to empty.	Cleared after buffer is emptied.
1	FMDB	Flush Master Data Buffer 0 Normal operation 1 Set master data buffer to empty.	Cleared after buffer is emptied.
2	EGC	Enable General Call 0 Ignore general call on IIC bus. 1 Respond to general call on IIC bus.	IIC0_MDCNTL[ESM] overrides this field; if IIC0_MDCNTL[ESM] = 1, a general call is ignored.
3	FSM	Fast/Standard Mode 0 IIC transfers run at 100 kHz (standard mode). 1 IIC transfers run at 400 kHz (fast mode).	
4	ESM	Enable Slave Mode 0 Slave transfers are ignored. 1 Slave transfers are enabled.	Program IIC0_LSADR and IIC0_HSADR before setting this field.
5	EINT	Enable Interrupt 0 Interrupts are disabled. 1 Enables interrupts for interrupts enabled in IIC0_INTRMSK.	
6	EUBS	Exit Unknown IIC Bus State 0 Normal operation. 1 IIC bus control state machine exits unknown bus state, if in an unknown state.	If the IIC bus control state machine is in a known state, setting IIC0_MDCNTL[EUBS] = 1 has no effect.
7	HSCL	Hold IIC Serial Clock Low 0 If slave is not ready, issue a NACK in response to slave transfer request. 1 If slave is not ready, hold the IICSCL signal low until slave is ready.	This field is used only when in slave mode.

18.3.7 IIC0 Status Register

The IIC0 Status register (IIC0_STS) provides a summary of the state of the IIC interface and the status of any previously requested master transfer.

During and after transfers, software can read the IIC0_STS and IIC0_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

> **Programming Note:** IIC0_STS should be the first register read by an interrupt or error handler routine. IIC0_STS can also be read in a polling loop if software does not use the IIC interrupts.

Software must clear IIC0_STS before requesting another master transfer, except for IIC0_STS[SSS]. Because IIC0_STS[SSS] involves slave transfers, it can remain set.

Figure 18-10 illustrates IIC0_STS.

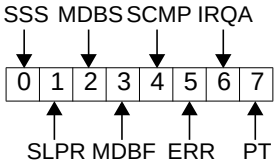


Figure 18-10. IIC0 Status Register (IIC0_STS)

0	SSS	Slave Status Set 0 No slave operations are in progress. 1 Slave operation is in progress.	Read-only; this field is set when any of the following fields are set: IIC0_XTCNTLSS[SRC, SRRS, SWC, SWRS].
1	SLPR	Sleep Request 0 Normal operation. 1 Sleep mode (CPC0_ER[IIC] = 1).	Read-only. The IIC interface is awakened when a start signal is detected on the IIC bus or when the CPC0_ER[IIC] is cleared.
2	MDBS	Master Data Buffer Status 0 Master data buffer is empty. 1 Master data buffer contains data.	Read-only.
3	MDBF	Master Data Buffer Full 0 Master data buffer is not full. 1 Master data buffer is full.	Read-only.
4	SCMP	Stop Complete 0 No request to halt transfer, or master data transfer, is complete. 1 Request to halt transfer, or master data transfer, is complete.	To clear IIC0_STS[SCMP], set IIC0_STS[SCMP] = 1.
5	ERR	Error 0 No error has occurred. 1 One of the following fields is set: IIC0_EXTSTS[LA, ICT, XFRA] = 1.	Read-only.
6	IRQA	IRQ Active 0 No IIC interrupt has been sent to the universal interrupt controller (UIC). 1 An IIC interrupt has been sent to the UIC.	To clear IIC0_STS[IRQA], set IIC0_STS[IRQA] = 1. If IIC0_MDCNTL[EINT] = 0, then IIC0_STS[IRQA] is not set.

7	PT	Pending Transfer 0 No transfer is pending, or transfer is in progress. 1 Transfer is pending.	Read-only.

The Error and Pending Transfer, IIC0_STS[ERR, PT], bit fields indicate the success or failure of the requested transfer. Table 18-3 interprets the transfer status for all possible combinations of the IIC0_STS[ERR,PT] bit fields.

Table 18-3. IIC0_STS[ERR, PT] Decoding

ERR	PT	Status
0	0	Requested transfer completed without errors
0	1	Requested transfer is in progress; no errors were detected
1	0	Requested transfer is complete, but not all data was transferred
1	1	Requested transfer is in progress; but an error was detected

Programming Note: Software should not take any action regarding a master transfer unless all pending transfers are completed, IIC0_STS[PT] = 0.

If an error requires the IIC interface to send a Stop, the Stop Complete bit field is set, IIC0_STS[SCMP] = 1. Note that slave operations should be serviced regardless of the state of a requested master transfer.

IIC0_MDCNTL[EUBS] must be set after a reset before the IIC interface can be placed in sleep mode. The IIC interface is placed in sleep mode by setting the CPC0_ER[IIC] via software. Awaking the IIC interface is possible directly through software by clearing the CPC0_ER[IIC] or indirectly by detecting a Start condition on the IIC bus. When a Start condition is detected, the IIC interface is awakened, and CPC0_SR[IIC] and IIC0_STS[SLPR] are cleared.

The IIC0_STS[MDBS, MDBF] contain the current status of the Master Data Buffer, IIC0_MDBUF. When the IIC0_MDBUF contains data, IIC0_STS[MDBS] is set. When the IIC0_MDBUF is full, IIC0_STS[MDBF] is set.

The state of the IIC0_MDBUF is not instantly recorded by the IIC0_STS[MDBS, MDBF]. The delay depends on the size of the buffer access. For halfword accesses, these fields are valid on the third OPB clock following the transfer. For byte accesses, these fields are valid on the second OPB clock following the transfer.

18.3.8 IIC0 Extended Status Register

The IIC0 Extended Status register (IIC0_EXTSTS) reports additional IIC status.

During and after transfers, software can read the IIC0_STS and IIC0_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

Figure 18-11 illustrates the IIC0_EXTSTS.

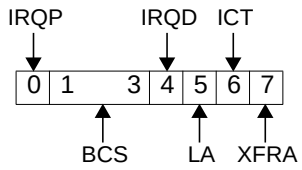


Figure 18-11. IIC0 Extended Status Register (IIC0_EXTSTS)

0	IRQP	IRQ Pending 0 No IRQ is pending. 1 An IRQ is active, another IRQ is on-deck, and another interrupt-generating condition has occurred.	• IIC0_EXTSTS[IRQP] might be set momentarily while an IRQ moves from the Pending to the On-deck state. • An interrupt remains pending, IIC0_EXTSTS[IRQP]=1, until the current on-deck interrupt becomes active, IIC0_EXTSTS[IRQD]=0 and IIC0_STS[IRQA]]=1. • Writing 1 to IIC0_EXTSTS[IRQP] clears the field. • When the IIC interrupt is disabled, IIC0_MDCNTL[IRQP] = 0, IIC0_EXTSTS[IRQP] should be ignored.
1:3	BCS	Bus Control State 000 Unused; if this value is read, a major IIC hardware problem occurred. 001 Slave-selected state; the IIC interface has detected and decoded a slave transfer request on the IIC bus. 010 Slave Transfer state; the IIC interface has detected but has not decoded a slave transfer request on the IIC bus. 011 Master Transfer state; entered after a master transfer request has started on the IIC bus. 100 Free Bus state; the bus is free and no transfer request is pending. 101 Busy Bus state; the bus is busy. 110 Unknown state; value after IIC reset. 111 Unused; if this value is read, a major IIC hardware problem occurred.	Read-only.

4	IRQD	IRQ On-Deck 0 No IRQ is on-deck. 1 An interrupt is active, and another interrupt-generating condition has occurred.	• IIC0_EXTSTS[IRQD] might be set momentarily while an IRQ moves from the On-deck to the Active state. • An interrupt remains on-deck, IIC0_EXTSTS[IRQD] = 1, until the current active interrupt is no longer active, IIC0_STS[IRQA] = 0. • If IIC0_EXTSTS[IRQP] = 1, IIC0_EXTSTS[IRQD] is set on the next OPB clock. • Writing 1 to IIC0_EXTSTS[IRQD] clears the field. • When the IIC interrupt is disabled, IIC0_EXTSTS[IRQP]=0, IIC0_EXTSTS[IRQD] should be ignored.
5	LA	Lost Arbitration 0 Normal operation. 1 Loss of arbitration has ended the requested master transfer.	• If arbitration is lost, any requested master transaction may have terminated prematurely. Read data may be incomplete and not all write data may have been written. • If arbitration is lost during a repeat start, the master may not own the IIC bus.
6	ICT	Incomplete Transfer 0 Normal operation. 1 Some of the bytes of the requested master transfer were not transferred.	For an incomplete transfer, read the transfer count, IIC0_XFRCNT, to determine how bytes were transferred.
7	XFRA	Transfer Aborted 0 No transfer is pending, or transfer is in progress. 1 A requested master transfer was aborted by a NACK during the transfer of the address byte, or was aborted because arbitration was lost. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.	Transfer aborted. When set to a 1, a requested master transfer was aborted by a NOT acknowledge during the transfer of the address byte. It is also set to a 1 when a requested master transfer loses data. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.

IIC0_EXTSTS[IRQP, IRQD] and IIC0_STS[IRQA] provide a FIFO for storing interrupts. A new interrupt is considered pending, and remains pending while an on-deck interrupt is present. Once the on-deck interrupt becomes active, the pending interrupt moves on-deck, and remains on-deck until there is no active interrupt. When the active interrupt is cleared, the on-deck (initially pending) interrupt becomes active.

Programming Note: An active interrupt remains active until software clears it.

IIC0_EXTSTS[BCS] indicates the state of the IIC interface. The field is read-only.

IIC0_EXTSTS[LA, ICT, XFRA] are cleared when IIC0_CNTL[PT] = 1.

Writing 1 to IIC0_EXTSTS[IRQP, IRQD, LA, ICT, XFRA] clears these fields.

18.3.9 IIC0 Low Slave Address Register

The IIC0 Low Slave Address Register (IIC0_LSADR) and IIC0 High Slave Address Register (IIC0_HSADR) program the slave address of the IIC interface. IIC0_HSADR is used only for 10-bit addressing, and is not programmed in 7-bit addressing mode.

When 7-bit addressing is used, IIC0_LSADR is written with the slave address; IIC0_HSADR must be written with zeros. For 7-bit addressing, IIC0_LSADR[A0:A6] contain the address transmitted on the IIC bus; IIC0_LSADR[A7] is a don't care.

When 10-bit addressing is used, IIC0_LSADR[A0:A7] contain the second address byte transmitted on the IIC bus.

Figure 18-6 illustrates the IIC0_LSADR.

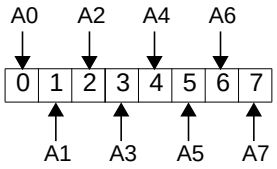


Figure 18-12. IIC0 Low Slave Address Register (IIC0_LSADR)

0	A0	Address bit 0	
1	A1	Address bit 1	
2	A2	Address bit 2	
3	A3	Address bit 3	
4	A4	Address bit 4	
5	A5	Address bit 5	
6	A6	Address bit 6	LSb for 7-bit addresses
7	A7	Address bit 7	LSb for 10-bit addresses; don't care for 7-bit addresses

18.3.10 IIC0 High Slave Address Register

For 7-bit addressing, set IIC0 High Slave Address Register (IIC0_HSADR) to 0.

To enable 10-bit slave addressing, IIC0_HSADR must be programmed to 0b1111 0*yyx*, where *yy* are the high-order bits of a 10-bit address and *x* is a don't care.

> **Programming Note:** IIC0_HSADR is used only for 10-bit addressing, and should be set to 0 for 7-bit addressing mode.

Thus, in 10-bit address mode, IIC0_HSADR[A6:A7] contain the two highest -order bits of the 10-bit address; IIC0_HSADR[A7] is a don't care. IIC0_LSADR contains the low-order byte of the 10-bit address.

Figure 18-13 illustrates the IIC0_HSADR.

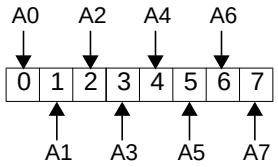


Figure 18-13. IIC0 High Slave Address Register (IIC0_HSADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	MSb for 10-bit addresses
6	A6	Address bit 6	Next to MSb for 10-bit addresses
7	A7	Address bit 7	Don't care for 10-bit addresses

Thus, in 10-bit address mode, bits 0:6 are used to decode the first address byte that was transmitted on the IIC bus, and bit 7 is in a *don't care* state.

18.3.11 IIC0 Clock Divide Register

The IIC0 Clock Divide Register (IIC0_CLKDIV) establishes a reference between the OPB clock and the IIC bus serial clock.

> **Programming Note:** IIC0_CLKDIV must be initialized before IIC0_MDCTRL. Until IIC0_CLKDIV is initialized, all IIC bus activity is ignored.

Figure 18-14 illustrates the IIC0_CLKDIV.

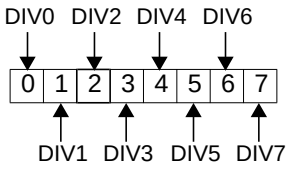


Figure 18-14. IIC0 Clock Divide Register (IIC0_CLKDIV)

0	DIV0	Divisor bit 0
1	DIV1	Divisor bit 1
2	DIV2	Divisor bit 2
3	DIV3	Divisor bit 3
4	DIV4	Divisor bit 4
5	DIV5	Divisor bit 5
6	DIV6	Divisor bit 6

7	DIV7	Divisor bit 7

IIC0_CLKDIV divides PPC405CR's on-chip peripheral bus (OPB) clock to form the base clock for the IIC bus.

Table 22-4 lists the divisor values for several OPB frequency ranges. These divisor values apply for standard and fast mode. Select the divisor value by matching the OPB clock frequency to the corresponding frequency range in Table 22-4. For example, if the OPB clock frequency is 50MHz, select a divisor value of 0x4.

Table 18-4. IIC0 Clock Divide Programming

OPB Frequency Range (MHz)	Divisor Value
20	0x1
$20 < f \leq 30$	0x2
$30 < f \leq 40$	0x3
$40 < f \leq 50$	0x4
$50 < f \leq 60$	0x5

18.3.12 IIC0 Interrupt Mask Register

The IIC0 Interrupt Mask Register (IIC0_INTRMSK) specifies which conditions can generate an IIC interrupt when the IIC interrupt is enabled, IIC0_MDCNTL[EINT]=1.

Figure 18-15 illustrates the IIC0_INTRMSK.

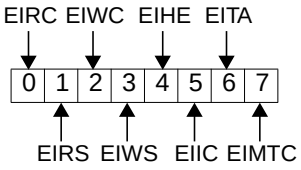


Figure 18-15. IIC0 Interrupt Mask Register (IIC0_INTRMSK)

0	EIRC	Enable IRQ on Slave Read Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave read on the IIC bus. IIC0_XTCNTLSS[SRC] = 1 indicates a Slave Read Complete.
1	EIRS	Enable IRQ on Slave Read Needs Service 0 Disable 1 Enable	The interrupt is activated upon receipt of a slave read on the IIC bus and the slave buffer was empty or went empty and more data was requested on the IIC bus. **Note:** IIC0_XTCNTLSS[SRS] = 1 indicates a Slave Read Needs Service.
2	EIWC	Enable IRQ on Slave Write Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave write on the IIC bus. **Note:** IIC0_XTCNTLSS[SWC] = 1 indicates a Slave Write Compete.

3	EIWS	Enable IRQ on Slave Write Needs Service 0 Disable 1 Enable	The interrupt is activated when the slave buffer becomes full during a slave write on the IIC bus. **Note:** IIC0_XTCNTLSS[SWS] = 1 indicates a Slave Write Needs Service.
4	EIHE	Enable IRQ on Halt Executed 0 Disable 1 Enable	
5	EIIC	Enable IRQ on Incomplete Transfer 0 Disable 1 Enable	
6	EITA	Enable IRQ on Transfer Aborted 0 Disable 1 Enable	
7	EIMTC	Enable IRQ on Requested Master Transfer Complete 0 Disable 1 Enable	

18.3.13 IIC0 Transfer Count Register

The IIC0 Transfer Count Register (IIC0_XFRCNT) reports the number of bytes transferred on the IIC bus during a master or a slave operation.

Figure 18-16 illustrates the IIC0_XFRCNT.

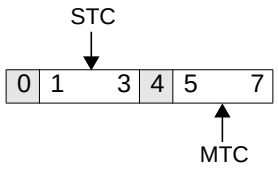


Figure 18-16. IIC0 Transfer Count Register (IIC0_XFRCNT)

0		Reserved
1:3	STC	Slave Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved
4		Reserved

5:7	MTC	Master Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved

IIC0_XFRCNT[MTC] is cleared when there is a pending transfer, IIC0_CNTL[PT] = 1.

IIC0_XFRCNT[STC] is cleared when:

- A slave operation starts on the IIC bus
- Software indicates the slave does not need service by clearing IIC0_XTCNTLSS[SRS] or IIC0_XTCNTLSS[SWS].

18.3.14 IIC0 Extended Control and Slave Status Register

The IIC0 Extended Control and Slave Status Register (IIC0_XTCNTLSS) provides additional control of IIC interface functions and reports the status of slave operations.

Figure 18-17 illustrates the IIC0_XTCNTLSS.

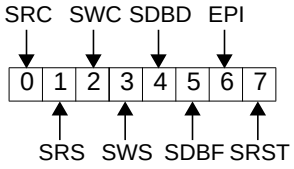


Figure 18-17. IIC0 Extended Control and Slave Status Register (IIC0_XTCNTLSS)

0	SRC	Slave Read Complete 0 Normal operation, or IIC0_MDCNTL[HSCL] = 0, IIC0_SDBUF is empty, and a read operation is in progress. 1 A NACK or Stop condition was received over the IIC bus, or a repeated Start condition ended a read operation.	Check whether the read operation emptied IIC0_SDBUF.

1	SRS	Slave Read Needs Service 0 Normal operation or slave read does not need service. 1 IIC0_SDBUF is empty, and a read operation was requested on the IIC bus. The set condition may also indicate that IIC0_SDBUF is empty due to a slave read and additional data is requested by the master.	1. If IIC0_MDCNTL[HSCL]=0 and IIC0_SDBUF contains no data, the slave will issue a NACK and IIC0_XTCNTLSS[SRS] is set. 2. If IIC0_MDCNTL[HSCL]=0, and IIC0_SDBUF contains data, the slave will send the data. IIC0_XTCNTLSS[SRS] is not set unless the master request additional data. 3. If IIC0_MDCNTL[HSCL]=0, and IIC0_SDBUF contains no data, the slave will hold IICSCL low to indicate the slave is busy. IIC0_XTCNTLSS[SRS] is set until the IIC0_SDBUF is filled. Once filled, IICSCL is released, IIC0_XTCNTLSS[SRS] is cleared, and the slave sends the data. 4. If IIC0_MDCNTL[HSCL]=1, and IIC0_SDBUF contains data, the slave will send the data. IIC0_XTCNTLSS[SRS] is not set unless the master requests additional data.
2	SWC	Slave Write Complete 0 Normal operation or slave write in progress. 1 A Stop signal was received during a write operation, or a repeated Start condition ended a write operation.	
3	SWS	Slave Write Needs Service 0 Normal operation or slave write does not need service. 1 IIC0_SDBUF is full during a slave write.	1. If IIC0_MDCNTL[HSCL] = 1 and IIC0_SDBUF is full, the slave will hold IICSCL low to indicate the slave is busy. IIC0_XTCNTLSS[SWS] is set until IIC0_SDBUF is empty. Once empty, IICSCL is released, IIC0_XTCNTLSS[SWS] is cleared, and the slave receives the data. 2. If IIC0_MDCNTL[HSCL] = 0 and IIC0_SDBUF is full, the slave will issue a NACK and IIC0_XTCNTLSS[SWS] is set.
4	SDBD	Slave Data Buffer Has Data 0 IIC0_SDBUF is empty 1 IIC0_SDBUF contains data	Read-only
5	SDBF	Slave Data Buffer Full 0 IIC0_SDBUF is not full 1 IIC0_SDBUF is full	Read-only
6	EPI	Enable Pulsed IRQ on Transfer Aborted 0 The internal IIC interrupt signal to the UIC remains active until the status is cleared, IIC0_STS[IRQA] =0. 1 The internal IIC interrupt signal to the PPC405CR UIC is active for one OPB clock cycle.	

7	SRST	Soft Reset 0 Normal operation 1 Soft reset

Writing a 1 to IIC0_XTCNTLSS[SRC, SRS, SWC, SWS] clears these fields.

Care must be used when changing IIC0_XTCNTLSS[EPI]. If this field changes (from 1 to 0 or from 0 to 1) while an interrupt is active, the IIC interrupt signal is asserted to the universal interrupt controller (UIC).

The IIC0_XTCNTLSS[SDBD, SDBF] contain the current status of the Slave Data Buffer, IIC0_SDBUF. When the IIC0_SDBUF contains data, IIC0_XTCNTLSS[SDBD] is set. When the IIC0_SDBF is full, IIC0_XTCNTLSS[SDBF] is set.

The state of the IIC0_SDBUF is not instantly recorded by the IIC0_XTCNTL[SDBD, SDBF]. The delay depends on the size of the buffer access. For half-word accesses, these fields are valid on the third OPB clock following the transfer. For byte accesses, these fields are valid on the second OPB clock following the transfer.

If any of the following fields: IIC0_XTCNTLSS[SRC, SRS, SWC, SWS] = 1 and IIC0_MDCNTL[HSCL] = 0; no new slave operations will be accepted over the IIC bus. A NACK is issued until the slave service bits and the slave complete bits are cleared: IIC0_XTCNTLSS[SRC, SRS, SWC, SWS] = 0.

Soft reset, IIC0_XTCNTLSS[SRST], provides a last means of recovery from IIC interface or IIC bus failure. Once enabled, soft reset completely resets the IIC interface. All IIC registers are affected. All transmissions from the IIC interface are terminated. Enabling soft reset during an IIC transmission may improperly terminate the transmission and hang the IIC bus.

18.3.15 IIC0 Direct Control Register

The IIC0 Direct Control Register (IIC0_DIRECTCNTL), which controls and monitors the IIC serial clock (IICSCL) and serial data (IICSDA) signal, is used for error recovery when a malfunction is detected on the IIC interface.

Figure 18-18 illustrates the IIC0_DIRECTCNTL.

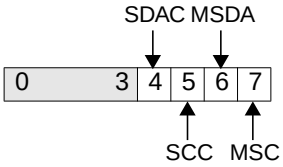


Figure 18-18. IIC0 Direct Control Register (IIC0_DIRECTCNTL)

0:3		Reserved
4	SDAC	IICSDA Output Control Directly controls the IICSDA output. 0 IICSDA is a logic 0 1 IICSDA is a logic 1
5	SCC	IICSCL Output Control Directly controls the IICSCL output 0 IICSCL is a logic 0 1 IICSCL is a logic 1
6	MSDA	Monitor IICSDA Read-only Used to monitor the IICSDA input 0 IICSDA is a logic 0 1 IICSDA is a logic 1
7	MSC	Monitor IICSCL. Used to monitor the IICSCL input. Read-only 0 IICSCL is a logic 0 1 IICSCL is a logic 1

IIC0_DIRECTCNTL[SDAC, SCC] can be written to control the IICSDA and IICSCL signals. When controlling the IICSDA and IICSCL signals directly, the IIC controller must be placed in the reset state, IIC0_XTCNTLSS[SRST] = 1.

IIC0_DIRECTCNTL[MSDA, MSC] are used to verify that IIC0_DIRECTCNTL[SDAC, SCC] were written successfully, and that the IICSCL signals can be controlled. If IIC0_DIRECTCNTL[MSDA, MSC] do not correspond to IIC0_DIRECTCNTL[SDAC, SCC], respectively, toggle IICSCL repeatedly to regain control.

IIC0_DIRECTCNTL[SDAC, SCC, MSDA, MSC] = 1 after a chip or system reset. A Soft Reset, IIC0_XTCNTLSS[SRST] = 1, does not affect the state IIC0_DIRECRCNTL.

18.4 Interrupt Handling

The IIC interface can handle interrupts in two ways. The processor can poll IIC0_STS[SSS, PT]. Alternatively, software can use IIC interface interrupts, IIC0_MDCNTL[EINT] and the interrupt mask bits in the IIC0_INTRMSK control interrupts. IIC0_XTCNTLSS[EPI] controls whether the IIC IRQ is set at a 1-clock pulse or at a constant level.

The IIC controller guarantees that the interrupt is activated after the state of the has been set to ensure that interrupt handlers read stable values from the IIC interface registers. IIC interrupts can

generate interrupt requests to the UIC. If enabled properly in the UIC, an IIC interrupt can interrupt the processor core. See “Interrupt Handling in the Processor Core” on page 9-17 for more information.

Because a master operation can have one interrupt and a slave operation can have two interrupts, the IIC interface can queue up to three interrupts. The current interrupt is referred to as the *active* interrupt. The first interrupt in the queue is referred to as the *on-deck* interrupt; the second queued interrupt is called the *pending* interrupt. The queue holds multiple interrupts until the active interrupt is cleared by writing a 1 to IIC0_STS[IRQA]. When an active interrupt is cleared, the on-deck interrupt becomes the active interrupt and the pending interrupt becomes the on-deck interrupt.

Status associated with an IIC interrupt is immediately set in its corresponding register. Thus, an interrupt handler can see the status for the current and the queued interrupts. This can occur if, for example, the interrupt handler is itself interrupted before reading any status in the IIC registers. If a second or third interrupt occurs before the interrupt handler regains control, the status seen at this time will include status associated with all three interrupting conditions. In this case, the interrupt handler cannot determine the first interrupting condition; all the interrupts will appear to have been merged. Note that if the interrupt handler is intelligent enough to handle and clear all conditions currently set in the status, instead of choosing only one interrupt and servicing it, the routine would not see status for the subsequent interrupts.

A more typical situation involves the case where an interrupt handler has read the IIC status for the active interrupt and a second (on-deck) interrupt occurs. In this case, the status changed after it was first read by the interrupt handler. Because status bits are cleared by writing a 1, no ill effects, such as lost status information, result from this case. To illustrate, consider that the interrupt handler might have read IIC0_XTCNTLSS = 0x10 when the first interrupt occurred. Then, sometime later, the routine clears this status by writing 0x10 back to the IIC0_XTCNTLSS. If the slave write operation completes while the register is read and written, the register will contain a 0x30 when the 0x10 is written. Because writing 0 to a bit has no effect after it is set to 1, the slave write complete status is not lost.

Under certain conditions, the IIC interface merges slave read (write) needs service and slave read (write) complete interrupts into one interrupt. If a slave read (write) needs service interrupt is active, or queued, and a slave read (write) complete interrupt occurs, and IIC0_XTCNTLSS has not yet been read, the two interrupts are merged into a single interrupt. This merge function is performed in the IIC interface logic, and is not under software control.

18.5 General Considerations

1. After a reset, the IIC interface enters the unknown IIC bus state. This state is exited when either activity is seen on the bus or when the exit unknown IIC bus state bit, in the mode control register, is set to a 1. If the IIC interface is being used in a single master system as the master, then the exit unknown IIC bus state bit must be used to force the logic out of the unknown state.
2. Once a byte is written into either the master or slave buffer, a total of four OPB clock periods must occur before the data can be read. Flushing the master or slave buffer also requires four OPB clock periods to complete.

3. The master and slave buffers are actually 4 × 1 byte-wide FIFOs. Exercise care when using master and slave buffers. As an example, consider the case where one byte of data is written on the IIC bus. The data is first written into the master or slave buffer. After four OPB clock cycles the data is placed on the IIC bus. There is no way to verify data in the buffer without disturbing the IIC transaction. The act of verification requires removing the data. If the data is removed, invalid data is placed on the IIC bus.
4. Use care when monitoring the IIC0_XCNTLSS[SDBD] or IIC0_STS[MDBS] to determine when data is present. These bits are set to 1 when the buffer contains data in any stage. Consider the case where the master buffer is empty prior to being loaded with a byte received over the IIC bus. The byte enters the fourth stage of the buffer and the IIC0_STS[MDBS] is set to 1. Stages 1, 2, and 3 do not contain data. Therefore, the data is not available for four OPB clock cycles. Any attempt to prematurely read the data yields invalid data.
5. When responding to a slave needs service request, manage the data first. Read data out of the slave buffer, IIC0_SDBUF, for slave reads or write data into the IIC0_SDBUF for slave writes. Next clear the slave needs service request. For reads, clear IIC0_XTCNTLSS[SRS]. For writes, clear IIC0_XTCNTLSS[SWS]. Last, clear the active interrupt, IIC0_STS[IRQA] =0.
6. There is no timeout function implemented in the IIC interface. If this type of error recovery function is needed, it must be implemented in software.
7. Avoid the situations listed in Section 7.2 of the *Philips Semiconductors I^{2}C Specification,* dated 1995. For your convenience, the section is summarized as follows:

   If multiple masters can be simultaneously involved in a transfer to the same address, or device, then the design of the system must be done in such a way that arbitration between:

   – A repeated Start condition and a data bit does not occur.

   – A Stop condition and a data bit does not occur.

   – A repeated Start condition and a Stop condition does not occur.

One example of a not allowed case would be if one master were to write 1 byte while another device wrote 2 bytes to the same device. In this situation, the first master would issue a Stop while the second master sends the MSb of the second data byte.

Chapter 19. GPIO Operations

This chapter describes the General Purpose I/O (GPIO) controller located on the on-chip peripheral bus (OPB) of the PPC405CR. The GPIO controller allows flexible control of up to 23 multiplexed I/Os with user-defined functions.

19.1 GPIO Controller Overview

The GPIO Controller is an OPB macro that controls up to 23 bidirectional module I/O pins with user-programmable functions. Each of these IOs are multiplexed with other signals to reduce the quantity of module I/O on the PPC405CR package. Control for this multiplexing is performed using the CPC0_CR0 register. See “External Module Signals” on page 19-3 for more information.

The GPIO outputs can be programmed to emulate an open drain driver.

All module I/O inputs are synchronized to the OPBClk before being stored in the Input register.

All registers, except the Input Register, are both read and write accessible. The Input register is read-only. Registers provide direct control of all GPIO Controller functions.

All register bits and core input and output signals maintain a bit-for-bit correspondence. For example, GPIO_Out[20] is controlled by GPIO0_OR Register Bit 20.

19.2 Features

The GPIO Controller has the following features:

- Direct control of all functions from registers programmed via memory-mapped addresses
- Control of 23 bidirectional GPIO module pins
  - Each GPIO output has programmable three-state control
  - Each GPIO output is separately programmable to emulate an open drain driver
  - Each GPIO input is observable from a register bit

19.3 GPIO Interface Signals

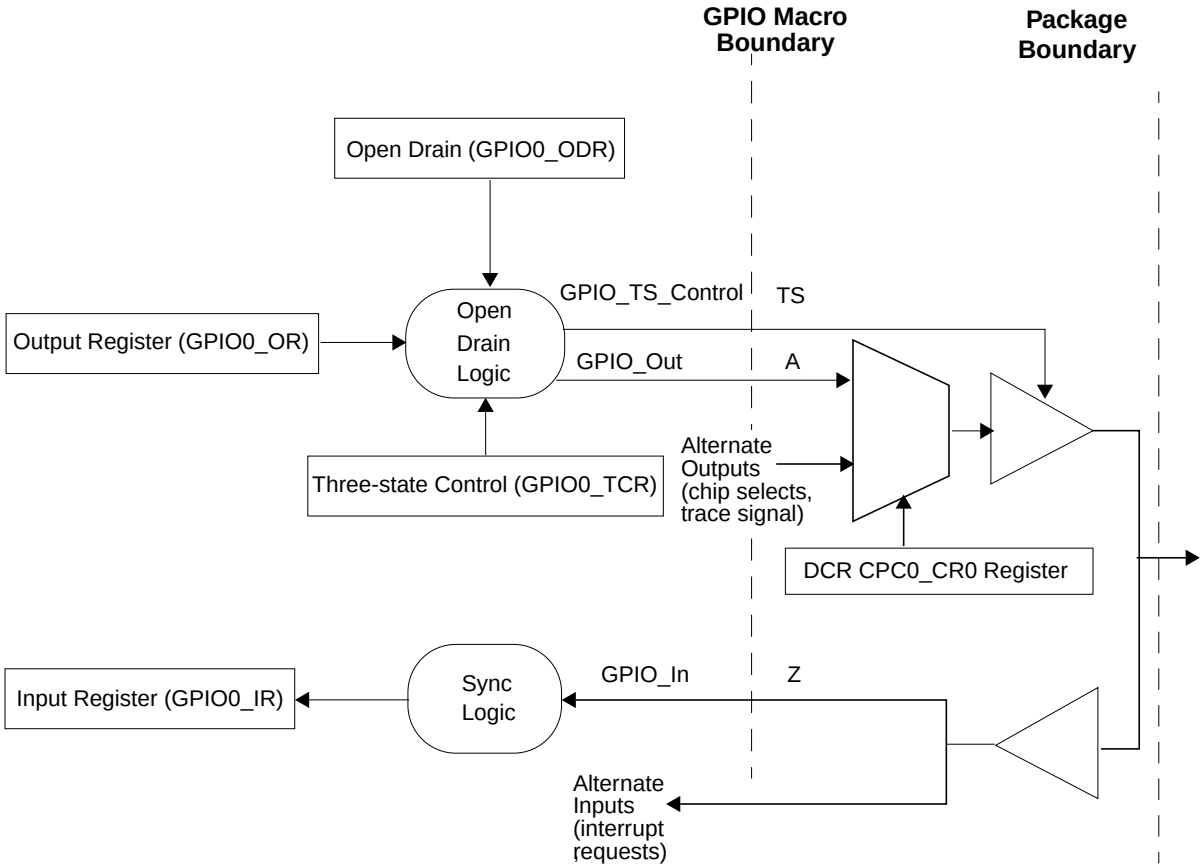


Figure 19-1. GPIO Functional Block Diagram

The following sections discuss the GPIO interface signals.

19.3.1 External Macro Signals

All registers within the GPIO Controller are synchronous to OPBClk, and all GPIO_In inputs are synchronized to OPBClk before being stored.

Table 19-1. Macro I/O Interface Signals

Signal Name	I/O	Function
GPIO_In(0:23)	I	Input to the GPIO Controller directly from Pin Z (Receiver Output). The bi-directional three-state driver books must be in the high impedance state in order to receive valid data as input from off chip.
GPIO_Out(0:23)	O	Data outputs from the GPIO Controller. These signals are designed to tie directly to Pin A (Driver Data Input).
GPIO_TS_Control(0:23)	O	Three-state control outputs from the GPIO Controller macro. These signals are designed to tie directly to each corresponding module I/O Pin TS (Driver Three-State Control). 0 = High Impedance 1 = Active (drive GPIO_Out signal)

19.3.2 External Module Signals

The GPIO signals do not have dedicated module pins. The module pins, used by the GPIO signals, are shared (multiplexed) with other signals. The Chip Control Register 0 (CPC0_CR0) controls the way in which the signal on a shared pin is interpreted. The following sections provide details of the CPC0_CR0 register bit settings and the function of the GPIO registers.

The GPIO signals are multiplexed with the following signals:

- 9 instruction trace signals
- 7 of the 8 EBC chip selects
- 7 external interrupt request inputs

Multiplexing is controlled by the setting of CPC0_CR0[0:31] as shown in Figure 19-2.

Figure 19-2. CPC0_CR0 Bits Controlling GPIO

4	TRE	CPU Trace Enable 0 GPIO1-9 are enabled 1 GPIO1-9 are disabled	Trace interface cannot be used when GPIO is enabled.
5	G10E	GPIO 10 Enable 0 Enable $\overline{\text{PerCS1}}$ as a chip select 1 Enable $\overline{\text{PerCS1}}$ as GPIO10	
6	G11E	GPIO 11 Enable 0 Enable $\overline{\text{PerCS2}}$ as a chip select 1 Enable $\overline{\text{PerCS2}}$ as GPIO11	

Figure 19-2. CPC0_CR0 Bits Controlling GPIO (continued)

7	G12E	GPIO 12 Enable 0 Enable $\overline{\text{PerCS3}}$ as a chip select 1 Enable $\overline{\text{PerCS3}}$ as GPIO12	
8	G13E	GPIO 13 Enable 0 Enable $\overline{\text{PerCS4}}$ as a chip select 1 Enable $\overline{\text{PerCS4}}$ as GPIO13	
9	G14E	GPIO 14 Enable 0 Enable $\overline{\text{PerCS5}}$ as a chip select 1 Enable $\overline{\text{PerCS5}}$ as GPIO14	
10	G15E	GPIO 15 Enable 0 Enable $\overline{\text{PerCS6}}$ as a chip select 1 Enable $\overline{\text{PerCS6}}$ as GPIO15	
11	G16E	GPIO 16 Enable 0 Enable $\overline{\text{PerCS7}}$ as a chip select 1 Enable $\overline{\text{PerCS7}}$ as GPIO16	
12	G17E	GPIO 17 Enable 0 Enable interrupt IRQ0 as an interrupt 1 Enable interrupt IRQ0 as GPIO17	The purpose of GPIO_17_EN through GPIO_23_EN is to isolate the interrupt controller from activity on a shared pin when that pin is being used as a GPIO. For instance, when G17E is set to a 1, IRQ0 at the UIC will always be forced to a zero. **Note:** Setting G17E to a 0 will not prevent GPIO channel 17 (if configured as an output) from creating contention with the off-chip source of the IRQ input. Therefore, be sure to leave the shared GPIO channel disabled when using the pin as an interrupt input.
13	G18E	GPIO 18 Enable 0 Enable interrupt IRQ1 as an interrupt 1 Enable interrupt IRQ1 as GPIO18	
14	G19E	GPIO 19 Enable 0 Enable interrupt IRQ2 as an interrupt 1 Enable interrupt IRQ2 as GPIO19	
15	G20E	GPIO 20 Enable 0 Enable interrupt IRQ3 as an interrupt 1 Enable interrupt IRQ3 as GPIO20	
16	G21E	GPIO 21 Enable 0 Enable interrupt IRQ4 as an interrupt 1 Enable interrupt IRQ4 as GPIO21	
17	G22E	GPIO 22 Enable 0 Enable interrupt IRQ5 as an interrupt 1 Enable interrupt IRQ5 as GPIO22	

Figure 19-2. CPC0_CR0 Bits Controlling GPIO (continued)

18	G23E	GPIO 23 Enable 0 Enable interrupt IRQ6 as an interrupt 1 Enable interrupt IRQ6 as GPIO23

In all cases when a pin is used as a GPIO, it is necessary to configure the appropriate bit in this register as well as the GPIO registers located in the GPIO controller.

19.4 Clock and Power Management

The GPIO Controller macro supports Clock and Power Management. Unconditional Sleep (Class 1) power management is implemented, and is enabled by setting the CPC0_ER[GPIO] bit.

19.5 GPIO Register Overview

Table 19-2 contains a summary of the GPIO registers.

Table 19-2. GPIO Register Summary

Mnemonic	Address	Access	Description
GPIO0_OR	0xEF600700	R/W	GPIO Output
GPIO0_TCR	0xEF600704	R/W	GPIO Three-State Control
GPIO0_ODR	0xEF600718	R/W	GPIO Open Drain
GPIO0_IR	0xEF60071C	R	GPIO Input
Note: GPIO registers are memory-mapped and accessed using **load**/**store** instructions at the register address.			

19.5.1 GPIO Register Reset Values

When a system reset occurs, each register in the GPIO macro, except GPIO0_IR, is reset to 0. All outputs are in the high impedance state. GPIO0_IR is not reset because it is always clocked and always follows the GPIO_In state.

19.5.2 Detailed Register Description

The following sections provide a bit description of the GPIO registers. All registers are accessed from the OPB. The GPIO0_IR register is read-only; all other registers are both read and write accessible.

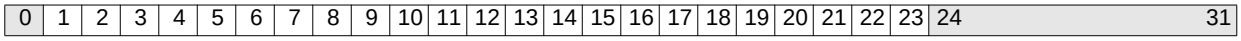


Figure 19-3. GPIO Registers

0		Reserved
1:23		GPIO register bits
24:31		Reserved

19.5.2.1 GPIO Output Register (GPIO0_OR)

The state of each bit in the GPIO0_OR Register may be reflected in the corresponding GPIO Controller macro output signal GPIO_Out.

19.5.2.2 GPIO Three-State Control Register (GPIO0_TCR)

Each bit in the GPIO0_TCR Register controls the corresponding GPIO_TS_Control macro output signal. Each bit in the GPIO0_TCR Register controls the corresponding GPIO_TS_Control macro output signal.

When 0, GPIO_TS_Control forces the module I/O drivers into the high impedance state.

The state of the GPIO_Out signal driving Pin A (Driver Data Input) of the module I/O three-state driver is irrelevant when the driver is in the high impedance state. High impedance takes precedence over data output signals.

19.5.2.3 GPIO Open Drain Register (GPIO0_ODR)

The GPIO0_ODR configures the module I/O three-state driver to emulate open drain drivers on a bit-by-bit basis. This is done by controlling the GPIO_Out and GPIO_TS_Control signals, via the GPIO0_OR and GPIO0_TCR registers, respectively.

When programmed to 1, each bit in the GPIO0_ODR forces the corresponding GPIO_Out signal to 0 and the corresponding GPIO_TS_Control signal to the inverted GPIO_Out signal. In this case, the value of the corresponding bit in the GPIO0_TCR register is ignored. The open drain logic is shown in Table 19-3, “GPIO0_ODR Control Logic,” on page 19-7.

When emulating an open drain driver, the module I/O driver never drives a 1 level. It either drives a 0 level or it is in the high impedance state emulating an open drain 1 level.

Table 19-3. GPIO0_ODR Control Logic

GPIO0_ODR Bit	GPIO0_OR Bit	GPIO_Out Macro Output	GPIO0_TCR Bit	GPIO_TS_Control Macro Output	Module I/O Three-State Driver
0	x	x	0	0	Forced to high impedance state
0	0	0	1	1	Driving 0
0	1	1	1	1	Driving 1
1	0	0	x	1	Driving 0
1	1	0	x	0	Forced to high impedance state

19.5.2.4 GPIO Input Register (GPIO0_IR)

The state of each bit in the GPIO0_IR Register reflects the corresponding GPIO Controller macro input signal GPIO_In. All GPIO_In signals are synchronized to OPBClk before being stored in the GPIO0_IR Register. The GPIO0_IR Register is read-only and does not change during a read access.

All bi-directional three-state drivers must be in the high impedance state in order to receive valid data as input from off chip.

Part V. Reference

Chapter 20. Instruction Set

Descriptions of the PPC405CR 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 exceptions, and provide programming notes.

20.1 Instruction Set Portability

To support embedded real-time applications, the instruction sets of the PPC405CR and other IBM PowerPC 400Series embedded controllers implement the IBM PowerPC Embedded Environment, which is not part of the PowerPC Architecture defined in *The PowerPC Architecture: A Specification for a New Family of RISC Processors*.

Programs using these instructions are not portable to PowerPC implementations that do not implement the IBM PowerPC Embedded Environment.

The PPC405CR implements a number of implementation-specific instructions that are not part of the PowerPC Architecture or the IBM PowerPC Embedded Environment, which are listed in Table 20-1. In the table, the syntax "[**o**]" indicates that an instruction has an "o" form, which updates the XER[SO,OV] fields, and a "non-o" form. The syntax "[**.**]" indicates that an instruction has a "record" form, which updates CR[CR0], and a "non-record" form.

Table 20-1. Implementation-Specific Instructions

dccci	**macchw**[**o**][**.**]	**mfdcr**	**nmacchw**[**o**][**.**]	**rfci**
dcread	**macchws**[**o**][**.**]	**mtdcr**	**nmacchws**[**o**][**.**]	**tlbre**
iccci	**macchwsu**[**o**][**.**]	**mulchw**[**.**]	**nmachhw**[**o**][**.**]	**tlbsx**[**.**]
icread	**macchwu**[**o**][**.**]	**mulchwu**[**.**]	**nmachhws**[**o**][**.**]	**tlbwe**
	machhw[**o**][**.**]	**mulhhw**[**.**]	**nmaclhw**[**o**][**.**]	**wrtee**
	machhws[**o**][**.**]	**mulhhwu**[**.**]	**nmaclhws**[**o**][**.**]	**wrteei**
	machhwsu[**o**][**.**]	**mullhw**[**.**]		
	machhwu[**o**][**.**]	**mullhwu**[**.**]		
	maclhw[**o**][**.**]			
	maclhws[**o**][**.**]			
	maclhwsu[**o**][**.**]			
	maclhwu[**o**][**.**]			

20.2 Instruction Formats

For more detailed information about instruction formats, including a summary of instruction field usage and instruction format diagrams for the PPC405CR, see “Instruction Formats” on page 20-2.

Instructions are four bytes long. Instruction addresses are always word-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 execute all invalid instruction forms without causing an illegal instruction exception.

20.3 Pseudocode

The pseudocode that appears in the instruction descriptions provides a semi-formal language for describing instruction operations.

The pseudocode uses the following notation:

=	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 scope of a loop.
leave	Leave innermost do loop or do loop specified in a leave statement.
n	A decimal number
0xn	A hexadecimal number
0bn	A binary number
FLD	An instruction or register field
FLD_b	A bit in a named instruction or register field
$FLD_{b:b}$	A range of bits in a named instruction or register field
$FLD_{b,b,\ldots}$	A list of bits, by number or name, in a named instruction or register field
REG_b	A bit in a named register
$REG_{b:b}$	A range of bits in a named register
$REG_{b,b,\ldots}$	A list of bits, by number or name, in a named register
REG[FLD]	A field in a named register
REG[FLD, FLD . . .]	A list of fields in a named register
REG[FLD:FLD]	A range of fields in a named register
GPR(r)	General Purpose Register (GPR) r, where $0 \le r \le 31$.
(GPR(r))	The contents of GPR r, where $0 \le r \le 31$.
DCR(DCRN)	A Device Control Register (DCR) specified by the DCRF field in an **mfdcr** or **mtdcr** instruction
SPR(SPRN)	An SPR specified by the SPRF field in an **mfspr** or **mtspr** instruction
TBR(TBRN)	A Time Base Register (TBR) specified by the TBRF field in an **mftb** instruction
GPRs	RA, RB, . . .
(Rx)	The contents of a GPR, where *x* is A, B, S, or T
(RA|0)	The contents of the register RA or 0, if the RA field is 0.
$c_{0:3}$	A four-bit object used to store condition results in compare instructions.
^{n}b	The 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 pseudocode, 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)	The 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.
EA_b	A bit in an effective address.
$EA_{b:b}$	A range of bits in an effective address.
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 0s elsewhere.
instruction(EA)	An instruction operating on a data or instruction cache block associated with an EA.

20.3.1 Operator Precedence

Table 20-2 lists the pseudocode operators and their associativity in descending order of precedence:

Table 20-2. Operator Precedence

Operators	Associativity
REG_b, REG[FLD], function evaluation	Left to right
^nb	Right to left
¬, – (unary minus)	Right to left
×, ÷	Left to right
+, –	Left to right
‖	Left to right
=, ≠, <, >, $\overset{u}{<}$, $\overset{u}{>}$	Left to right
∧, ⊕	Left to right
∨	Left to right
←	None

20.4 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 the CR, see “Condition Register (CR)” on page 3-11. For discussion of XER, see “Fixed Point Exception Register (XER)” on page 3-7.

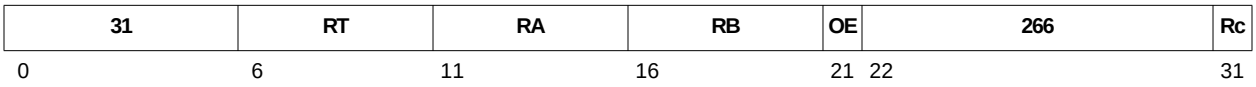
20.5 Alphabetical Instruction Listing

The following pages list the instructions available in the PPC405CR 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 = 1
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.

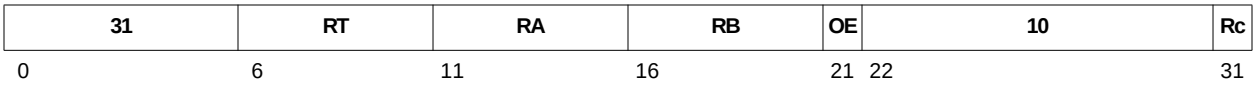
Registers Altered

- 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	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

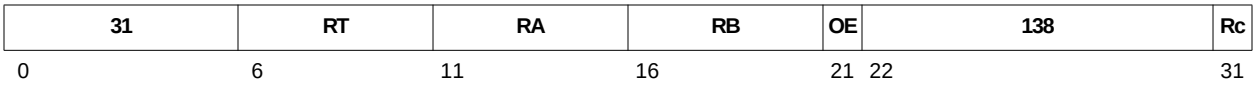
Architecture Note

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] ≯ 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.

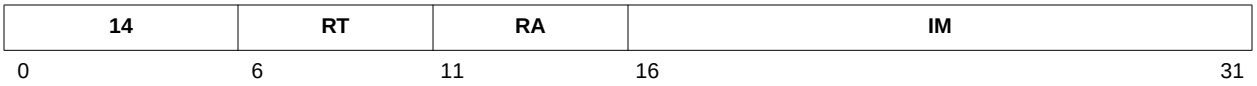
Registers Altered

- 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 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 RT, set RA = 0.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

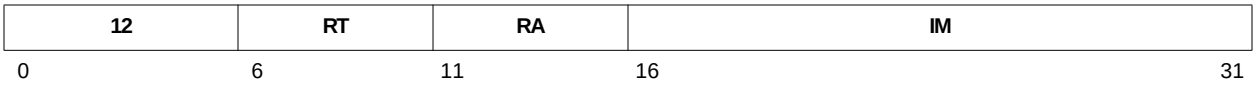
Table 20-3. Extended Mnemonics for addi

Mnemonic	Operands	Function	Other Registers Altered
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**	
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) $\overset{u}{>} 2^{32} - 1$ 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]

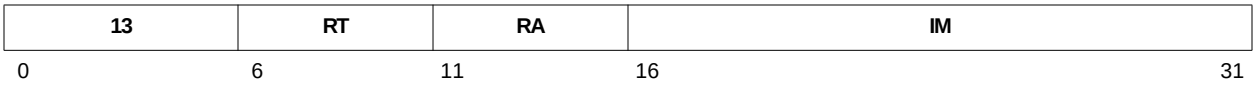
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-4. Extended Mnemonics for addic

Mnemonic	Operands	Function	Other Registers Altered
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. RT, RA, IM



(RT) ← (RA) + EXTS(IM)
if (RA) + EXTS(IM) ⩾ $2^{32} - 1$ 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.

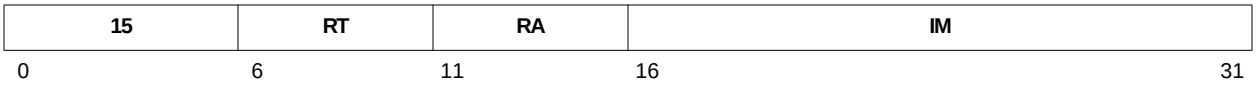
Table 20-5. Extended Mnemonics for addic.

Mnemonic	Operands	Function	Other Registers Altered
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
```

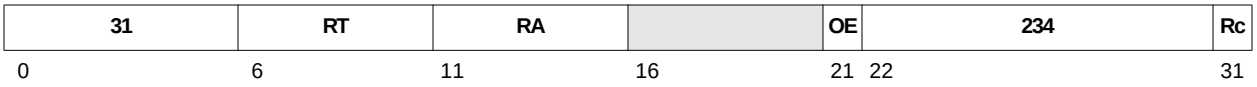
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-6. Extended Mnemonics for addis

Mnemonic	Operands	Function	Other Registers Altered
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	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) $\leftarrow$ (RA) + XER[CA] + (–1)
if (RA) + XER[CA] + 0xFFFF FFFF $\overset{u}{>}$ 2^{32} – 1 then
  XER[CA] $\leftarrow$ 1
else
  XER[CA] $\leftarrow$ 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

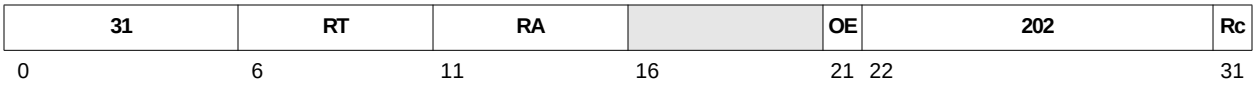
Architecture Note

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) ← (RA) + XER[CA]
if (RA) + XER[CA] $\overset{U}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 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

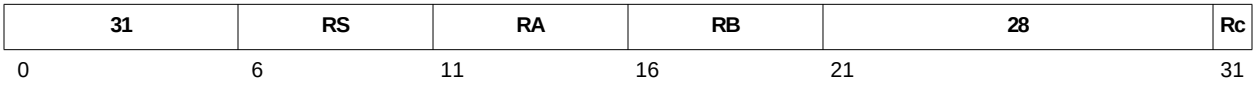
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

and	RA, RS, RB	Rc=0
and.	RA, RS, RB	Rc=1



(RA) $\leftarrow$ (RS) $\wedge$ (RB)

The contents of register RS are ANDed 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

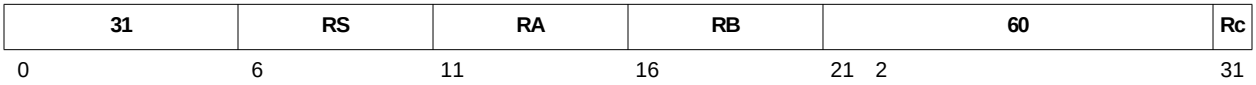
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 are ANDed with the ones complement of the contents of register RB; the result is placed into register RA.

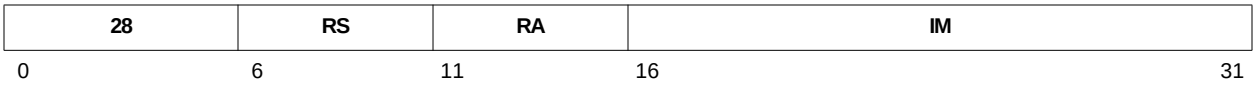
Registers Altered

- 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. 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.**.

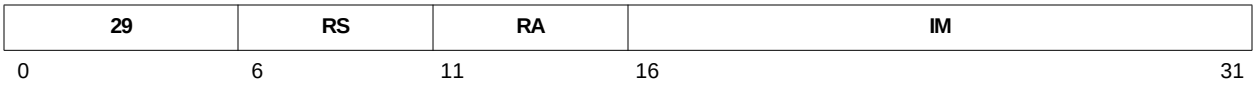
Architecture Note

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

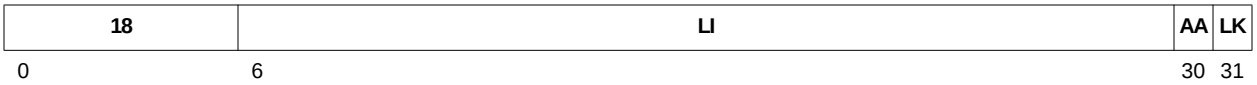
The **andis.** instruction can test whether any of the 16 most-significant bits in a GPR are 1-bits.

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	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

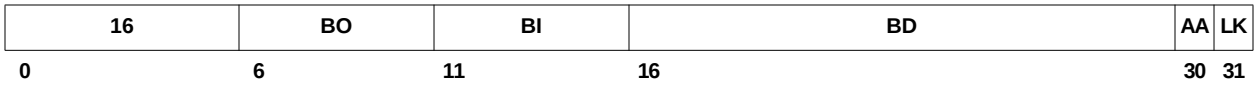
Architecture Note

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 decrements.

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 “Branch Prediction” on page 3-32 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 20-7. Extended Mnemonics for bc, bca, bcl, bcla

Mnemonic	Operands	Function	Other Registers Altered
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.

Table 20-7. Extended Mnemonics for bc, bca, bcl, bcla (continued)

Mnemonic	Operands	Function	Other Registers Altered
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.
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

Table 20-7. Extended Mnemonics for bc, bca, bcl, bcla (continued)

Mnemonic	Operands	Function	Other Registers Altered
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
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.

Table 20-7. Extended Mnemonics for bc, bca, bcl, bcla (continued)

Mnemonic	Operands	Function	Other Registers Altered
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.
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.

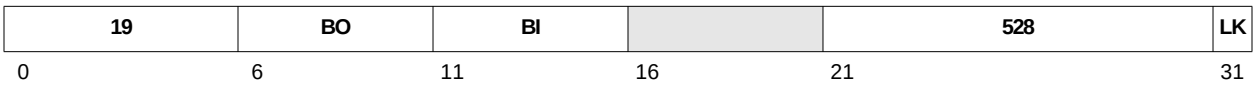
Table 20-7. Extended Mnemonics for bc, bca, bcl, bcla (continued)

Mnemonic	Operands	Function	Other Registers Altered
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 “Branch Prediction” on page 3-32 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 20-8. Extended Mnemonics for bcctr, bcctrl

Mnemonic	Operands	Function	Other Registers Altered
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.

Table 20-8. Extended Mnemonics for bcctr, bcctrl (continued)

Mnemonic	Operands	Function	Other Registers Altered
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.
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.

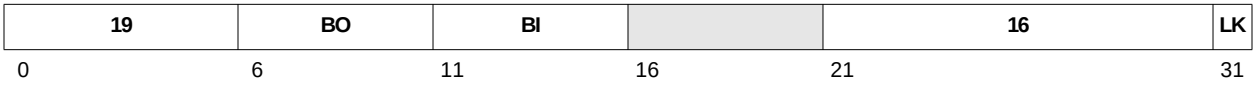
Table 20-8. Extended Mnemonics for bcctr, bcctrl (continued)

Mnemonic	Operands	Function	Other Registers Altered
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 “Branch Prediction” on page 3-32 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 20-9. Extended Mnemonics for bclr, bclrl

Mnemonic	Operands	Function	Other Registers Altered
blr		Branch unconditionally to address in LR. *Extended mnemonic for* **bclr 20,0**	
blrl		*Extended mnemonic for* **bclrl 20,0**	(LR) ← CIA + 4.

Table 20-9. Extended Mnemonics for bclr, bclrl (continued)

Mnemonic	Operands	Function	Other Registers Altered
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.

Table 20-9. Extended Mnemonics for bclr, bclrl (continued)

Mnemonic	Operands	Function	Other Registers Altered
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.
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.

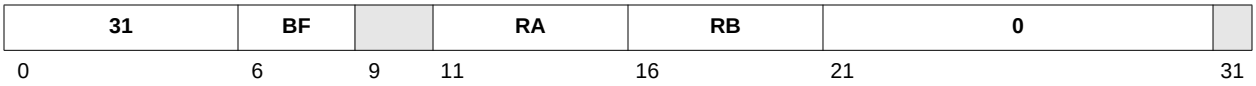
Table 20-9. Extended Mnemonics for bclr, bclrl (continued)

Mnemonic	Operands	Function	Other Registers Altered
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.
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[CR*n*] 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 PPC405CR, use of the extended mnemonic **cmpw BF,RA,RB** is recommended.

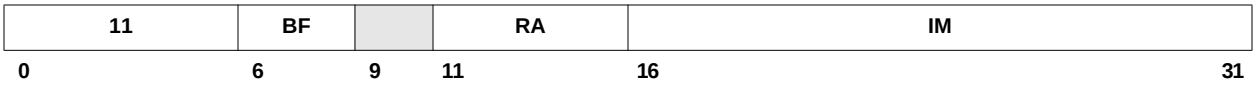
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-10. Extended Mnemonics for cmp

Mnemonic	Operands	Function	Other Registers Altered
cmpw	[BF,] RA, RB	Compare Word; use CR0 if BF is omitted. *Extended mnemonic for* **cmp BF,0,RA,RB**	

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] $\leftarrow c_{0:3}$

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[CR*n*] 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 the PPC405CR, use of the extended mnemonic **cmpwi BF,RA,IM** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

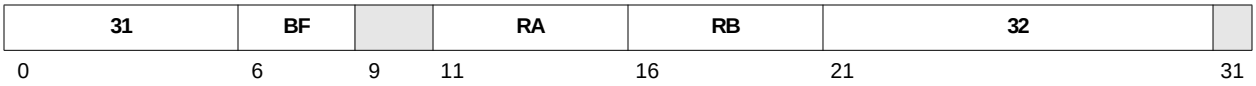
Table 20-11. Extended Mnemonics for cmpi

Mnemonic	Operands	Function	Other Registers Altered
cmpwi	[BF,] RA, IM	Compare Word 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) $\overset{u}{<}$ (RB) then $c_0 \leftarrow 1$
if (RA) $\overset{u}{>}$ (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[CR*n*] 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 PPC405CR, use of the extended mnemonic **cmplw BF,RA,RB** is recommended.

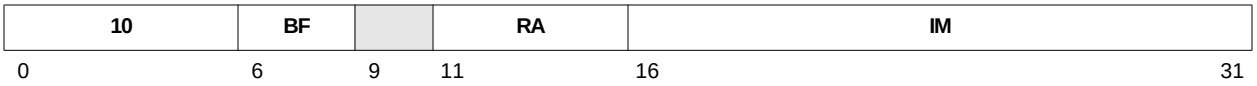
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-12. Extended Mnemonics for cmpl

Mnemonic	Operands	Function	Other Registers Altered
cmplw	[BF,] RA, RB	Compare Logical Word. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpl BF,0,RA,RB**	

cmpli BF, 0, RA, IM



$c_{0:3} \leftarrow {}^{4}0$
if (RA) $\overset{u}{<}$ ($^{16}0$ || IM) then $c_0 \leftarrow 1$
if (RA) $\overset{u}{>}$ ($^{16}0$ || 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[CR*n*] 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 PPC405CR, use of the extended mnemonic **cmplwi BF,RA,IM** is recommended.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

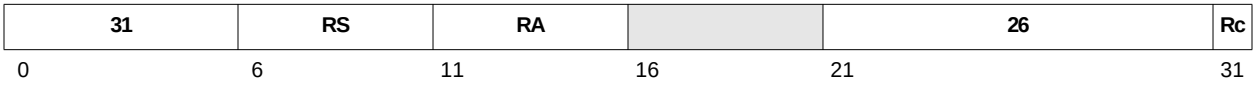
Table 20-13. Extended Mnemonics for cmpli

Mnemonic	Operands	Function	Other Registers Changed
cmplwi	[BF,] RA, IM	Compare Logical Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpli BF,0,RA,IM**	

cntlzw

Count Leading Zeros Word

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

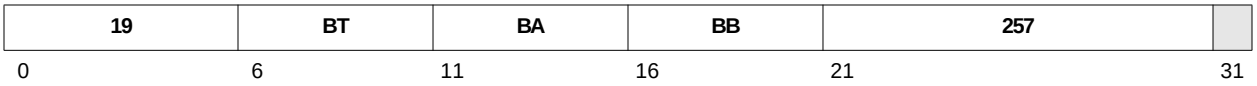
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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

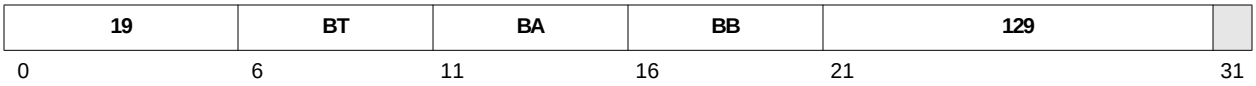
Architecture Note

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

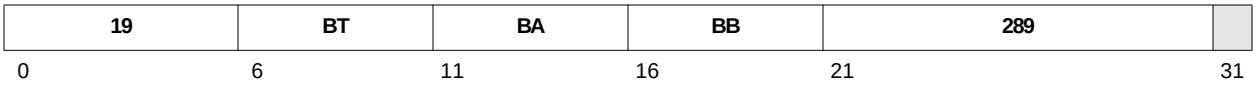
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

creqv 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.

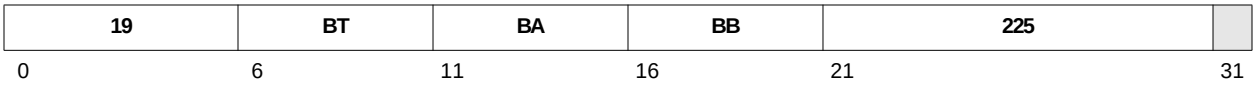
Table 20-14. Extended Mnemonics for creqv

Mnemonic	Operands	Function	Other Registers Altered
crset	bx	CR 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

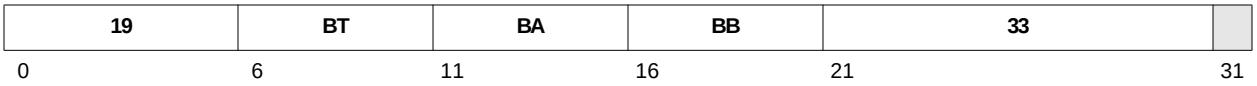
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

crnor 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.

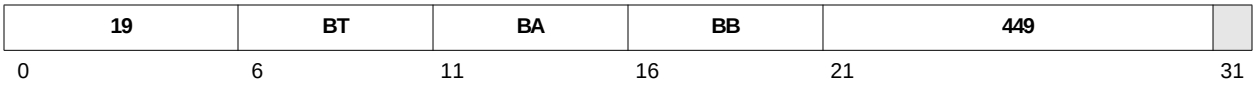
Table 20-15. Extended Mnemonics for crnor

Mnemonic	Operands	Function	Other Registers Altered
crnot	bx, by	CR 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

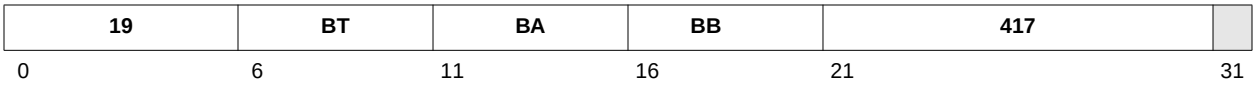
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-16. Extended Mnemonics for cror

Mnemonic	Operands	Function	Other Registers Altered
crmove	bx, by	CR move. *Extended mnemonic for* **cror bx,by,by**	

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

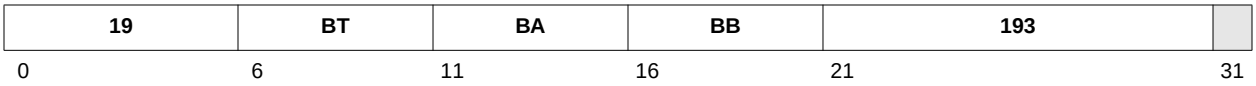
Architecture Note

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

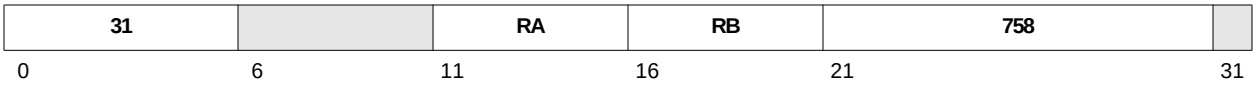
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-17. Extended Mnemonics for crxor

Mnemonic	Operands	Function	Other Registers Altered
crclr	bx	Condition register clear. *Extended mnemonic for* **crxor bx,bx,bx**	

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 cachable and non-write-through, the data in the cache block is architecturally undefined. For the PPC405CR, 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 cachable 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-cachable, 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 be left unmodified. Alternatively, the data block at the EA can be undefined in the data cache and in main storage. For the PPC405CR, 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 PPC405CR, 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 **dcba** 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 will soon be stored to 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.

dcba

Data Cache Block Allocate

Exceptions

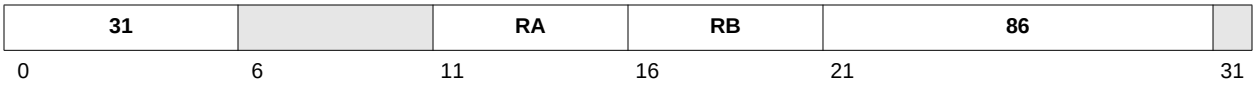
This instruction is considered a “store” with respect to data storage exceptions. However, this instruction does not cause data storage exceptions or data TLB-miss exceptions. If conditions occur that would otherwise cause such exceptions, **dcba** is treated as a no-op.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See “Data Storage Interrupt” on page 9-31.

Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

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 cachable.

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. See “Data Storage Interrupt” on page 9-31.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

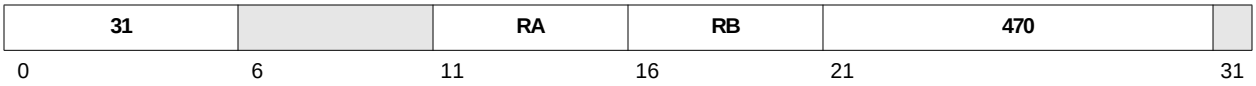
Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

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 cachable. 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

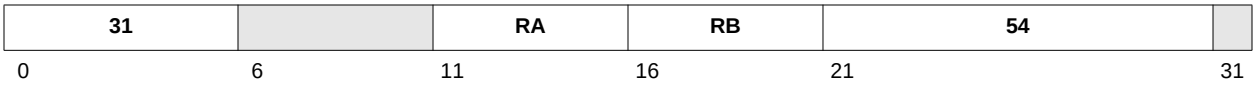
This instruction is considered a “store” with respect to data storage exceptions. See “Data Storage Interrupt” on page 9-31.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

Architecture Note

This instruction is part of the IBM PowerPC Embedded Operating Environment.

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 cachable.

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 “Data Storage Interrupt” on page 9-31.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

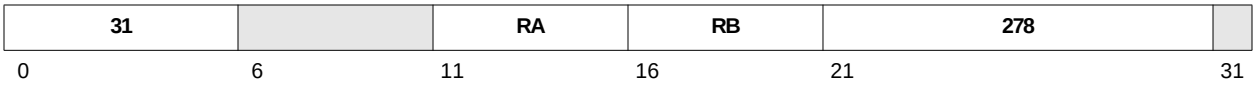
Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

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 cachable, 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-cachable, 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

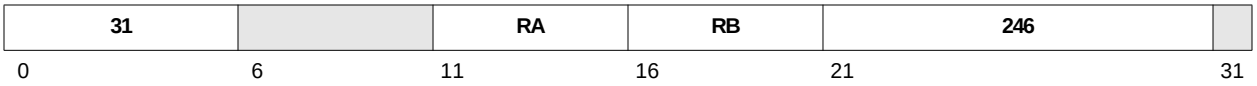
This instruction is considered a “load” with respect to data storage exceptions. See “Data Storage Interrupt” on page 9-31.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

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 cachable, the data block is loaded into the data cache.

If the EA is marked as non-cachable, 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 “Data Storage Interrupt” on page 9-31.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

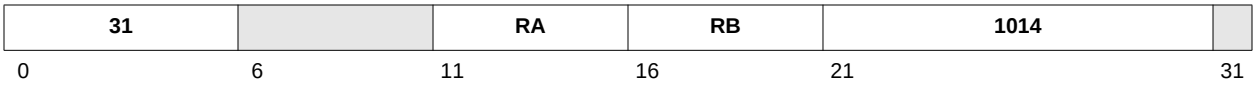
Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

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 cachable 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 cachable 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-cachable, 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 **dcbz** 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-cachable, 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-cachable, 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 cachable; 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-cachable or as write-through.

This instruction is considered a “store” with respect to data storage exceptions. See “Data Storage Interrupt” on page 9-31.

This instruction is considered a “store” with respect to data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

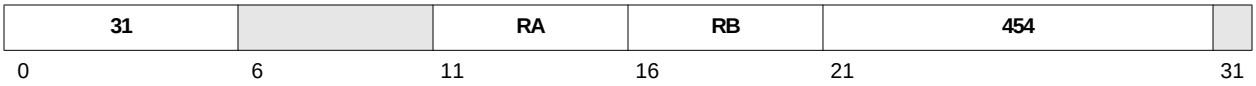
Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

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 $EA_{20:26}$ 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 cachable.

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. Cachability can then be enabled.

Exceptions

See “Access Protection for Cache Control Instructions” on page 5-15.

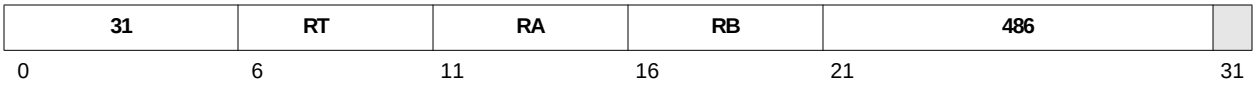
The execution of an **dccci** instruction can cause a data TLB miss exception, at the specified EA, regardless of the non-specific intent of that EA.

This instruction does not cause data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

Architecture Note

This instruction is implementation-specific and may not be portable to other implementations.

dcread RT, RA, RB



EA ← (RA|0) + (RB)
if ((CCR0[CIS] = 0) ∧ (CCR0[CWS] = 0)) then (RT) ← (d-cache data, way A)
if ((CCR0[CIS] = 0) ∧ (CCR0[CWS] = 1)) then (RT) ← (d-cache data, way B)
if ((CCR0[CIS] = 1) ∧ (CCR0[CWS] = 0)) then (RT) ← (d-cache tag, way A)
if ((CCR0[CIS] = 1) ∧ (CCR0[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 $EA_{19:26}$, where *m* is the number of bits in the cache tag. The cache information is read into register RT.

If CCR0[CIS] = 0, the information is a word of data cache array data from the addressed congruence class. The word is specified by $EA_{27:29}$ ($EA_{0:19}$ and $EA_{30:31}$ are ignored). If $EA_{30:31}$ are not 00, an alignment exception occurs. If CCR0[CWS] = 0, the data is from the A-way; otherwise; the data is from the B-way.

If CCR0[CIS] = 1, the information is a cache tag from the addressed congruence class ($EA_{0:19}$ and $EA_{30:31}$ are ignored). If CCR0[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 shown:

0:19	TAG	Cache Tag
20:25		Reserved
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

dcread

Data Cache Read

Invalid Instruction Forms

- Reserved fields

Programming Note

Execution of this instruction is privileged.

Exceptions

If EA is not word-aligned, an alignment exception occurs.

This instruction is considered a “load” with respect to data storage exceptions, but cannot cause a data storage exception. See “Access Protection for Cache Control Instructions” on page 5-15.

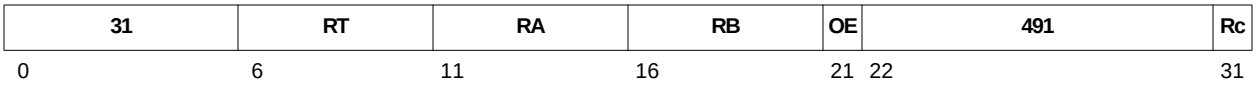
The execution of an **dcread** instruction can cause a data TLB miss exception, at the specified EA, regardless of the non-specific intent of that effective address.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions. See “Debug Interrupt” on page 9-39.

Architecture Note

This instruction is implementation-specific and may not be portable to other implementations.

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 ($n \div 0$), 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.

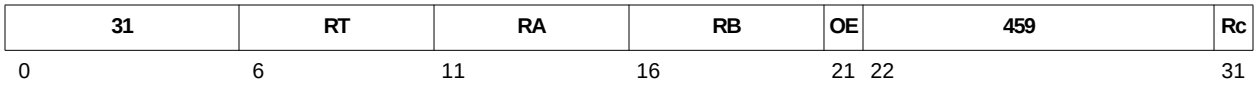
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

divwu

Divide Word 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) $\leftarrow$ (RA) $\div$ (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 $\times$ divisor) + remainder

If an attempt is made to perform ($n \div 0$), 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

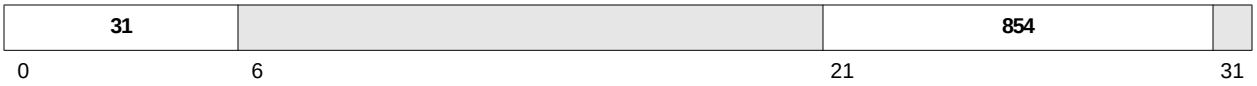
```
divwu   RT,RA,RB        # RT = quotient
mullw   RT,RT,RB        # RT = quotient × divisor
subf    RT,RT,RA        # RT = remainder
```

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



The **eieio** instruction ensures that all loads and stores preceding **eieio** complete with respect to main storage before any loads and stores following **eieio** 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 PPC405CR, 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 that corresponds to the desired behavior.

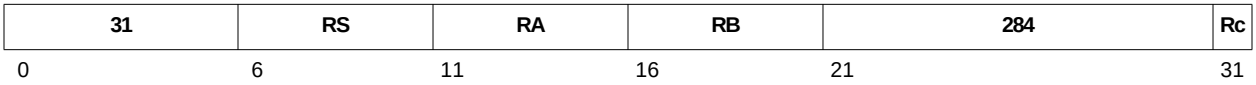
Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

eqv

Equivalent

eqv	RA, RS, RB	Rc=0
eqv.	RA, RS, RB	Rc=1



(RA) $\leftarrow$ $\neg$((RS) $\oplus$ (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.

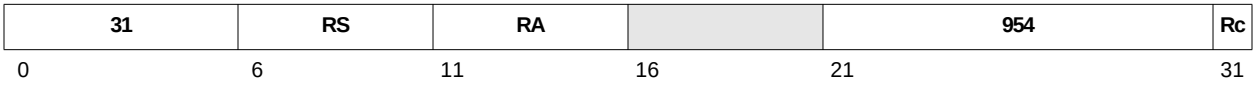
Registers Altered

- 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	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

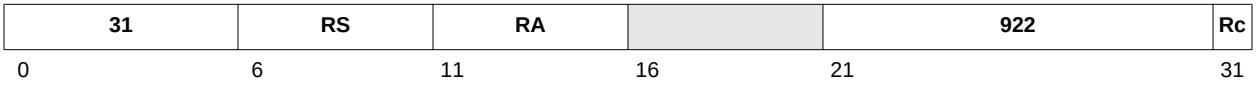
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

extsh

Extend Sign Halfword

extsh	RA, RS	Rc=0
extsh.	RA, RS	Rc=1



(RA) $\leftarrow$ EXTS(RS)$_{16:31}$

The least significant halfword 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

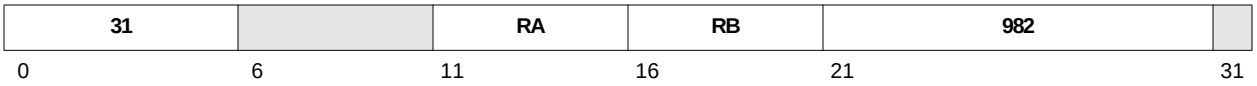
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

icbi 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 cachable 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, cachability 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. See “Data Storage Interrupt” on page 9-31.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions.

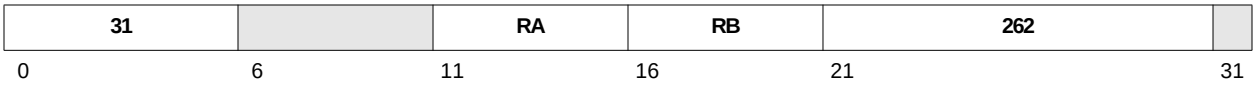
Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

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 cachable, 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-cachable, 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

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, cachability for the effective address of the operand of instruction cache operations is determined by the ICCR, not the DCCR.

Exceptions

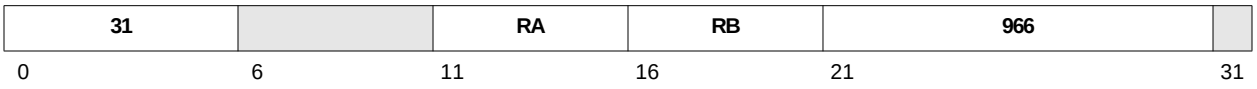
Instruction storage exceptions and instruction-side TLB miss exceptions are associated with instruction *fetching*, not with instruction execution. Exceptions occurring during *execution* of instruction cache operations cause data storage and data TLB miss exceptions.

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 data storage exceptions. This instruction is also considered a “load” with respect to data address compare (DAC) debug exceptions.

Architecture Note

This instruction is part of the IBM PowerPC Embedded Operating Environment.



EA ← (RA|0) + (RB)
ICCCI(ICU cache array)

This instruction invalidates the entire ICU cache array. The EA is not used; previous implementations have used the EA for protection checks. The instruction form is maintained for software and tool compatibility.

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. Cachability can then be enabled.

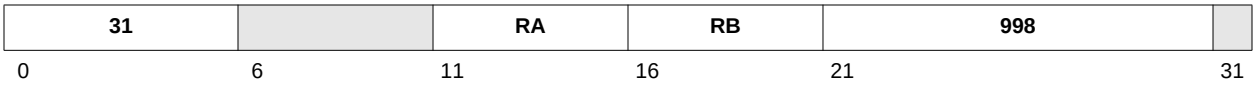
Architecture Note

This instruction is implementation-specific and may not be portable to other implementations.

icread

Instruction Cache Read

icread RA, RB



EA ← (RA|0) + (RB)
if ((CCR0[CIS] = 0) ∧ (CCR0[CWS] = 0)) then (ICDBDR) ← (i-cache data, way A)
if ((CCR0[CIS] = 0) ∧ (CCR0[CWS] = 1)) then (ICDBDR) ← (i-cache data, way B)
if ((CCR0[CIS] = 1) ∧ (CCR0[CWS] = 0)) then (ICDBDR) ← (i-cache tag, way A)
if ((CCR0[CIS] = 1) ∧ (CCR0[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 $EA_{19:26}$. 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 CCR0[CIS] = 0, the information is a word of instruction cache data from the addressed line. The word is specified by $EA_{27:29}$ ($EA_{0:21}$ and $EA_{30:31}$ are ignored). If CCR0[CWS] = 0, the data is from the A-way, otherwise from the B-way.

If (CCR0[CIS] = 1), the information is a cache tag from the addressed congruence class ($EA_{0:21}$ and $EA_{28:31}$ are ignored). If (CCR0[CWS] = 0), the tag is from the A-way, otherwise from the B-way.

Instruction cache tag information is placed in the ICDBDR as shown:

0:21	TAG	Cache Tag
22:26		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, cachability 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 **icread** 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” and cannot cause a data storage exception.

This instruction is considered a “load” with respect to data address compare (DAC) debug exceptions, but will not cause DAC debug events.

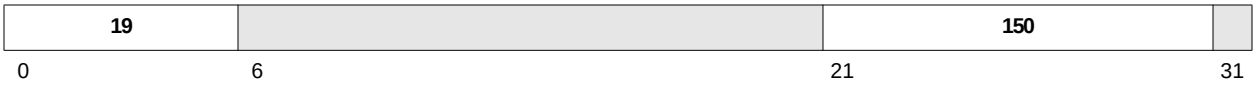
Architecture Note

This instruction is implementation-specific and may not be portable to other implementations.

isync

Instruction Synchronize

isync



The **isync** instruction is a context synchronizing instruction.

isync provides an ordering function for the effects of all instructions executed by the processor. Executing **isync** insures that all instructions preceding the **isync** instruction execute before **isync** completes, except that storage accesses caused by those instructions need not have completed.

No subsequent instructions are initiated by the processor until **isync** 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 **isync**.

isync 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 “Synchronization” on page 3-39.

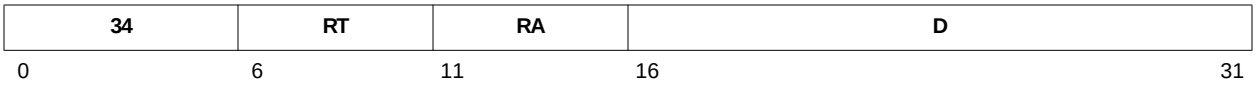
The following code example illustrates the necessary steps for self-modifying code. This example assumes that addr1 is both data and instruction cachable.

```
stw     regN, addr1     # 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 IBM PowerPC Embedded Virtual Environment.

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

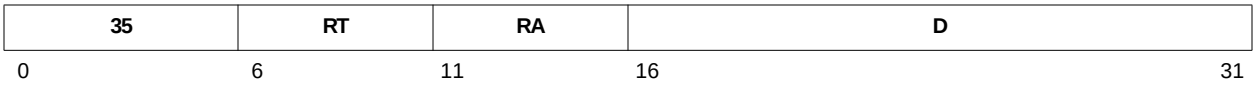
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzu

Load Byte and Zero with Update

lbzu RT, D(RA)



EA $\leftarrow$ (RA|0) + EXTS(D)
(RA) $\leftarrow$ EA
(RT) $\leftarrow$ $^{24}0$ || 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

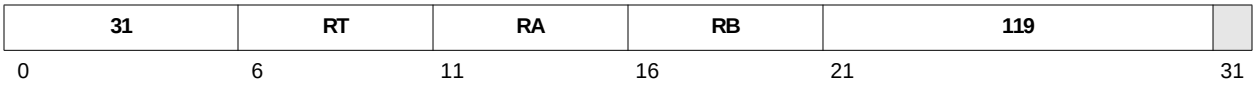
Invalid Instruction Forms

- RA=RT
- RA=0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzux 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

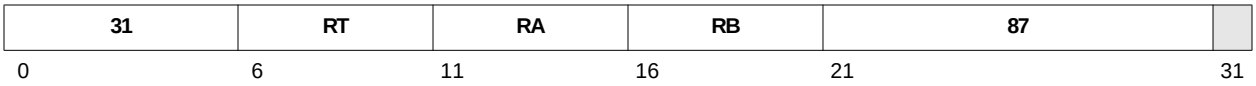
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lbzx

Load Byte and Zero Indexed

lbzx RT,RA, RB



EA $\leftarrow$ (RA|0) + (RB)
(RT) $\leftarrow$ $^{24}0$ || 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

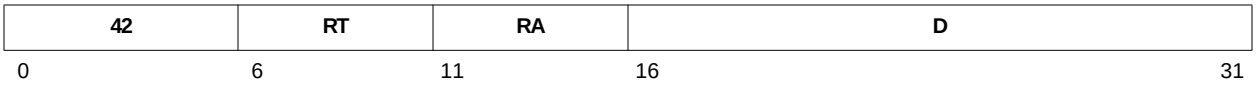
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lha 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 halfword at the EA is sign-extended to 32 bits and placed into register RT.

Registers Altered

- RT

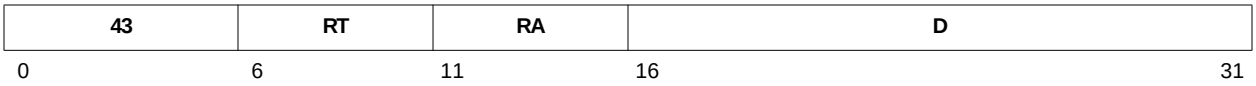
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhau

Load Halfword 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 halfword at the EA is sign-extended to 32 bits and placed into register RT.

Registers Altered

- RA
- RT

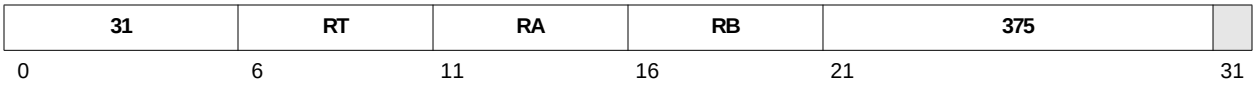
Invalid Instruction Forms

- RA = RT
- RA = 0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 halfword 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

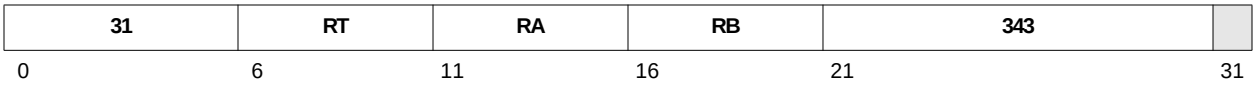
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhax

Load Halfword 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 halfword 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

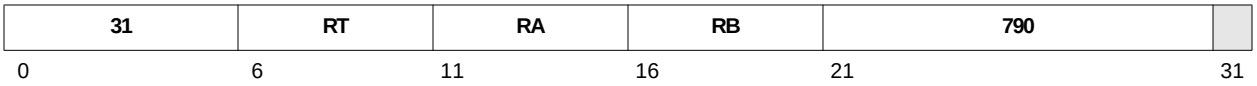
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 halfword at the EA is byte-reversed. The resulting halfword 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

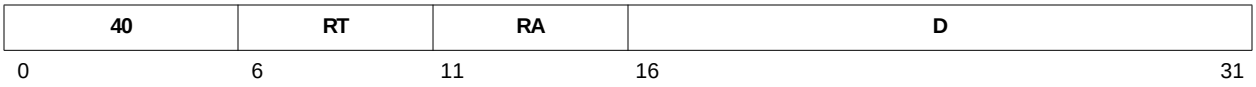
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhz

Load Halfword 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 halfword at the EA is extended to 32 bits by concatenating 16 0-bits to its left. The result is placed into register RT.

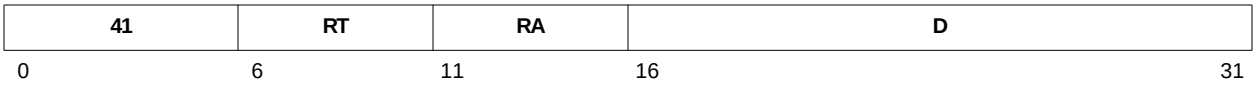
Registers Altered

- RT

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 halfword 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

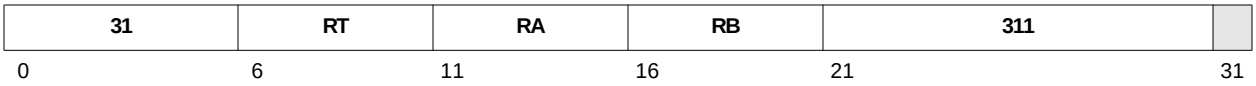
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzux

Load Halfword and Zero with Update Indexed

lhzux RT, RA, RB



EA $\leftarrow$ (RA|0) + (RB)
(RA) $\leftarrow$ EA
(RT) $\leftarrow$ $^{16}0$ || 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 halfword 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

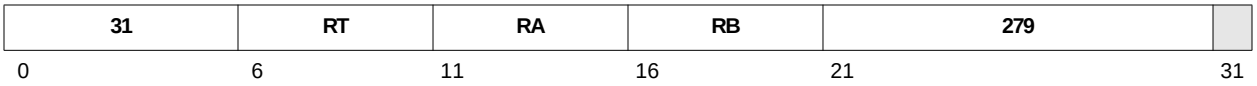
Invalid Instruction Forms

- Reserved fields
- RA = RT
- RA = 0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lhzx RT, RA, RB



EA $\leftarrow$ (RA|0) + (RB)
(RT) $\leftarrow$ $^{16}0$ || 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 halfword 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

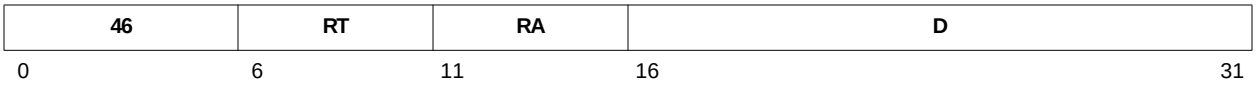
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lmw

Load Multiple Word

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 word which would have been placed into register RA is discarded.

Registers Altered

- RT through GPR(31).

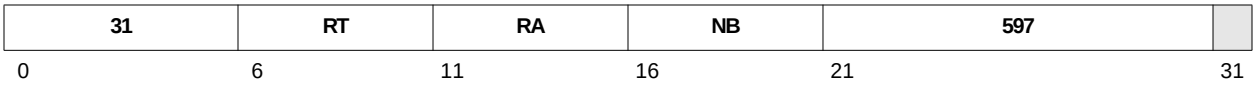
Invalid Instruction Forms

- RA is in the range of registers to be loaded, including the case RA = RT = 0.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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), 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}, 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.

lswi

Load String Word Immediate

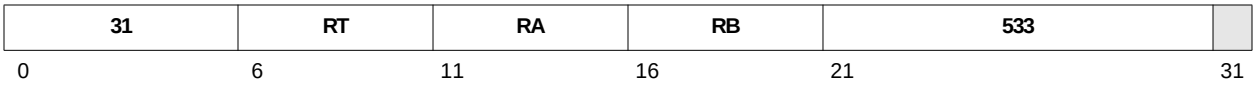
Invalid Instruction Forms

- Reserved fields
- RA is in the range of registers to be loaded
- RA = RT = 0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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.

lswx

Load String Word Indexed

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 will not occur. Data storage exceptions and alignment exceptions are examples of precise data exceptions.

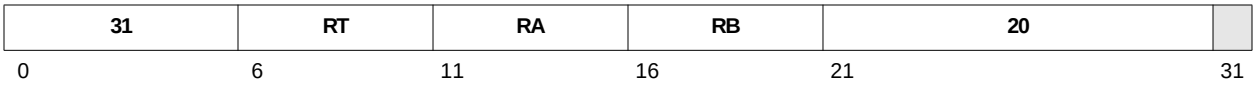
However, the PowerPC Architecture makes no statement regarding imprecise exceptions related to **lswx** with XER[TBC] = 0. The PPC405CR generates an imprecise exception (machine check) on this instruction when all of the following conditions are true:

- The instruction passes all protection bounds checking
- The address is cachable
- 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

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 word 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 word-aligned.

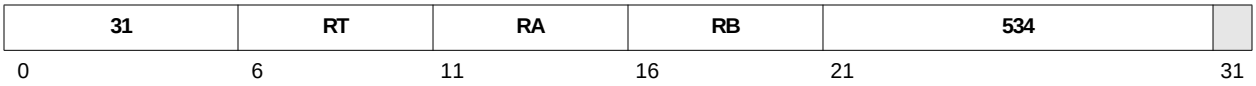
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwbrx

Load Word 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 word 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 word is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

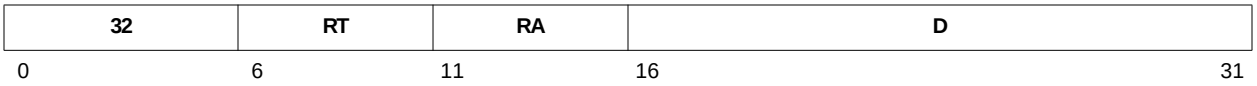
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 word at the EA is placed into register RT.

Registers Altered

- RT

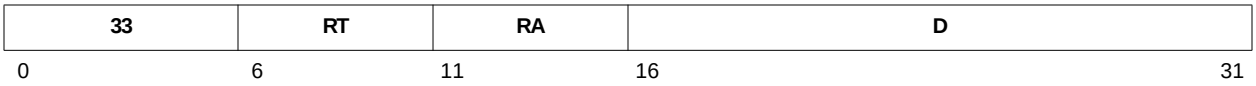
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzu

Load Word 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 word at the EA is placed into register RT.

Registers Altered

- RA
- RT

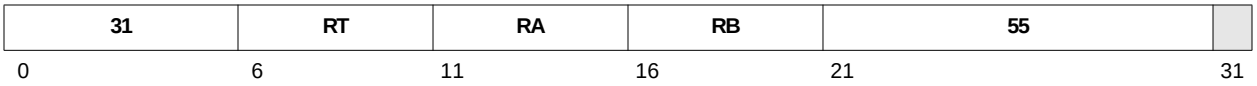
Invalid Instruction Forms

- RA = RT
- RA = 0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 word 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

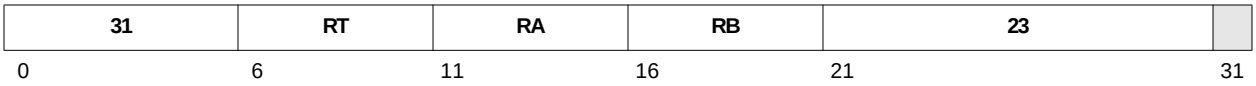
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

lwzx

Load Word 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 word at the EA is placed into register RT.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- RT

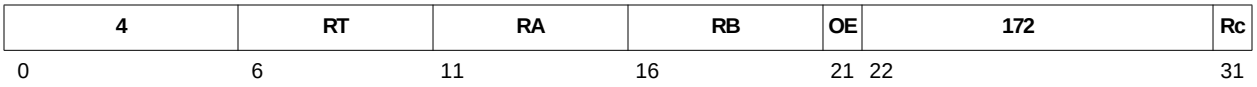
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

macchw	RT, RA, RB	OE=0, Rc=0
macchw.	RT, RA, RB	OE=0, Rc=1
macchwo	RT, RA, RB	OE=1, Rc=0
macchwo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ signed

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the high-order halfword of RB. The signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

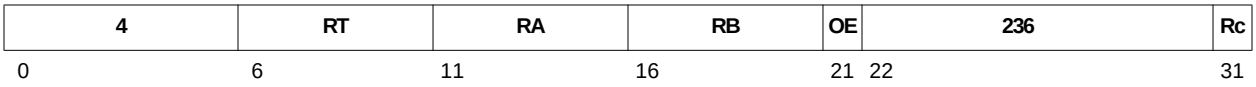
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

macchws

Multiply Accumulate Cross Halfword to Word Saturate Signed

macchws	RT, RA, RB	OE=0, Rc=0
macchws.	RT, RA, RB	OE=0, Rc=1
macchwso	RT, RA, RB	OE=1, Rc=0
macchwso.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ signed

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \,||\, ^{31}(\neg RT_0))$

else $(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the high-order halfword of RB. The signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is less than -2^{31}, the value stored in RT is -2^{31}. Likewise, if a result is greater than $2^{31} - 1$, the value stored in RT is $2^{31} - 1$.

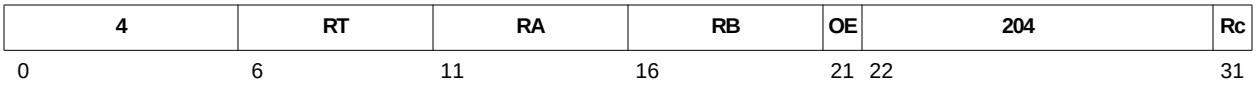
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

macchwsu	RT, RA, RB	OE=0, Rc=0
macchwsu.	RT, RA, RB	OE=0, Rc=1
macchwsuo	RT, RA, RB	OE=1, Rc=0
macchwsuo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ unsigned

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_{0})$

The low-order halfword of RA is multiplied by the high-order halfword of RB. The unsigned product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is greater than $2^{32} – 1$, the value stored in RT is $2^{32} – 1$.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

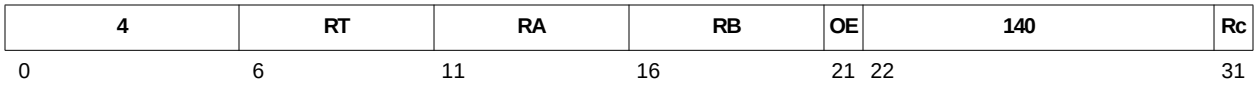
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

macchwu

Multiply Accumulate Cross Halfword to Word Modulo Unsigned

macchwu	RT, RA, RB	OE=0, Rc=0
macchwu.	RT, RA, RB	OE=0, Rc=1
macchwuo	RT, RA, RB	OE=1, Rc=0
macchwuo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ unsigned

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the high-order halfword of RB. The unsigned product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

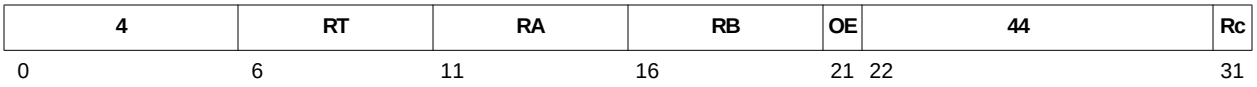
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

machhw	RT, RA, RB	OE=0, Rc=0
machhw.	RT, RA, RB	OE=0, Rc=1
machhwo	RT, RA, RB	OE=1, Rc=0
machhwo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{0:15}$ x $(RB)_{0:15}$ signed

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The high-order halfword of RA is multiplied by the high-order halfword of RB. The signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

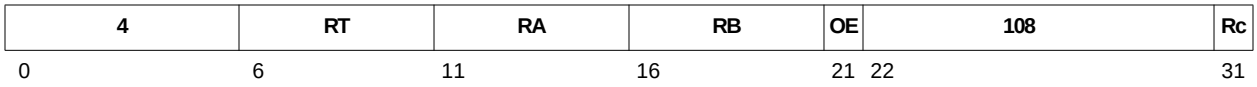
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

machhws

Multiply Accumulate High Halfword to Word Saturate Signed

machhws	RT, RA, RB	OE=0, Rc=0
machhws.	RT, RA, RB	OE=0, Rc=1
machhwso	RT, RA, RB	OE=1, Rc=0
machhwso.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{0:15}$ x $(RB)_{0:15}$ signed

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \,||\, ^{31}(\neg RT_0))$

else $(RT) \leftarrow temp_{1:32}$

The high-order halfword of RA is multiplied by the high-order halfword of RB. The signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is less than -2^{31}, the value stored in RT is -2^{31}. Likewise, if a result is greater than $2^{31} - 1$, the value stored in RT is $2^{31} - 1$.

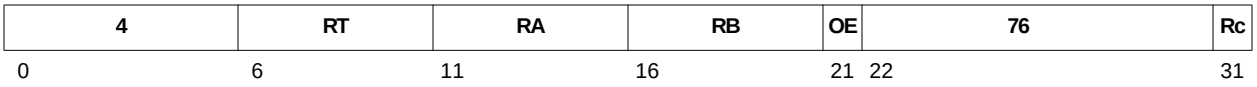
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

machhwsu	RT, RA, RB	OE=0, Rc=0
machhwsu.	RT, RA, RB	OE=0, Rc=1
machhwsuo	RT, RA, RB	OE=1, Rc=0
machhwsuo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{0:15}$ x $(RB)_{0:15}$ unsigned

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_{0})$

The high-order halfword of RA is multiplied by the high-order halfword of RB. The unsigned product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is greater than $2^{32} – 1$, the value stored in RT is $2^{32} – 1$.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

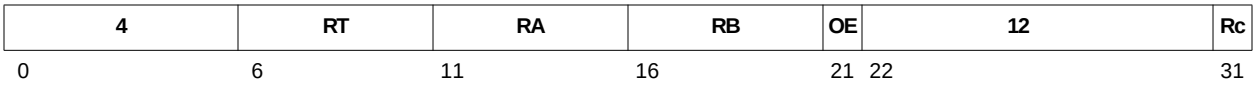
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

machhwu

Multiply Accumulate High Halfword to Word Modulo Unsigned

machhwu	RT, RA, RB	OE=0, Rc=0
machhwu.	RT, RA, RB	OE=0, Rc=1
machhwuo	RT, RA, RB	OE=1, Rc=0
machhwuo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{0:15}$ x $(RB)_{0:15}$ unsigned

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The high-order halfword of RA is multiplied by the high-order halfword of RB. The unsigned product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

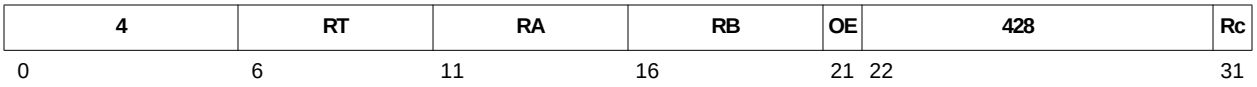
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

maclhw	RT, RA, RB	OE=0, Rc=0
maclhw.	RT, RA, RB	OE=0, Rc=1
maclhwo	RT, RA, RB	OE=1, Rc=0
maclhwo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the low-order halfword of RB. The signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

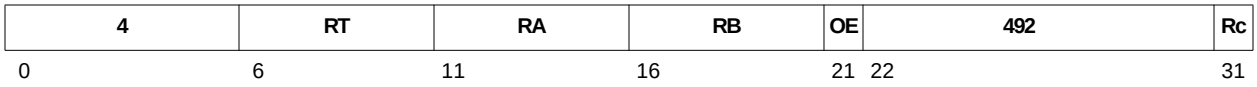
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

maclhws

Multiply Accumulate Low Halfword to Word Saturate Signed

maclhws	RT, RA, RB	OE=0, Rc=0
maclhws.	RT, RA, RB	OE=0, Rc=1
maclhwso	RT, RA, RB	OE=1, Rc=0
maclhwso.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \,||\, ^{31}(\neg RT_0))$

else $(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the low-order halfword of RB. The signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is less than -2^{31}, the value stored in RT is -2^{31}. Likewise, if a result is greater than $2^{31} - 1$, the value stored in RT is $2^{31} - 1$.

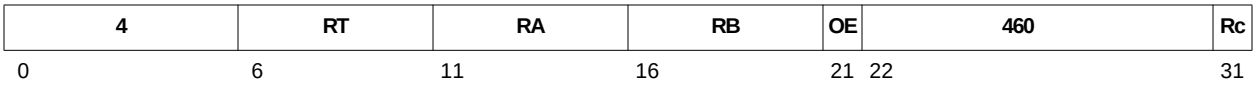
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

maclhwsu	RT, RA, RB	OE=0, Rc=0
maclhwsu.	RT, RA, RB	OE=0, Rc=1
maclhwsuo	RT, RA, RB	OE=1, Rc=0
maclhwsuo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ unsigned

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_{0})$

The low-order halfword of RA is multiplied by the low-order halfword of RB. The unsigned product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is greater than $2^{32} - 1$, the value stored in RT is $2^{32} - 1$.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

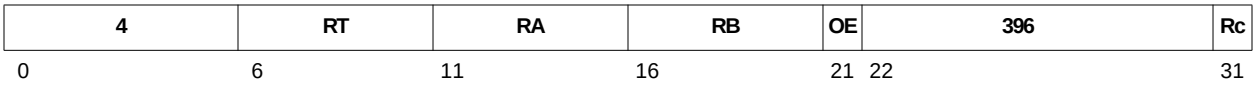
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

maclhwu

Multiply Accumulate Low Halfword to Word Modulo Unsigned

maclhwu	RT, RA, RB	OE=0, Rc=0
maclhwu.	RT, RA, RB	OE=0, Rc=1
maclhwuo	RT, RA, RB	OE=1, Rc=0
maclhwuo.	RT, RA, RB	OE=1, Rc=1



$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ unsigned

$temp_{0:32} \leftarrow prod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the low-order halfword of RB. The unsigned product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

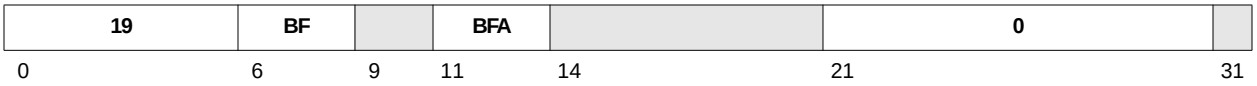
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

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[CR*n*] where *n* is specified by the BF field.

Invalid Instruction Forms

- Reserved fields

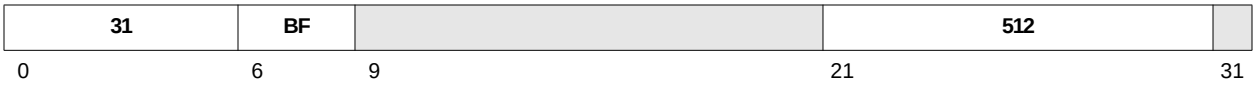
Architecture Note

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 Table 20-18 for clarification.

Table 20-18. Transfer Bit Mnemonic Assignment

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[CR*n*] where *n* is specified by the BF field.
- XER[SO, OV, CA]

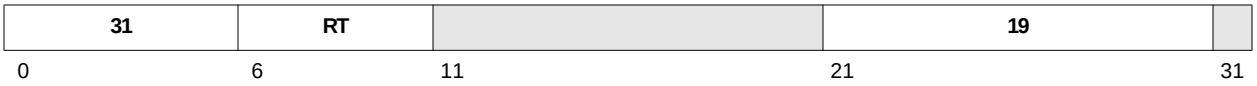
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mfcr 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

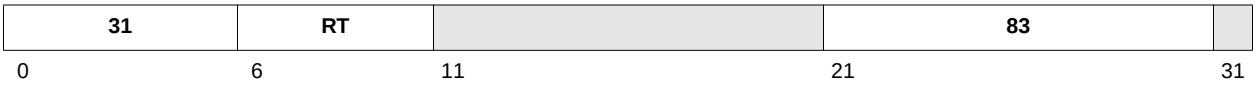
Execution of this instruction is privileged.

The DCR number (DCRN) specified in the assembler language coding of **mfdcr** refers to a DCR number. The assembler handles the unusual register number encoding to generate the DCRF field.

Architecture Note

This instruction is implementation-specific and may not be portable to other implementations.

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 IBM PowerPC Embedded Operating Environment.

mfspr

Move From Special Purpose Register

mfspr RT, SPRN

SPRN ← $SPRF_{5:9}$ || $SPRF_{0:4}$
(RT) ← (SPR(SPRN))

The contents of the SPR specified by the SPRF field are placed into register RT. See “Special Purpose Registers” on page 21-1 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 “Privileged Mode Operation” on page 3-37.

The SPR number (SPRN) specified in the assembler language coding of **mfspr** refers to an SPR number (see “Special Purpose Registers” on page 21-1 for a list of SPRN values). The assembler handles the unusual register number encoding to generate the SPRF field. Also, see “Privileged SPRs” on page 3-38 for information about privileged SPRs.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-19. Extended Mnemonics for mfspr

Mnemonic	Operands	Function	Other Registers Changed
mfccr0 **mfctr** **mfdac1** **mfdac2** **mfdear** **mfdbcr0** **mfdbcr1** **mfdbsr** **mfdccr** **mfdcwr** **mfdvc1** **mfdvc2** **mfesr** **mfevpr** **mfiac1** **mfiac2** **mfiac3** **mfiac4** **mficcr** **mficdbdr** **mflr** **mfpid** **mfpit** **mfpvr** **mfsgr** **mfsler** **mfsprg0** **mfsprg1** **mfsprg2** **mfsprg3** **mfsprg4** **mfsprg5** **mfsprg6** **mfsprg7** **mfsrr0** **mfsrr1** **mfsrr2** **mfsrr3** **mfsu0r** **mftcr** **mftsr** **mfxer** **mfzpr**	RT	Move from special purpose register SPRN. *Extended mnemonic for* **mfspr RT,SPRN** See “Special Purpose Registers” on page 21-1 for a list of valid SPRN values.	

mftb

Move From Time Base

mftb RT, TBRN

TBRN $\leftarrow$ $TBRF_{5:9}$ || $TBRF_{0:4}$
(RT) $\leftarrow$ (TBR(TBRN))

The contents of the time base register (TBR) specified by the TBRF field are placed into register RT. The following table lists the TBRN and TBRF values.

Table 20-20. Extended Mnemonics for mftb

Register Mnemonic	Register Name	TBRN		TBRF	Access
		Decimal	Hex		
TBL	Time Base Lower	268	0x10C	0x188	Read-only
TBU	Time Base Upper	269	0x10D	0x1A8	Read-only

If TBRN is a value other than those listed in the table, the results are boundedly undefined.

Registers Altered

- RT

Invalid Instruction Forms

- Reserved fields
- Invalid TBRF values

Programming Notes

The mnemonic **mftb** serves as both a hardware mnemonic and an extended mnemonic. The assembler recognizes an **mftb** mnemonic having two operands as the hardware form; an **mftb** mnemonic having one operand is recognized as the extended form.

The TBR number (TBRN) specified in the assembler language coding of the **mftb** instruction refers to a TBR number listed in the preceding table. The assembler handles the unusual register number encoding to generate the TBRF field.

Architecture Note

This instruction is part of the IBM PowerPC Embedded Virtual Environment.

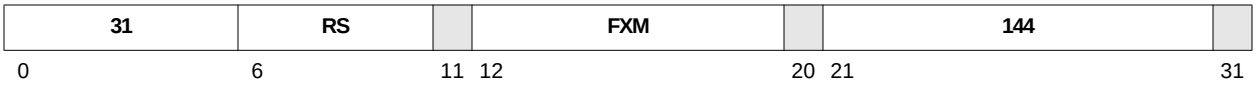
Table 20-21. Extended Mnemonics for mftb

Mnemonic	Operands	Function	Other Registers Altered
mftb	RT	Move the contents of TBL into RT. *Extended mnemonic for* **mftb RT,TBL**	
mftbu	RT	Move the contents of TBU into RT. *Extended mnemonic for* **mftb RT,TBU**	

mtcrf

Move to Condition Register Fields

mtcrf FXM, RS



mask ← $^4(FXM_0)$ || $^4(FXM_1)$ || ... || $^4(FXM_6)$ || $^4(FXM_7)$
(CR) ← ((RS) ∧ mask) ∨ ((CR) ∧ ¬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 20-22. Extended Mnemonics for mtcrf

Mnemonic	Operands	Function	Other Registers Altered
mtcr	RS	Move to CR. *Extended mnemonic for* **mtcrf 0xFF,RS**	

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 **mtdcr** refers to a DCR number. The assembler handles the unusual register number encoding to generate the DCRF field.

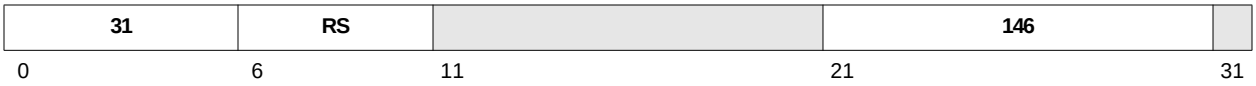
Architecture Note

This instruction is implementation-specific and may not be portable to other implementations.

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 IBM PowerPC Embedded Operating Environment.

mtspr SPRN, RS

SPRN ← $SPRF_{5:9}$ || $SPRF_{0:4}$
(SPR(SPRN)) ← (RS)

The contents of register RS are placed into register RT. See “Special Purpose Registers” on page 21-1 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 a 1. See “Privileged SPRs” on page 3-38 for more information.

The SPR number (SPRN) specified in the assembler language coding of the **mtspr** instruction refers to an SPR number (see “Special Purpose Registers” on page 21-1 for a list of SPRN values). The assembler handles the unusual register number encoding to generate the SPRF field.

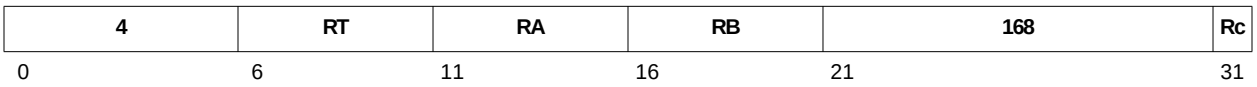
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-23. Extended Mnemonics for mtspr

Mnemonic	Operands	Function	Other Registers Altered
mtccr0 **mtctr** **mtdac1** **mtdac2** **mtdbcr0** **mtdbcr1** **mtdbsr** **mtdccr** **mtdcwr** **mtdear** **mtdvc1** **mtdvc2** **mtesr** **mtevpr** **mtiac1** **mtiac2** **mtiac3** **mtiac4** **mticcr** **mticdbdr** **mtlr** **mtpid** **mtpit** **mtpvr** **mtsgr** **mtsler** **mtsprg0** **mtsprg1** **mtsprg2** **mtsprg3** **mtsprg4** **mtsprg5** **mtsprg6** **mtsprg7** **mtsrr0** **mtsrr1** **mtsrr2** **mtsrr3** **mtsu0r** **mttbl** **mttbu** **mttcr** **mttsr** **mtxer** **mtzpr**	RS	Move to special purpose register SPRN. *Extended mnemonic for* **mtspr SPRN,RS** See “Special Purpose Registers” on page 21-1 for a list of valid SPRN values.	

mulchw	RT, RA, RB	Rc=0
mulchw.	RT, RA, RB	Rc=1



$(RT)_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ signed

The low-order halfword of RA is multiplied by the high-order halfword of RB. The resulting signed product replaces the contents of RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

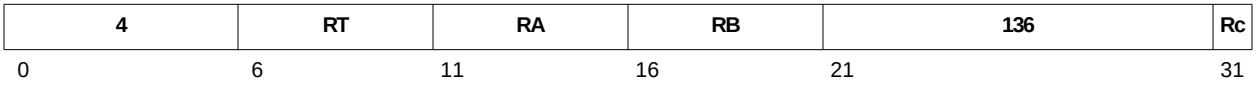
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

mulchwu

Multiply Cross Halfword to Word Unsigned

mulchwu	RT, RA, RB	Rc=0
mulchwu.	RT, RA, RB	Rc=1



$(RT)_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ unsigned

The low-order halfword of RA is multiplied by the high-order halfword of RB. The resulting unsigned product replaces the contents of RT.

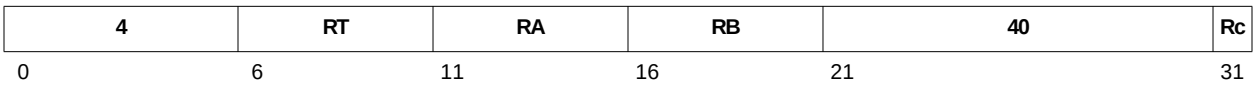
Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

mulhhw RT, RA, RB Rc=0
mulhhw. RT, RA, RB Rc=1



$(RT)_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ signed

The high-order halfword of RA is multiplied by the high-order halfword of RB. The resulting signed product replaces the contents of RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

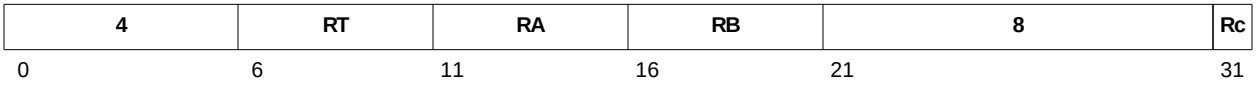
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

mulhhwu

Multiply High Halfword to Word Unsigned

mulhhwu	RT, RA, RB	Rc=0
mulhhwu.	RT, RA, RB	Rc=1



$(RT)_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ unsigned

The high-order halfword of RA is multiplied by the high-order halfword of RB. The resulting unsigned product replaces the contents of RT.

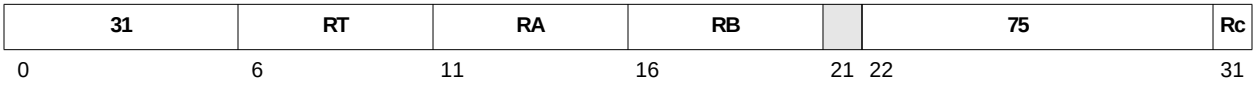
Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

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. **mulhw** generates the correct result when these operands are interpreted as signed quantities. **mulhwu** generates the correct result when these operands are interpreted as unsigned quantities.

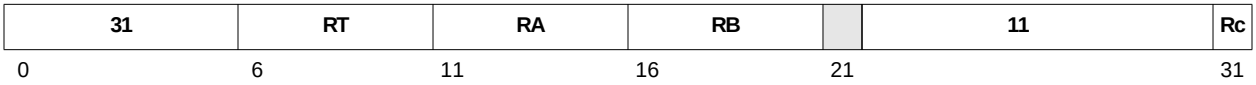
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mulhwu

Multiply High Word 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

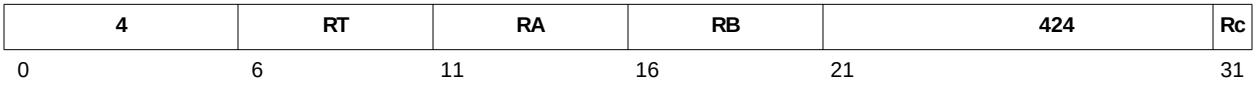
Programming Note

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.

mullhw	RT, RA, RB	Rc=0
mullhw.	RT, RA, RB	Rc=1



$(RT)_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed

The low-order halfword of RA is multiplied by the low-order halfword of RB. The resulting signed product replaces the contents of RT.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

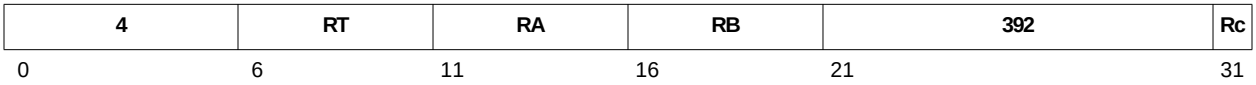
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

mullhwu

Multiply Low Halfword to Word Unsigned

mullhwu	RT, RA, RB	OE=0, Rc=0
mullhwu.	RT, RA, RB	OE=0, Rc=1



$(RT)_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ unsigned

The low-order halfword of RA is multiplied by the low-order halfword of RB. The resulting unsigned product replaces the contents of RT.

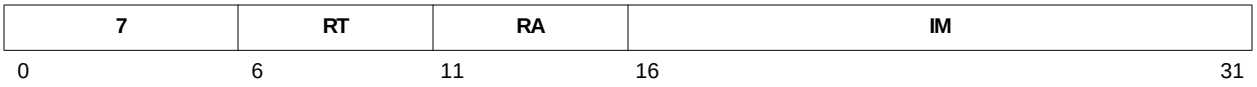
Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

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.

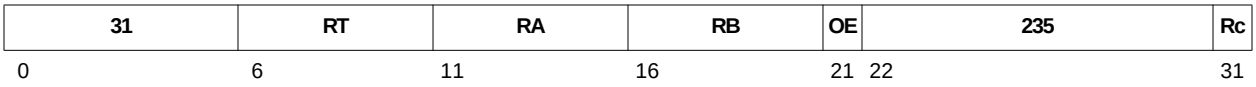
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

mullw

Multiply Low Word

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

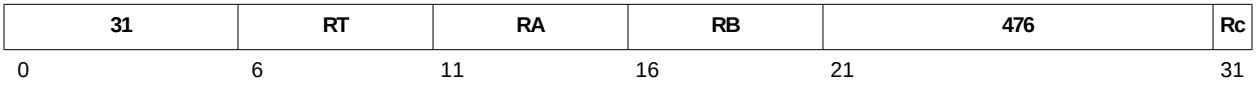
Programming Note

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	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

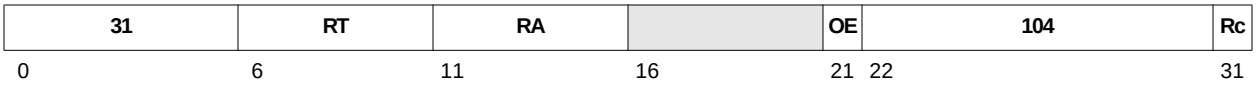
Architecture Note

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[SO, OV] if OE=1

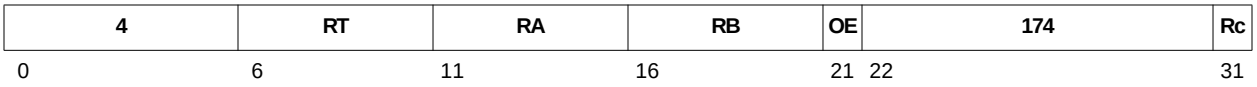
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

nmacchw	RT, RA, RB	OE=0, Rc=0
nmacchw.	RT, RA, RB	OE=0, Rc=1
nmacchwo	RT, RA, RB	OE=1, Rc=0
nmacchwo.	RT, RA, RB	OE=1, Rc=1



$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{0:15})$ signed

$temp_{0:32} \leftarrow nprod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the high-order halfword of RB. The negated signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1

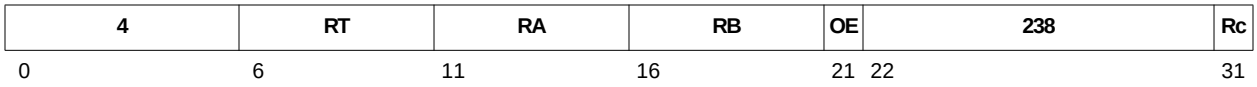
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

nmacchws

Negative Multiply Accumulate Cross Halfword to Word Saturate Signed

nmacchws	RT, RA, RB	OE=0, Rc=0
nmacchws.	RT, RA, RB	OE=0, Rc=1
nmacchwso	RT, RA, RB	OE=1, Rc=0
nmacchwso.	RT, RA, RB	OE=1, Rc=1



$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{0:15}$ signed

$temp_{0:32} \leftarrow nprod_{0:31} + (RT)$

if $((nprod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \,||\, {}^{31}(\neg RT_0))$

else $(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the high-order halfword of RB. The negated signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is less than -2^{31}, the value stored in RT is -2^{31}. Likewise, if a result is greater than $2^{31} - 1$, the value stored in RT is $2^{31} - 1$.

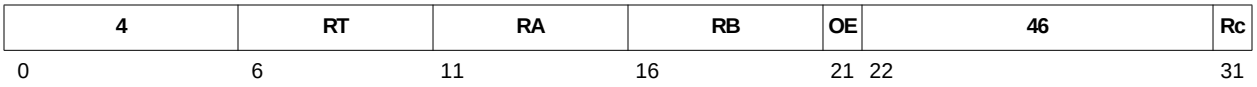
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

nmachhw	RT, RA, RB	OE=0, Rc=0
nmachhw.	RT, RA, RB	OE=0, Rc=1
nmachhwo	RT, RA, RB	OE=1, Rc=0
nmachhwo.	RT, RA, RB	OE=1, Rc=1



$nprod_{0:31} \leftarrow -((RA)_{0:15} \times (RB)_{0:15})$ signed

$temp_{0:32} \leftarrow nprod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The high-order halfword of RA is multiplied by the high-order halfword of RB. The negated signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

Registers Altered

- RT
- $CR[CR0]_{LT,\ GT,\ EQ,\ SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

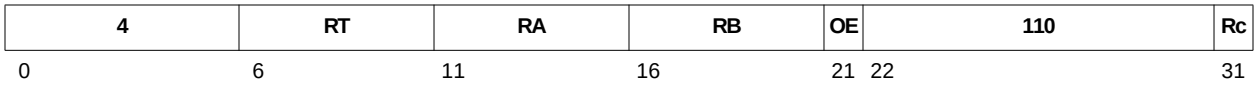
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

nmachhws

Negative Multiply Accumulate High Halfword to Word Saturate Signed

nmachhws	RT, RA, RB	OE=0, Rc=0
nmachhws.	RT, RA, RB	OE=0, Rc=1
nmachhwso	RT, RA, RB	OE=1, Rc=0
nmachhwso.	RT, RA, RB	OE=1, Rc=1



$nprod_{0:31} \leftarrow -((RA)_{0:15} \times (RB)_{0:15})$ signed

$temp_{0:32} \leftarrow nprod_{0:31} + (RT)$

if $((nprod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \,||\, ^{31}(\neg RT_0))$

else $(RT) \leftarrow temp_{1:32}$

The high-order halfword of RA is multiplied by the high-order halfword of RB. The negated signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow (i.e., it is accurately representable in 32 bits), the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is less than -2^{31}, the value stored in RT is -2^{31}. Likewise, if a result is greater than $2^{31} - 1$, the value stored in RT is $2^{31} - 1$.

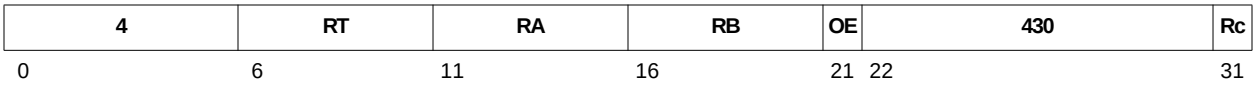
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

nmaclhw	RT, RA, RB	OE=0, Rc=0
nmaclhw.	RT, RA, RB	OE=0, Rc=1
nmaclhwo	RT, RA, RB	OE=1, Rc=0
nmachlwo.	RT, RA, RB	OE=1, Rc=1



$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{16:31})$ signed

$temp_{0:32} \leftarrow nprod_{0:31} + (RT)$

$(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the low-order halfword of RB. The negated signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register. The contents of RT are replaced by the low-order 32 bits of the temporary register.

Registers Altered

- RT
- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- XER[SO, OV] if OE contains 1

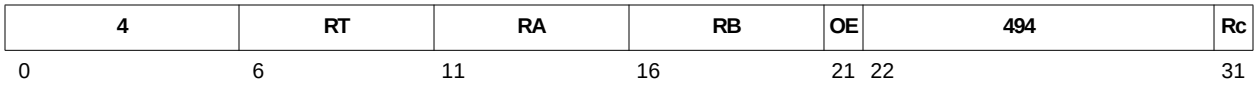
Architecture Note

This instruction is part of the Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

nmaclhws

Negative Multiply Accumulate High Halfword to Word Saturate Signed

nmaclhws	RT, RA, RB	OE=0, Rc=0
nmaclhws.	RT, RA, RB	OE=0, Rc=1
nmaclhwso	RT, RA, RB	OE=1, Rc=0
nmachlwso.	RT, RA, RB	OE=1, Rc=1



$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{16:31})$ signed

$temp_{0:32} \leftarrow nprod_{0:31} + (RT)$

if $((nprod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \,||\, ^{31}(\neg RT_0))$

else $(RT) \leftarrow temp_{1:32}$

The low-order halfword of RA is multiplied by the low-order halfword of RB. The negated signed product is summed with the contents of RT and the sum is stored in a 33-bit temporary register.

If a result does not overflow, the low-order 32 bits of the temporary register are stored in RT.

If a result overflows, the returned result is the nearest representable value. Thus, if a result is less than -2^{31}, the value stored in RT is -2^{31}. Likewise, if a result is greater than $2^{31} - 1$, the value stored in RT is $2^{31} - 1$.

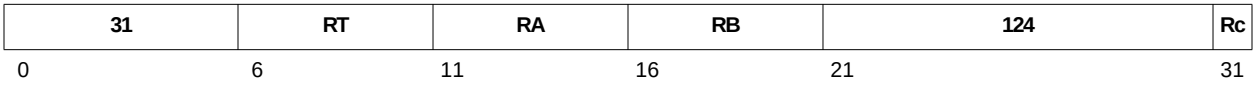
Registers Altered

- 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 Multiply-Accumulate instruction set extensions and complies with the architectural requirements for APUs of the IBM PowerPC Embedded Environment. As such, it is not part of the PowerPC Architecture, nor is it part of the IBM PowerPC Embedded Environment. Programs that use this instruction may not be portable to other implementations.

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.

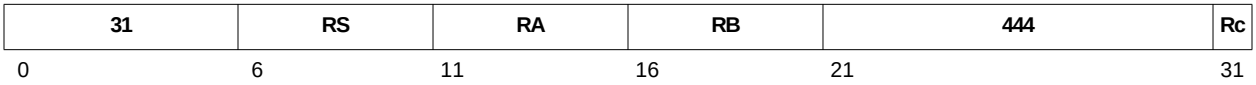
Table 20-24. Extended Mnemonics for nor, nor.

Mnemonic	Operands	Function	Other Registers Altered
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

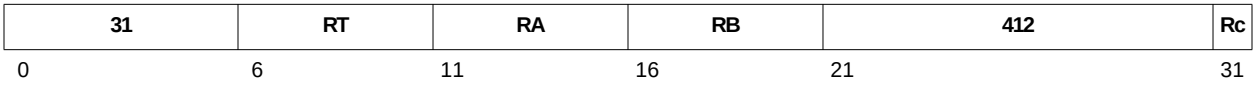
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-25. Extended Mnemonics for or, or.

Mnemonic	Operands	Function	Other Registers Altered
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	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

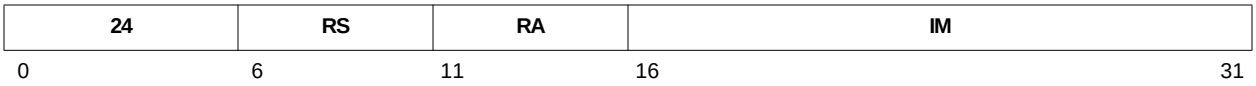
Architecture Note

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

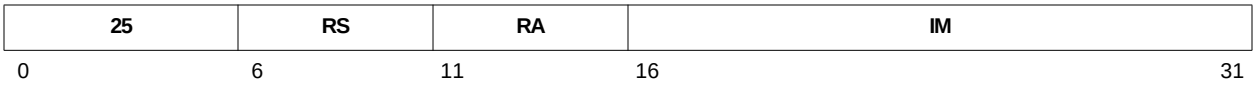
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-26. 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 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

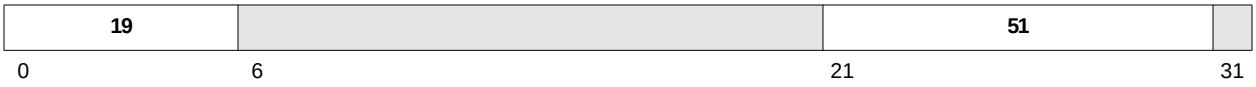
Architecture Note

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

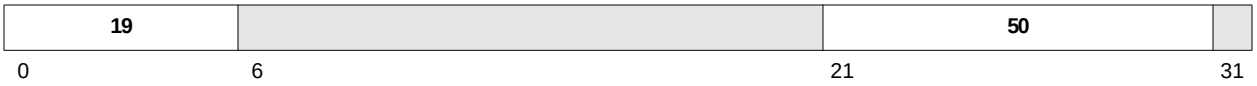
Programming Note

Execution of this instruction is privileged and context-synchronizing.

Architecture Note

This instruction part of the IBM PowerPC Embedded Operating Environment.

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.

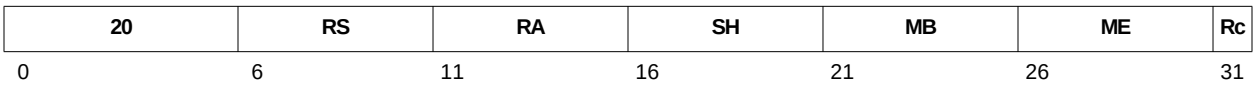
Architecture Note

This instruction is part of the IBM PowerPC Embedded Operating Environment.

rlwimi

Rotate Left Word 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

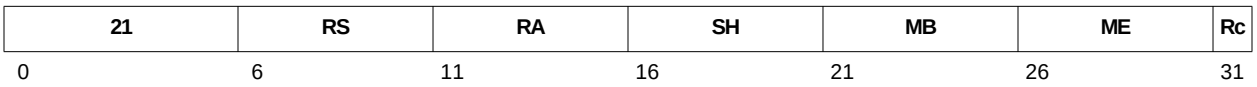
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-27. Extended Mnemonics for rlwimi, rlwimi.

Mnemonic	Operands	Function	Other Registers Altered
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	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 20-28. Extended Mnemonics for rlwinm, rlwinm.

Mnemonic	Operands	Function	Other Registers Altered
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. ($n \leq b < 32$) $(RA)_{b-n:31-n} \leftarrow (RS)_{b:31}$ $(RA)_{32-n:31} \leftarrow {}^{n}0$ $(RA)_{0:b-n-1} \leftarrow {}^{b-n}0$ *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]

Table 20-28. Extended Mnemonics for rlwinm, rlwinm. (continued)

Mnemonic	Operands	Function	Other Registers Altered
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]
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) ← 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) ← 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]

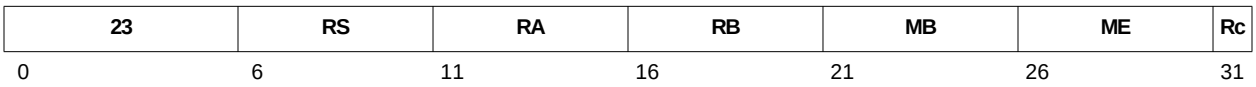
Table 20-28. Extended Mnemonics for rlwinm, rlwinm. (continued)

Mnemonic	Operands	Function	Other Registers Altered
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 Word then AND with Mask

rlwnm	RA, RS, RB, MB, ME	Rc=0
rlwnm.	RA, RS, RB, MB, ME	Rc=1



$r \leftarrow ROTL((RS), (RB)_{27:31})$
$m \leftarrow MASK(MB, ME)$
$(RA) \leftarrow r \wedge m$

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

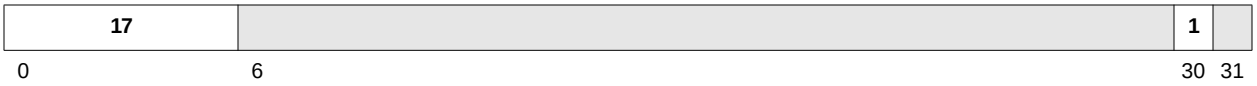
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-29. Extended Mnemonics for rlwnm, rlwnm.

Mnemonic	Operands	Function	Other Registers Altered
rotlw	RA, RS, RB	Rotate left. $(RA) \leftarrow ROTL((RS), (RB)_{27:31})$ *Extended mnemonic for* **rlwnm RA,RS,RB,0,31**	
rotlw.		*Extended mnemonic for* **rlwnm. RA,RS,RB,0,31**	CR[CR0]

sc



(SRR1) ← (MSR)
(SRR0) ← (PC)
PC ← $EVPR_{0:15}$ || 0x0C00
(MSR[WE, EE, PR, DR, IR]) ← 0

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 halfword of the Exception Vector Prefix Register (EVPR) to the left of 0x0C00.

The MSR[WE, EE, PR, DR, IR] bits are set to 0.

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]

Invalid Instruction Forms

- Reserved fields

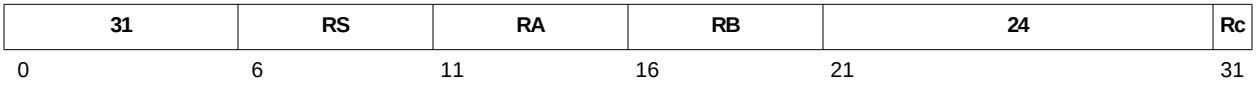
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

slw

Shift Left Word

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 RB_{26} = 1, register RA is set to zero.

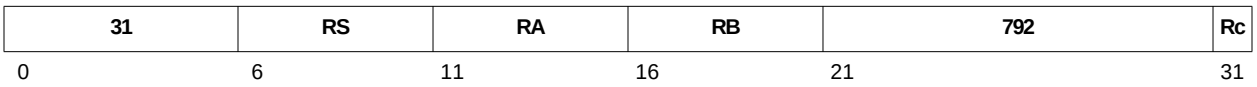
Registers Altered

- 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	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

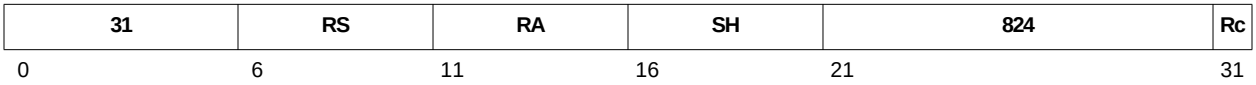
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

srawi

Shift Right Algebraic Word Immediate

srawi	RA, RS, SH	Rc=0
srawi.	RA, RS, SH	Rc=1



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)

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.

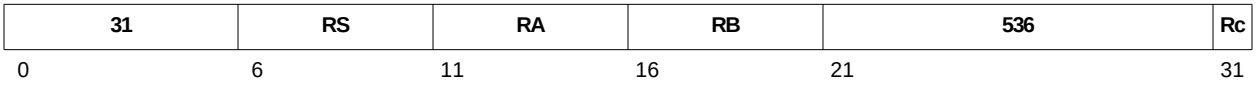
Registers Altered

- 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	RA, RS, RB	Rc=0
srw.	RA, RS, RB	Rc=1




```
n ← (RB)27:31
r ← ROTL((RS), 32 – n)
if (RB)26 = 0 then
   m ← MASK(n, 31)
else
   m ← 320
(RA) ← r ∧ 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

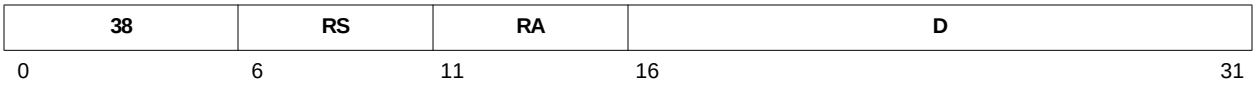
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stb

Store Byte

stb RS, D(RA)



EA $\leftarrow$ (RA|0) + EXTS(D)
MS(EA, 1) $\leftarrow$ $(RS)_{24:31}$

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.

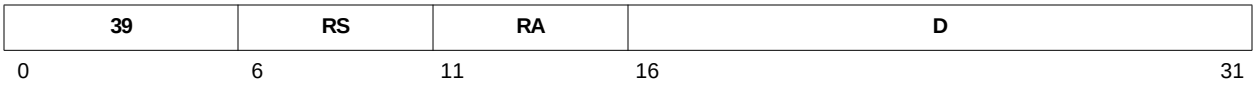
Registers Altered

- None

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbu 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

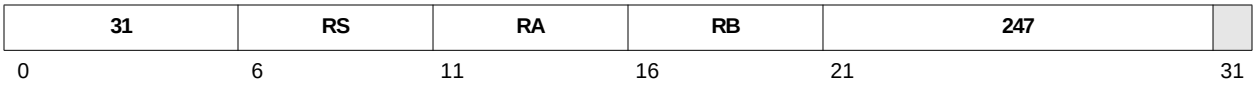
Architecture Note

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

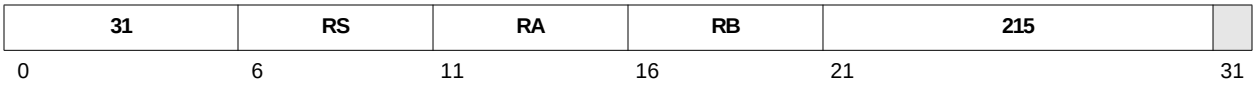
Invalid Instruction Forms

- Reserved fields
- RA = 0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stbx RS, RA, RB



EA ← (RA|0) + (RB)
MS(EA, 1) ← $(RS)_{24:31}$

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

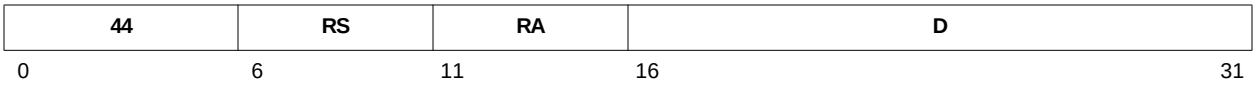
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sth

Store Halfword

sth RS, D(RA)



EA $\leftarrow$ (RA|0) + EXTS(D)
MS(EA, 2) $\leftarrow$ $(RS)_{16:31}$

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 halfword of register RS is stored into the halfword at the EA in main storage.

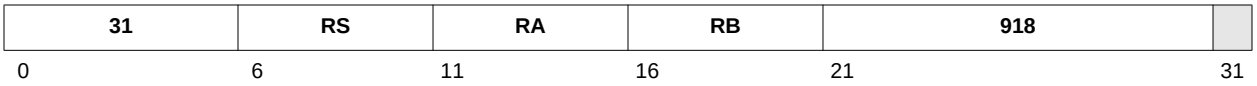
Registers Altered

- None

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthbrx RS, RA, RB



EA ← (RA|0) + (RB)
MS(EA, 2) ← $(RS)_{24:31}$ || $(RS)_{16:23}$

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 halfword of register RS is byte-reversed. The result is stored into the halfword at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

Invalid Instruction Forms

- Reserved fields

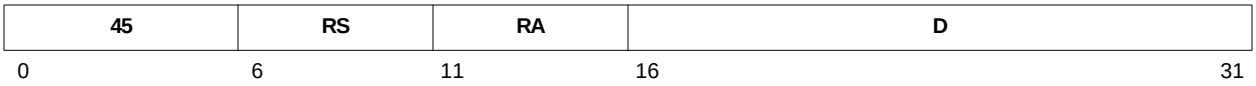
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthu

Store Halfword with Update

sthu RS, D(RA)



EA $\leftarrow$ (RA|0) + EXTS(D)
MS(EA, 2) $\leftarrow$ $(RS)_{16:31}$
(RA) $\leftarrow$ 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 halfword of register RS is stored into the halfword at the EA.

The EA is placed into register RA.

Registers Altered

- RA

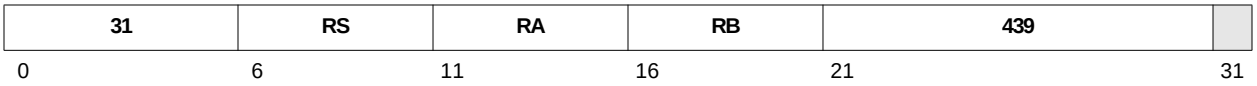
Invalid Instruction Forms

- RA = 0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthux RS, RA, RB



EA $\leftarrow$ (RA|0) + (RB)
MS(EA, 2) $\leftarrow$ $(RS)_{16:31}$
(RA) $\leftarrow$ 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 halfword of register RS is stored into the halfword 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

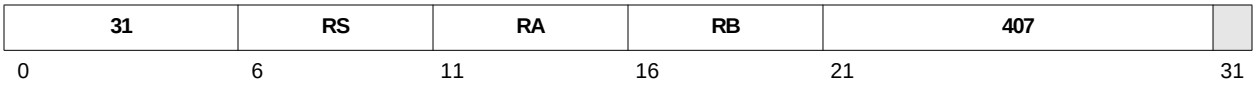
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sthx

Store Halfword 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 halfword of register RS is stored into the halfword at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

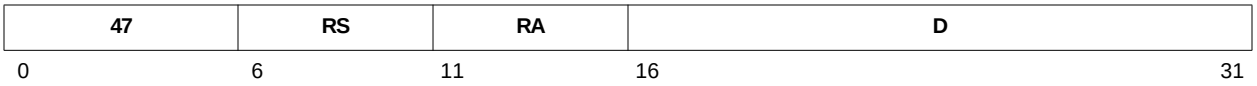
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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

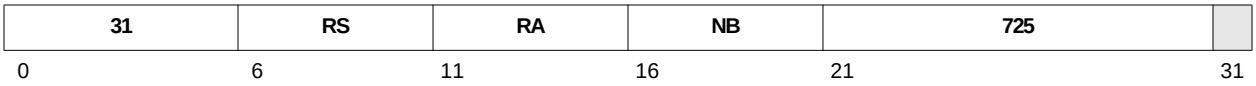
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stswi

Store String Word 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.

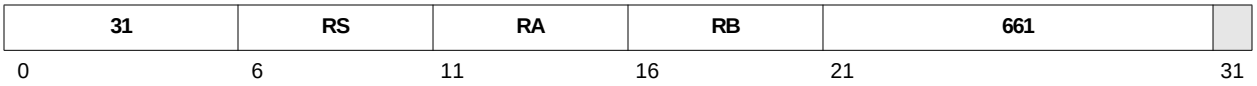
Registers Altered

- None

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 will not occur. Data storage exceptions and alignment exceptions are examples of precise data exceptions.

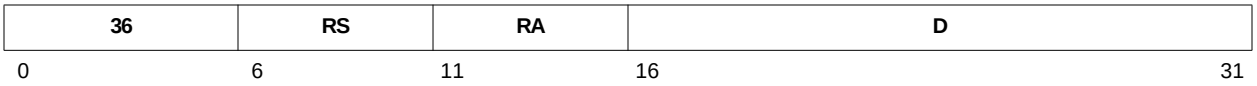
However, the architecture makes no statement regarding imprecise exceptions related to **stswx** when XER[TBC] = 0. IBM PowerPC Embedded controllers generate an imprecise exception (machine check) on this instruction when all of the following conditions are true:

- The instruction passes all protection bounds checking
- The address is cachable
- 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)

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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

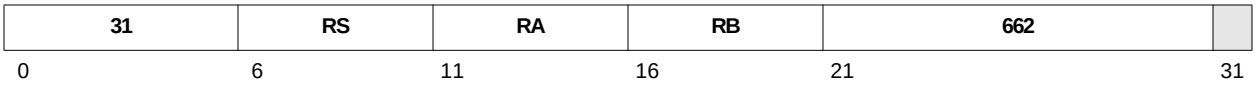
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwbrx

Store Word 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 word at the EA.

If instruction bit 31 contains 1, the contents of CR[CR0] are undefined.

Registers Altered

- None

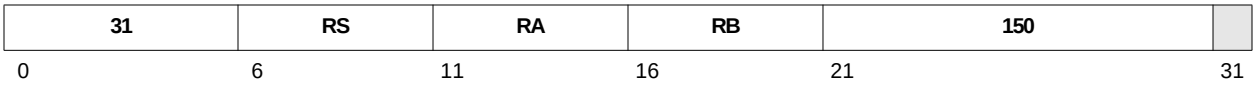
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwcx. RS, RA, RB




```
EA ← (RA|0) + (RB)
if RESERVE = 1 then
   MS(EA, 4) ← (RS)
   RESERVE ← 0
   (CR[CR0]) ← 20 || 1 || XERSO
else
   (CR[CR0]) ← 20 || 0 || XERSO
```

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 word 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 word-aligned.

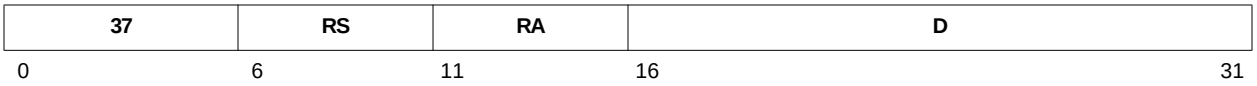
stwcx.

Store Word Conditional Indexed

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 word at the EA.

The EA is placed into register RA.

Registers Altered

- RA

Invalid Instruction Forms

- RA = 0

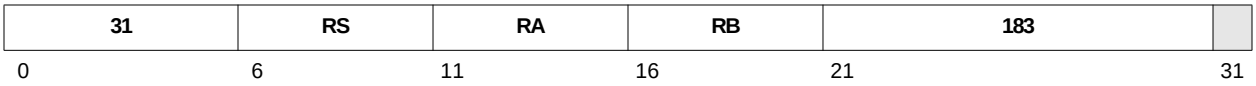
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

stwux

Store Word 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 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

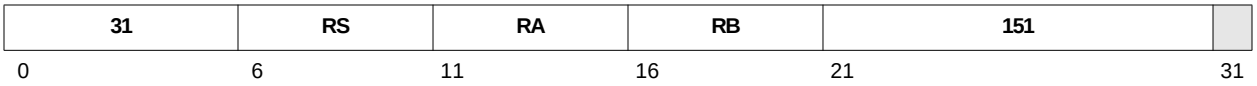
Invalid Instruction Forms

- Reserved fields
- RA = 0

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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 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

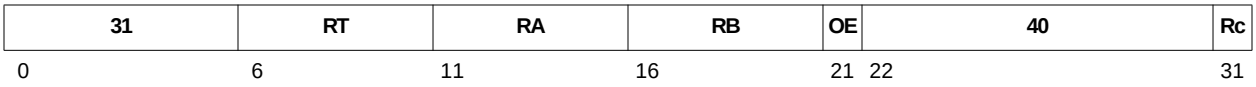
Architecture Note

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

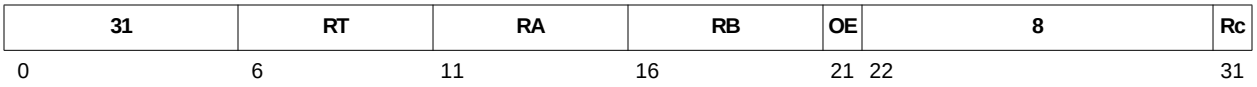
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

Table 20-30. Extended Mnemonics for subf, subf., subfo, subfo.

Mnemonic	Operands	Function	Other Registers Altered
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	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) $\leftarrow$ $\neg$(RA) + (RB) + 1
if $\neg$(RA) + (RB) + 1 $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] $\leftarrow$ 1
else
  XER[CA] $\leftarrow$ 0

The sum of the ones complement of register RA, register RB, and 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.

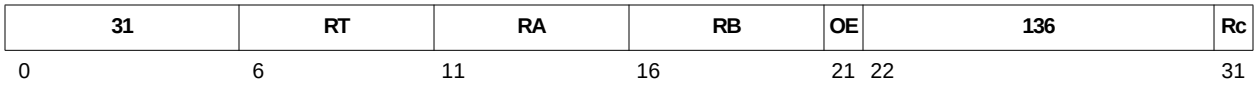
Table 20-31. Extended Mnemonics for subfc, subfc., subfco, subfco.

Mnemonic	Operands	Function	Other Registers Altered
subc	RT, RA, RB	Subtract (RB) from (RA). (RT) $\leftarrow$ $\neg$(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] $\overset{u}{>}$ $2^{32} - 1$ 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.

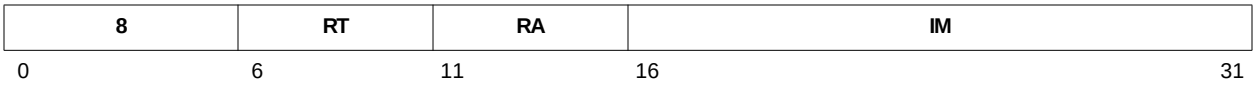
Registers Altered

- 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 RT, RA, IM



(RT) ← ¬(RA) + EXTS(IM) + 1
if ¬(RA) + EXTS(IM) + 1 $\overset{u}{>}$ $2^{32} - 1$ then
  XER[CA] ← 1
else
  XER[CA] ← 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]

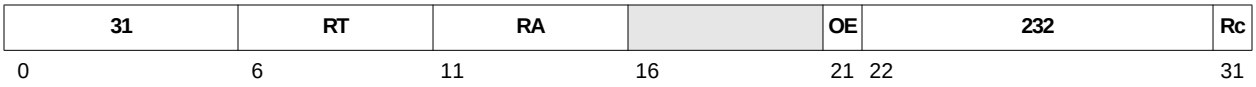
Architecture Note

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] $\geq^{u}$ 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]

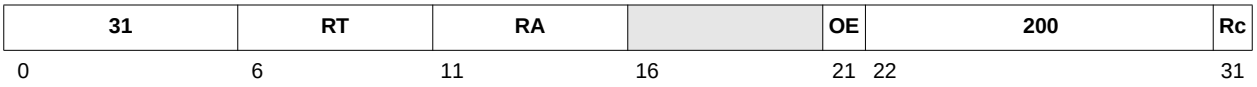
Invalid Instruction Forms

- Reserved fields

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

subfze	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

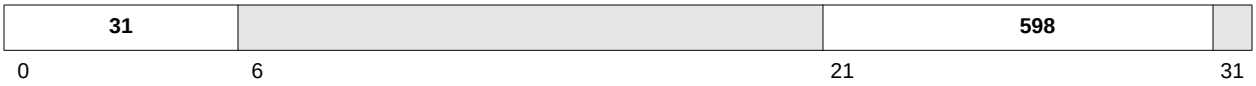
Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

sync

Synchronize

sync



The **sync** instruction guarantees that all instructions initiated by the processor preceding **sync** will complete before **sync** completes, and that no subsequent instructions will be initiated by the processor until after **sync** completes. When **sync** completes, all storage accesses that were initiated by the processor before the **sync** instruction will have been 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

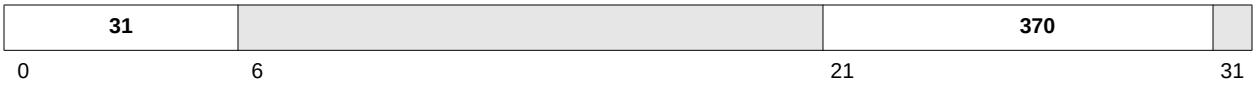
Architecturally, the **eieio** instruction orders storage access, not instruction completion. Therefore, non-storage operations that follow **eieio** could complete before storage operations that precede **eieio**. The **sync** instruction guarantees ordering of instruction completion and storage access. For the PPC405CR, 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 that corresponds to the desired behavior.

Architecture Note

This instruction is part of the PowerPC User Instruction Set Architecture.

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.

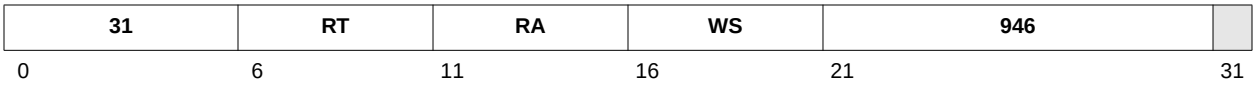
Architecture Note

This instruction is part of the IBM PowerPC Embedded Operating Environment.

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] ← U0
  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

This instruction part of the IBM PowerPC Embedded Operating Environment.

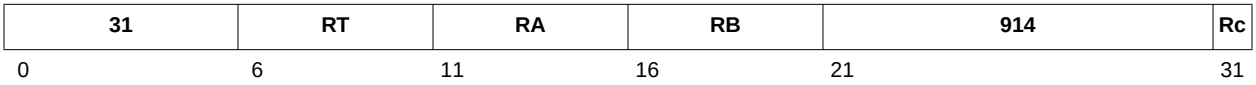
Table 20-32. Extended Mnemonics for tlbre

Mnemonic	Operands	Function	Other Registers Altered
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 XREF for details. The record bit (Rc) specifies whether the results of the search will 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.

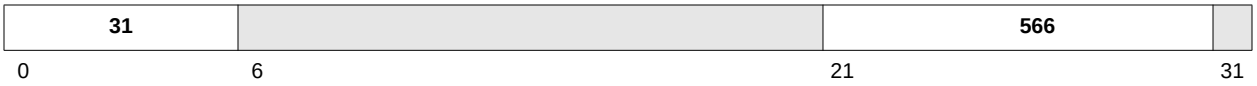
Programming Note

This instruction is privileged. Translation is not required to be active during the execution of this instruction.

Architecture Note

This instruction part of the IBM PowerPC Embedded Operating Environment.

tlbsync



The **tlbsync** instruction is provided in the PowerPC architecture to support synchronization of TLB operations among the processors of a multi-processor system. In the PPC405CR, this instruction performs no operation, 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.

Since the PPC405CR does not support tightly-coupled multiprocessor systems, **tlbsync** performs no operation.

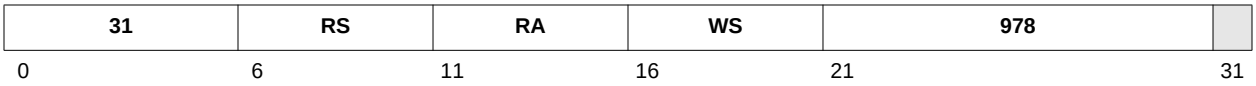
Architecture Note

This instruction is part of the IBM PowerPC Embedded Operating Environment.

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]
  U0← 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] ← RT[0:21]
EX,WR ← RS[22:23]
ZSEL[0:3] ← RS[24:27]
WIMG ← RS[28:31]

Architecture Note

This instruction part of the IBM PowerPC Embedded Operating Environment.

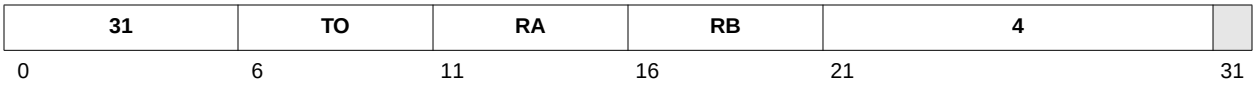
Table 20-33. Extended Mnemonics for tlbwe

Mnemonic	Operands	Function	Other Registers Altered
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 Word

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) $\overset{u}{<}$ (RB) $\wedge$ TO_3 = 1) $\vee$
((RA) $\overset{u}{>}$ (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 “Program Interrupt” on page 9-35.

  (SRR0) $\leftarrow$ address of **tw** instruction
  (SRR1) $\leftarrow$ (MSR)
  (ESR[PTR]) $\leftarrow$ 1
  (MSR[WE, EE, PR, DR, IR]) $\leftarrow$ 0
  PC $\leftarrow$ $EVPR_{0:15}$ || 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]) $\leftarrow$ 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]) $\leftarrow$ 1
  PC $\leftarrow$ 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 “Debug Interrupt” on page 9-39.

  (SRR2) $\leftarrow$ address of **tw** instruction
  (SRR3) $\leftarrow$ (MSR)
  (DBSR[TIE]) $\leftarrow$ 1
  (MSR[WE, EE, PR, CE, DE, DR, IR]) $\leftarrow$ 0
  PC $\leftarrow$ $EVPR_{0:15}$ || 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 “Program Interrupt” on page 9-35.

(SRR0) ← address of **tw** instruction
(SRR1) ← (MSR)
(ESR[PTR]) ← 1
(DBSR[TIE,IDE]) ← 1,1
(MSR[WE, EE, PR, DR, IR]) ← 0
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.

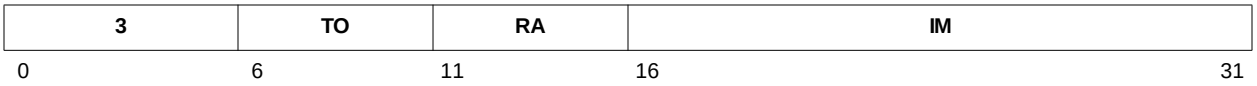
Table 20-34. Extended Mnemonics for tw

Mnemonic	Operands	Function	Other Registers Altered
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**	

Table 20-34. Extended Mnemonics for tw (continued)

Mnemonic	Operands	Function	Other Registers Altered
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 TO, RA, IM




```
if (   ((RA) <  EXTS(IM) ∧ TO0 = 1)  ∨
       ((RA) >  EXTS(IM) ∧ TO1 = 1)  ∨
       ((RA) =  EXTS(IM) ∧ TO2 = 1)  ∨
       ((RA) <= EXTS(IM) ∧ TO3 = 1)  ∨
       ((RA) >= EXTS(IM) ∧ TO4 = 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 “Program Interrupt” on page 9-35.

  (SRR0) ← address of **twi** instruction
  (SRR1) ← (MSR)
  (ESR[PTR]) ← 1
  (MSR[WE, EE, PR, DR, IR]) ← 0
  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 of the PPC405CR.

  (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 “Debug Interrupt” on page 9-39.

  (SRR2) ← address of **twi** instruction
  (SRR3) ← (MSR)
  (DBSR[TIE]) ← 1
  (MSR[WE, EE, PR, CE, DE, DR, IR]) ← 0
  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 will report the debug event as an *imprecise* event and will cause a Program interrupt. See “Program Interrupt” on page 9-35.

twi

Trap Word Immediate

(SRR0) ← address of **twi** instruction
(SRR1) ← (MSR)
(ESR[PTR]) ← 1
(DBSR[TIE,IDE]) ← 1,1
(MSR[WE, EE, PR, DR, IR]) ← 0
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 20-35. Extended Mnemonics for twi

Mnemonic	Operands	Function	Other Registers Altered
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**	

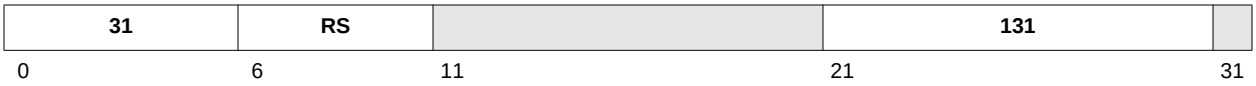
Table 20-35. Extended Mnemonics for twi (continued)

Mnemonic	Operands	Function	Other Registers Altered
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



$MSR[EE] \leftarrow (RS)_{16}$

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.

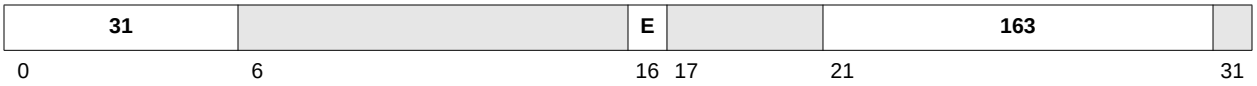
This instruction is used to provide atomic update of MSR[EE]. Typical usage is:

```
mfmsr Rn    #save EE in Rn[16]
wrteei 0    #Turn off EE
•           #Code with EE disabled
•
•
wrtee Rn    #restore EE without affecting any MSR changes that occurred in the disabled code
```

Architecture Note

This instruction part of the IBM PowerPC Embedded Operating Environment.

wrteei E



MSR[EE] ← E

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 an 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 any MSR changes that occurred in the disabled code
```

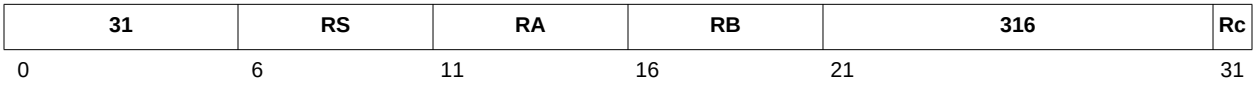
Architecture Note

This instruction part of the IBM PowerPC Embedded Operating 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.

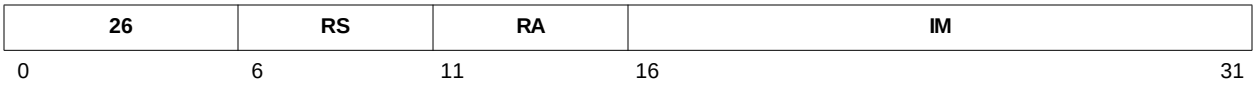
Registers Altered

- $CR[CR0]_{LT, GT, EQ, SO}$ if Rc contains 1
- RA

Architecture Note

This instruction part of the IBM PowerPC Embedded Operating Environment.

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

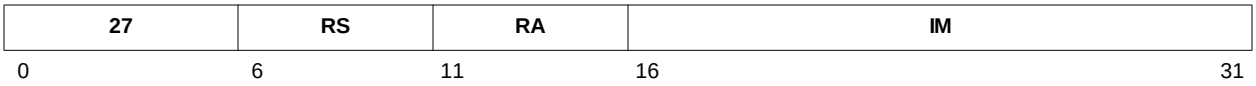
Architecture Note

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.

Chapter 21. Register Summary

The registers are grouped into categories, based on access mode: General Purpose Registers (GPRs), Special Purpose Registers (SPRs), Time Base Registers (TBRs), the Machine State Register (MSR), the Condition Register (CR), Device Control Registers (DCRs), and memory-mapped I/O (MMIO) registers.

21.1 Reserved Registers

Any register numbers not listed in the tables which follow are *reserved,* and should be neither read nor written. These reserved register numbers may be used for additional functions in future processors.

21.2 Reserved Fields

For all registers having fields marked as reserved, the reserved fields should be written as *zero* and read as *undefined*. That is, when writing to a reseved field, write a 0 to the field. When reading from a reserved field, ignore the field.

It is good coding practice 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.

21.3 General Purpose Registers

The PPC405CR processor core provides 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.

Table 21-1. PPC405CR General Purpose Registers

Mnemonic	Register Name	GPR Number		Access
		Decimal	Hex	
R0–R31	General Purpose Register 0–31	0–31	0x0–0x1F	Read/Write

21.4 Machine State Register and Condition Register

Because these registers are accessed using special instructions, they do not require addressing.

21.5 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.

Table 21-2 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 the five-bit subfields of the SPRF field, which 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.

All SPRs are privileged, except the Count Register (CTR), the Link Register (LR), SPR General Purpose Registers (SPRG4–SPRG7, read-only), User SPR General Purpose Register (USPRG0), and the Fixed-point Exception Register (XER). Note that access to the Time Base Lower (TBL) and Time Base Upper (TBU) registers, when addressed as SPRs, is write-only and privileged. However, when addressed as Time Base Registers (TBRs), read access to these registers is not privileged. See “Time Base Registers” on page 21-3 for more information.

Table 21-2 lists the SPRs, their mnemonics and names, their numbers (SPRN) and the corresponding SPRF numbers, and access. All SPR numbers not listed are reserved, and should be neither read nor written.

Table 21-2. Special Purpose Registers

Mnemonic	Register Name	SPRN		SPRF	Access
		Decimal	Hex		
CCR0	Core Configuration Register 0	947	0x3B3	0x27D	Read/Write
CTR	Count Register	9	0x009	0x120	Read/Write
DAC1	Data Address Compare 1	1014	0x3F6	0x2DF	Read/Write
DAC2	Data Address Compare 2	1015	0x3F7	0x2FF	Read/Write
DBCR0	Debug Control Register 0	1010	0x3F2	0x25F	Read/Write
DBCR1	Debug Control Register 1	957	0x3BD	0x3BD	Read/Write
DBSR	Debug Status Register	1008	0x3F0	0x21F	Read/Clear
DCCR	Data Cache Cachability Register	1018	0x3FA	0x35F	Read/Write
DCWR	Data Cache Write-through Register	954	0x3BA	0x35D	Read/Write
DVC1	Data Value Compare 1	950	0x3B6	0x2DD	Read/Write
DVC2	Data Value Compare 2	951	0x3B7	0x2FD	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
IAC1	Instruction Address Compare 1	1012	0x3F4	0x29F	Read/Write
IAC2	Instruction Address Compare 2	1013	0x3F5	0x2B5	Read/Write
IAC3	Instruction Address Compare 3	948	0x3B4	0x29D	Read/Write
IAC4	Instruction Address Compare 4	949	0x3B5	0x2BD	Read/Write
ICCR	Instruction Cache Cachability 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

Table 21-2. Special Purpose Registers (continued)

Mnemonic	Register Name	SPRN		SPRF	Access
		Decimal	Hex		
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
SLER	Storage Little Endian Register	955	0x3BB	0x37D	Read/Write
SPRG0	SPR General 0	272	0x110	0x208	Read/Write
SPRG1	SPR General 1	273	0x111	0x228	Read/Write
SPRG2	SPR General 2	274	0x112	0x248	Read/Write
SPRG3	SPR General 3	275	0x113	0x268	Read/Write
SPRG4	SPR General 4	260	0x104	0x088	Read-only
SPRG4	SPR General 4	276	0x114	0x288	Read/Write
SPRG5	SPR General 5	261	0x105	0x0A8	Read-only
SPRG5	SPR General 5	277	0x115	0x2A8	Read/Write
SPRG6	SPR General 6	262	0x106	0x0C8	Read-only
SPRG6	SPR General 6	278	0x116	0x2C8	Read/Write
SPRG7	SPR General 7	263	0x107	0x0E8	Read-only
SPRG7	SPR General 7	279	0x117	0x2E8	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
SU0R	Storage User-defined 0 Register	956	0x3BC	0x39D	Read/Write
TBL	Time Base Lower	284	0x11C	0x388	Write-only
TBU	Time Base Upper	285	0x11D	0x3A8	Write-only
TCR	Timer Control Register	986	0x3DA	0x35E	Read/Write
TSR	Timer Status Register	984	0x3D8	0x31E	Read/Clear
USPRG0	User SPR General 0	256	0x100	0x008	Read/Write
XER	Fixed Point Exception Register	1	0x001	0x020	Read/Write
ZPR	Zone Protection Register	944	0x3B0	0x21D	Privileged

21.6 Time Base Registers

The PowerPC Architecture provides a 64-bit time base. Chapter 10, “Timer Facilities,” describes the architected time base. In the PPC405CR, the time base is implemented as two 32-bit time base registers (TBRs). The low-order 32 bits of the time base are read from the TBL and the high-order 32 bits are read from the TBL.

User-mode access to the TBRs is read-only, and there is no explicitly privileged read access to the time base.

The **mftb** instruction reads from TBL and TBU. (Writing the time base is accomplished by moving the contents of a GPR to a pair of SPRs, which are also called TBL and TBU, using the **mtspr** instruction.)

Table 21-3 shows the mnemonics, names, and numbers of the TBRs. The columns under “TBRN” list the register numbers used as operands in assembler language coding of the **mftb** and **mtspr** instructions. The column labeled “TBRF” lists the corresponding fields contained in the *machine code* of **mftb** and **mtspr**. The TBRN field contains two five-bit subfields of the TBRF field; the subfields are *reversed* in the machine code for the **mftb** and **mtspr** instructions (TBRN $\leftarrow$ $TBRF_{5:9}$ || $TBRF_{0:4}$). Note that the assembler handles the special coding transparently.

Table 21-3. Time Base Registers

Mnemonic	Register Name	TBRN		TBRF	Access
		Decimal	Hex		
TBL	Time Base Lower (Read-only)	268	0x10C	0x188	Read-only
TBU	Time Base Upper (Read-only)	269	0x10D	0x1A8	Read-only

21.7 Device Control Registers

Device Control Registers (DCRs) are on-chip registers that are architecturally outside of the processor core. They are used to control, configure, and hold status for various functional units. DCRs are accessed using the **mfdcr** and **mtdcr** instructions.

The **mfdcr** and **mtdcr** instructions are privileged, for all DCR numbers. Therefore, all DCR accesses are privileged. All DCR numbers are reserved, and should be neither read nor written.

21.7.1 Directly Addressed DCRs

The following DCRs are directly accessed; that is, they are accessed using their DCR numbers.

Table 21-4. Directly Accessed DCRs

Register	DCR Number	Access	Description
DCRs Used for Indirect Access			
SDRAM0_CFGADDR	0x010	R/W	Memory Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	Memory Controller Data Register
EBC0_CFGADDR	0x012	R/W	Peripheral Controller Address Register
EBC0_CFGDATA	0x013	R/W	Peripheral Controller Data Register
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register
On-Chip Buses			
PLB0_BESR	0x084	R/Clear	PLB Bus Error Status Register
PLB0_BEAR	0x086	R/W	PLB Bus Error Address Register
PLB0_ACR	0x087	R/W	PLB Arbiter Control Register
POB0_BESR0	0x0A0	R/Clear	PLB to OPB Bus Error Status Register 0
POB0_BEAR	0x0A2	R	PLB to OPB Bus Error Address Register
POB0_BESR1	0x0A4	R/Clear	PLB to OPB Bus Error Status Register 1

Table 21-4. Directly Accessed DCRs

Register	DCR Number	Access	Description
Clocking, Power Management, and Chip Control			
CPC0_PLLMR	0x0B0	R/W	PLL Mode Register
CPC0_CR0	0x0B1	R/W	Chip Control Register 0
CPC0_CR1	0x0B2	R/W	Chip Control Register 1
CPC0_PSR	0x0B4	R	Chip Pin Strapping Register
CPC0_JTAGID	0x0B5	R	JTAG ID Register
CPC0_SR	0x0B8	R	CPM Status Register
CPC0_ER	0x0B9	R/W	CPM Enable Register
CPC0_FR	0x0BA	R/W	CPM Force Register
Universal Interrupt Controllers			
UIC0_SR	0x0C0	R/Clear	UIC0 Status Register
UIC0_ER	0x0C2	R/W	UIC0 Enable Register
UIC0_CR	0x0C3	R/W	UIC0 Critical Register
UIC0_PR	0x0C4	R/W	UIC0 Polarity Register
UIC0_TR	0x0C5	R/W	UIC0 Triggering Register
UIC0_MSR	0x0C6	R	UIC0 Masked Status Register
UIC0_VR	0x0C7	R	UIC0 Vector Register
UIC0_VCR	0x0C8	W	UIC0 Vector Configuration Register
Direct Memory Access			
DMA0_CR0	0x100	R/W	DMA Channel Control Register 0
DMA0_CT0	0x101	R/W	DMA Count Register 0
DMA0_DA0	0x102	R/W	DMA Destination Address Register 0
DMA0_SA0	0x103	R/W	DMA Source Address Register 0
DMA0_SG0	0x104	R/W	DMA Scatter/Gather Descriptor Address Register 0
DMA0_CR1	0x108	R/W	DMA Channel Control Register 1
DMA0_CT1	0x109	R/W	DMA Count Register 1
DMA0_DA1	0x10A	R/W	DMA Destination Address Register 1
DMA0_SA1	0x10B	R/W	DMA Source Address Register 1
DMA0_SG1	0x10C	R/W	DMA Scatter/Gather Descriptor Address Register 1
DMA0_CR2	0x110	R/W	DMA Channel Control Register 2
DMA0_CT2	0x111	R/W	DMA Count Register 2
DMA0_DA2	0x112	R/W	DMA Destination Address Register 2
DMA0_SA2	0x113	R/W	DMA Source Address Register 2
DMA0_SG2	0x114	R/W	DMA Scatter/Gather Descriptor Address Register 2
DMA0_CR3	0x118	R/W	DMA Channel Control Register 3
DMA0_CT3	0x119	R/W	DMA Count Register 3
DMA0_DA3	0x11A	R/W	DMA Destination Address Register 3
DMA0_SA3	0x11B	R/W	DMA Source Address Register 3
DMA0_SG3	0x11C	R/W	DMA Scatter/Gather Descriptor Address
DMA0_SR	0x120	R/Clear	DMA Status Register
DMA0_SGC	0x123	R/W	DMA Scatter/Gather Command Register

Table 21-4. Directly Accessed DCRs

Register	DCR Number	Access	Description
DMA0_SLP	0x125	R/W	DMA Sleep Mode Register
DMA0_POL	0x126	R/W	DMA Polarity Configuration Register

21.7.1.1 Indirectly Accessed DCRs

The DCRs for the SDRAM controller, external bus controller (EBC), and decompression controller are indirectly accessed.

21.7.2 Indirect Access of SDRAM Controller DCRs

The following procedure accesses the SDRAM controller DCRs listed in Table 21-5.

1. Write the offset from Table 21-6 to the Memory Controller Address Register (SDRAM0_CFGADDR).
2. Read data from or write data to the Memory Controller Data Register (SDRAM0_CFGDATA).

Table 21-5. SDRAM Controller DCR Usage

Register	DCR Number	Access	Description
SDRAM0_CFGADDR	0x010	R/W	Memory Controller Address Register
SDRAM0_CFGDATA	0x011	R/W	Memory Controller Data Register

Table 21-6. Offsets for SDRAM Controller Registers

Register	Offset	R/W	Description
SDRAM0_BESR0	0x00	R/Clear	Bus Error Syndrome Register 0
SDRAM0_BESR1	0x08	R/Clear	Bus Error Syndrome Register 1
SDRAM0_BEAR	0x10	R/W	Bus Error Address Register
SDRAM0_CFG	0x20	R/W	Memory Controller Options 1
SDRAM0_STATUS	0x24	R	SDRAM Controller Status
SDRAM0_RTR	0x30	R/W	Refresh Timer Register
SDRAM0_PMIT	0x34	R/W	Power Management Idle Timer
SDRAM0_B0CR	0x40	R/W	Memory Bank 0 Configuration Register
SDRAM0_B1CR	0x44	R/W	Memory Bank 1 Configuration Register
SDRAM0_B2CR	0x48	R/W	Memory Bank 2 Configuration Register
SDRAM0_B3CR	0x4C	R/W	Memory Bank 3 Configuration Register
SDRAM0_TR	0x80	R/W	SDRAM Timing Register 1
SDRAM0_ECCCFG	0x94	R/W	ECC Configuration
SDRAM0_ECCESR	0x98	R/Clear	ECC Error Status Register

21.7.3 Indirect Access of External Bus Controller DCRs

The following procedure accesses the EBC DCRs listed in Table 21-7.

1. Write the offset from Table 21-8 to the Peripheral Controller Address Register (EBC0_CFGADDR).
2. Read data from or write data to the Peripheral Controller Data Register (EBC0_CFGDATA).

Table 21-7. External Bus Controller DCR Usage

Register	DCR Number	Access	Description
EBC0_CFGADDR	0x012	R/W	Peripheral Controller Address Register
EBC0_CFGDATA	0x013	R/W	Peripheral Controller Data Register

Table 21-8. Offsets for External Bus Controller Registers

Register	Offset	Access	Description
EBC0_B0CR	0x00	R/W	Peripheral Bank 0 Configuration Register
EBC0_B1CR	0x01	R/W	Peripheral Bank 1 Configuration Register
EBC0_B2CR	0x02	R/W	Peripheral Bank 2 Configuration Register
EBC0_B3CR	0x03	R/W	Peripheral Bank 3 Configuration Register
EBC0_B4CR	0x04	R/W	Peripheral Bank 4 Configuration Register
EBC0_B5CR	0x05	R/W	Peripheral Bank 5 Configuration Register
EBC0_B6CR	0x06	R/W	Peripheral Bank 6 Configuration Register
EBC0_B7CR	0x07	R/W	Peripheral Bank 7 Configuration Register
EBC0_B0AP	0x10	R/W	Peripheral Bank 0 Access Parameters
EBC0_B1AP	0x11	R/W	Peripheral Bank 1 Access Parameters
EBC0_B2AP	0x12	R/W	Peripheral Bank 2 Access Parameters
EBC0_B3AP	0x13	R/W	Peripheral Bank 3 Access Parameters
EBC0_B4AP	0x14	R/W	Peripheral Bank 4 Access Parameters
EBC0_B5AP	0x15	R/W	Peripheral Bank 5 Access Parameters
EBC0_B6AP	0x16	R/W	Peripheral Bank 6 Access Parameters
EBC0_B7AP	0x17	R/W	Peripheral Bank 7 Access Parameters
EBC0_BEAR	0x20	R/W	Peripheral Bus Error Address Register
EBC0_BESR0	0x21	R/W	Peripheral Bus Error Status Register 0
EBC0_BESR1	0x22	R/W	Peripheral Bus Error Status Register 1
EBC0_CFG	0x23	R/W	External Peripheral Control Register

21.7.4 Indirect Access of Decompression Controller DCRs

The following procedure accesses the decompression controller DCRs listed in Table 21-9.

1. Write the offset from Table 21-10 to the Decompression Controller Address Register (DCP0_CFGADDR).
2. Read data from or write data to the Decompression Controller Data Register (DCP0_CFGDATA).

Table 21-9. Decompression Controller DCR Usage

Register	DCR Number	Access	Description
DCP0_CFGADDR	0x014	R/W	Decompression Controller Address Register
DCP0_CFGDATA	0x015	R/W	Decompression Controller Data Register

Table 21-10. Offsets for Decompression Controller Registers

Register	Offset	R/W	Description
DCP0_ITOR0	0x00	R/W	Index Table Origin Register 0
DCP0_ITOR1	0x01	R/W	Index Table Origin Register 1
DCP0_ITOR2	0x02	R/W	Index Table Origin Register 2
DCP0_ITOR3	0x03	R/W	Index Table Origin Register 3
DCP0_ADDR0	0x04	R/W	Address Decode Definition Register 0
DCP0_ADDR1	0x05	R/W	Address Decode Definition Register 1
DCP0_CFG	0x40	R/W	Decompression Controller Configuration Register
DCP0_ID	0x41	R	Decompression Controller ID Register
DCP0_VER	0x42	R	Decompression Controller Version Number Register
DCP0_PLBBEAR	0x50	R	Bus Error Address Register (PLB address)
DCP0_MEMBEAR	0x51	R	Bus Error Address Register (DCP to EBC address)
DCP0_ESR	0x52	R/Clear	Bus Error Status Register 0 (masters 0-3)
DCP0_RAM0–DCP0_RAM3FF	0x400–0x7FF	R/W	Decode Tables

21.7.5 Memory-Mapped Input/Output Registers

Some registers associated with on-chip peripherals are memory-mapped input/output (MMIO) registers. Such registers are mapped into the system memory space and are accessed using load/store instructions.

21.7.5.1 Directly Accessed MMIO Registers

Directly-accessed MMIO registers are accessed using load/store instructions that contain the register addresses. Table 21-11 lists the directly-accessed MMIO registers.

Table 21-11. Directly Accessed MMIO Registers

Register	Address	Access	Description
Serial Ports			
UART0_RBR	0xEF600300	R	UART 0 Receiver Buffer Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_THR		W	UART 0 Transmitter Holding Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_DLL		R/W	UART 0 Baud-rate Divisor Latch LSB **Note:** Set UART0_LCR[DLAB] = 1 to access.
UART0_IER	0xEF600301	R/W	UART 0 Interrupt Enable Register **Note:** Set UART0_LCR[DLAB] = 0 to access.
UART0_DLM		R/W	UART 0 Baud-rate Divisor Latch MSB **Note:** Set UART0_LCR[DLAB] = 1 to access.
UART0_IIR	0xEF600302	R	UART 0 Interrupt Identification Register
UART0_FCR	0xEF600302	W	UART 0 FIFO Control Register
UART0_LCR	0xEF600303	R/W	UART 0 Line Control Register
UART0_MCR	0xEF600304	R/W	UART 0 Modem Control Register
UART0_LSR	0xEF600305	R/W	UART 0 Line Status Register
UART0_MSR	0xEF600306	R/W	UART 0 Modem Status Register
UART0_SCR	0xEF600307	R/W	UART 0 Scratch Register
UART1_RBR	0xEF600400	R	UART 1 Receiver Buffer Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_THR		W	UART 1 Transmitter Holding Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_DLL		R/W	UART 1 Baud-rate Divisor Latch LSB **Note:** Set UART1_LCR[DLAB] = 1 to access.
UART1_IER	0xEF600401	R/W	UART 1 Interrupt Enable Register **Note:** Set UART1_LCR[DLAB] = 0 to access.
UART1_DLM		R/W	UART 1 Baud-rate Divisor Latch MSB **Note:** Set UART1_LCR[DLAB] = 1 to access.
UART1_IIR	0xEF600402	R	UART 1 Interrupt Identification Register
UART1_FCR	0xEF600402	W	UART 1 FIFO Control Register
UART1_LCR	0xEF600403	R/W	UART 1 Line Control Register
UART1_MCR	0xEF600404	R/W	UART 1 Modem Control Register
UART1_LSR	0xEF600405	R/W	UART 1 Line Status Register
UART1_MSR	0xEF600406	R/W	UART 1 Modem Status Register
UART1_SCR	0xEF600407	R/W	UART 1 Scratch Register

Table 21-11. Directly Accessed MMIO Registers

Register	Address	Access	Description
Inter-Integrated Circuit			
IIC0_MDBUF	0xEF600500	R/W	IIC0 Master Data Buffer
IIC0_SDBUF	0xEF600502	R/W	IIC0 Slave Data Buffer
IIC0_LMADR	0xEF600504	R/W	IIC0 Low Master Address
IIC0_HMADR	0xEF600505	R/W	IIC0 High Master Address
IIC0_CNTL	0xEF600506	R/W	IIC0 Control
IIC0_MDCNTL	0xEF600507	R/W	IIC0 Mode Control
IIC0_STS	0xEF600508	R/W	IIC0 Status
IIC0_EXTSTS	0xEF600509	R/W	IIC0 Extended Status
IIC0_LSADR	0xEF60050A	R/W	IIC0 Low Slave Address
IIC0_HSADR	0xEF60050B	R/W	IIC0 High Slave Address
IIC0_CLKDIV	0xEF60050C	R/W	IIC0 Clock Divide
IIC0_INTRMSK	0xEF60050D	R/W	IIC0 Interrupt Mask
IIC0_XFRCNT	0xEF60050E	R/W	IIC0 Transfer Count
IIC0_XTCNTLSS	0xEF60050F	R/W	IIC0 Extended Control and Slave Status
IIC0_DIRECTCNTL	0xEF600510	R/W	IIC0 Direct Control
OPB Arbiter			
OPBA0_PR	0xEF600600	R/W	OPB Arbiter Priority Register
OPBA0_CR	0xEF600601	R/W	OPB Arbiter Control Register
General-Purpose I/O			
GPIO0_OR	0xEF600700	R/W	GPIO0_IRO Output Register
GPIO0_TCR	0xEF600704	R/W	GPIO0_IRO Three-State Control Register
GPIO0_ODR	0xEF600718	R/W	GPIO0_IRO Open Drain Register
GPIO0_IR	0xEF60071C	R	GPIO0_IRO Input Register

21.8 Alphabetical Listing of Processor Core Registers

The following pages list the registers available in the processor core. For each register, the following information is supplied:

Register mnemonic and name

Cross reference to a detailed register description

Register type

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, field bit location, field name, and the function associated with various field values

CCR0

Core Configuration Register 0

SPR 0x3B3

See “Cache Control and Debugging Features” on page 4-11.

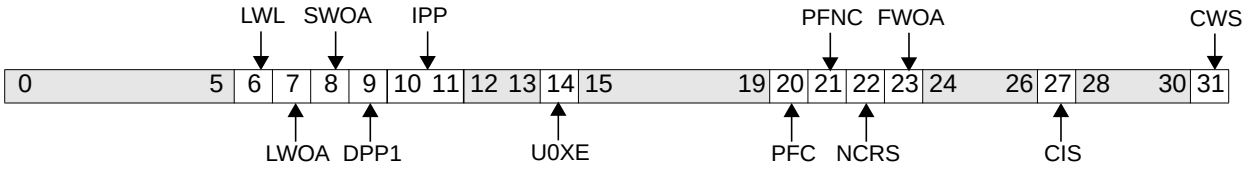


Figure 21-1. Core Configuration Register 0 (CCR0)

0:5		Reserved
6	LWL	Load Word as Line 0 The DCU performs load misses or non-cachable loads as words, halfwords, or bytes, as requested 1 For load misses or non-cachable loads, the DCU moves eight words (including the target word) into the line buffer
7	LWOA	Load Without Allocate 0 Load misses result in line fills 1 Load misses do not result in a line fill, but in non-cachable loads
8	SWOA	Store Without Allocate 0 Store misses result in line fills 1 Store misses do not result in line fills, but in non-cachable stores
9	DPP1	DCU PLB Priority Bit 1 0 DCU PLB priority 0 on bit 1 1 DCU PLB priority 1 on bit 1 **Note:** DCU logic dynamically controls DCU priority bit 0.
10:11	IPP	ICU PLB Priority Bits 0:1 00 Lowest ICU PLB priority 01 Next to lowest ICU PLB priority 10 Next to highest ICU PLB priority 11 Highest ICU PLB priority
12:13		Reserved
14	U0XE	Enable U0 Exception 0 Enables the U0 exception 1 Disables the U0 exception
15:19		Reserved
20	PFC	ICU Prefetching for Cachable Regions 0 Disables prefetching for cachable regions 1 Enables prefetching for cachable regions

21	PFNC	ICU Prefetching for Non-Cachable Regions 0 Disables prefetching for non-cachable regions 1 Enables prefetching for non-cachable regions
22	NCRS	Non-cachable ICU request size 0 Requests are for four-word lines 1 Requests are for eight-word lines
23	FWOA	Fetch Without Allocate 0 An ICU miss results in a line fill. 1 An ICU miss does not cause a line fill, but results in a non-cachable fetch.
24:26		Reserved
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

Condition Register

See “Condition Register (CR)” on page 3-11.

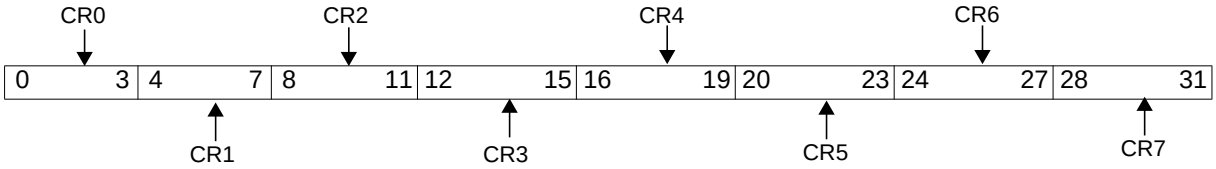


Figure 21-2. Condition Register (CR)

0:3	CR0	Condition Register Field 0
4:7	CR1	Condition Register Field 1
8:11	CR2	Condition Register Field 2
12:15	CR3	Condition Register Field 3
16:19	CR4	Condition Register Field 4
20:23	CR5	Condition Register Field 5
24:27	CR6	Condition Register Field 6
28:31	CR7	Condition Register Field 7

SPR 0x009

See “Count Register (CTR)” on page 3-6.

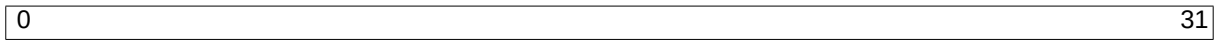


Figure 21-3. Count Register (CTR)

0:31		Count	Used as count for branch conditional with decrement instructions, or as address for branch-to-counter instructions.

DAC1–DAC2

Data Address Compare Registers

SPR 0x3F6–0x3F7

See “Data Address Compare Registers (DAC1–DAC2)” on page 11-14.

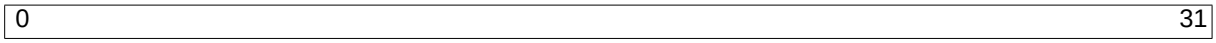


Figure 21-4. Data Address Compare Registers (DAC1–DAC2)

0:31		Data Address Compare (DAC) byte address	DBCR0[D1S] determines which address bits are examined.

SPR 0x3F2

See “Debug Control Registers” on page 11-9.

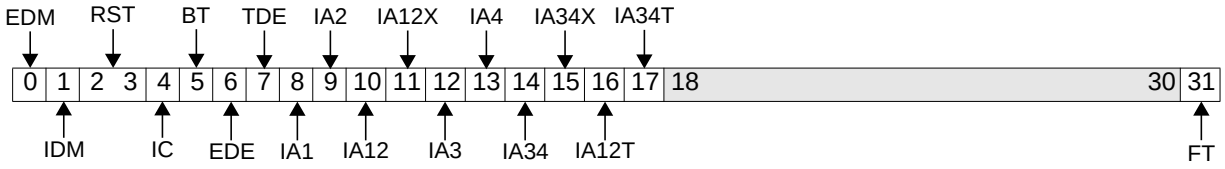


Figure 21-5. Debug Control Register 0 (DBCR0)

0	EDM	External Debug Mode 0 Disabled 1 Enabled	
1	IDM	Internal Debug Mode 0 Disabled 1 Enabled	
2:3	RST	Reset 00 No action 01 Core reset 10 Chip reset 11 System reset	Causes a processor reset request when set by software.
		Attention: Writing 01, 10, or 11 to this field causes a processor reset request.	
4	IC	Instruction Completion Debug Event 0 Disabled 1 Enabled	
5	BT	Branch Taken Debug Event 0 Disabled 1 Enabled	
6	EDE	Exception Debug Event 0 Disabled 1 Enabled	
7	TDE	Trap Debug Event 0 Disabled 1 Enabled	
8	IA1	IAC 1 Debug Event 0 Disabled 1 Enabled	
9	IA2	IAC 2 Debug Event 0 Disabled 1 Enabled	
10	IA12	Instruction Address Range Compare 1–2 0 Disabled 1 Enabled	Registers IAC1 and IAC2 define an address range used for IAC address comparisons.

DBCR0 (cont.)

Debug Control Register 0

11	IA12X	Enable Instruction Address Exclusive Range Compare 1–2 0 Inclusive 1 Exclusive	Selects the range defined by IAC1 and IAC2 to be inclusive or exclusive.
12	IA3	IAC 3 Debug Event 0 Disabled 1 Enabled	
13	IA4	IAC 4 Debug Event 0 Disabled 1 Enabled	
14	IA34	Instruction Address Range Compare 3–4 0 Disabled 1 Enabled	Registers IAC3 and IAC4 define an address range used for IAC address comparisons.
15	IA34X	Instruction Address Exclusive Range Compare 3–4 0 Inclusive 1 Exclusive	Selects range defined by IAC3 and IAC4 to be inclusive or exclusive.
16	IA12T	Instruction Address Range Compare 1-2 Toggle 0 Disabled 1 Enable	Toggles range 12 inclusive, exclusive DBCR[IA12X] on debug event.
17	IA34T	Instruction Address Range Compare 3–4 Toggle 0 Disabled 1 Enable	Toggles range 34 inclusive, exclusive DBCR[IA34X] on debug event.
18:30		Reserved	
31	FT	Freeze timers on debug event 0 Timers not frozen 1 Timers frozen	

SPR 0x3BD

See “Debug Control Registers” on page 11-9.

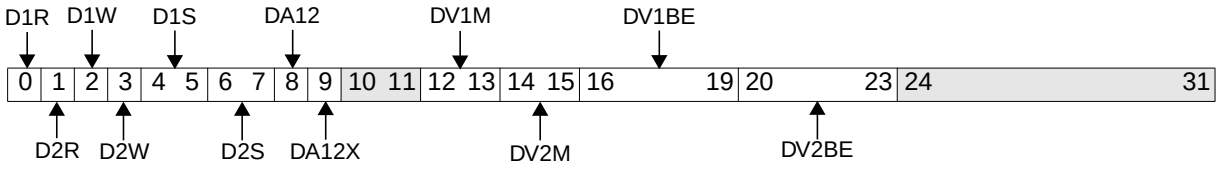


Figure 21-6. Debug Control Register 1 (DBCR1)

0	D1R	DAC1 Read Debug Event 0 Disabled 1 Enabled	
1	D2R	DAC 2 Read Debug Event 0 Disabled 1 Enabled	
2	D1W	DAC 1 Write Debug Event 0 Disabled 1 Enabled	
3	D2W	DAC 2 Write Debug Event 0 Disabled 1 Enabled	
4:5	D1S	DAC 1 Size 00 Compare all bits 01 Ignore lsb (least significant bit) 10 Ignore two lsbs 11 Ignore five lsbs	Address bits used in the compare: Byte address Halfword address Word address Cache line (8-word) address
6:7	D2S	DAC 2 Size 00 Compare all bits 01 Ignore lsb (least significant bit) 10 Ignore two lsbs 11 Ignore five lsbs	Address bits used in the compare: Byte address Halfword address Word address Cache line (8-word) address
8	DA12	Enable Data Address Range Compare 1:2 0 Disabled 1 Enabled	Registers DAC1 and DAC2 define an address range used for DAC address comparisons
9	DA12X	Data Address Exclusive Range Compare 1:2 0 Inclusive 1 Exclusive	Selects range defined by DAC1 and DAC2 to be inclusive or exclusive
10:11		Reserved	

DBCR1 (cont.)

Debug Control Register 1

12:13	DV1M	Data Value Compare 1 Mode 00 Undefined 01 AND 10 OR 11 AND-OR	Type of data comparison used: All bytes selected by DBCR1[DV1BE] must compare to the appropriate bytes of DVC1. One of the bytes selected by DBCR1[DV1BE] must compare to the appropriate bytes of DVC1. The upper halfword or lower halfword must compare to the appropriate halfword in DVC1. When performing halfword compares set DBCR1[DV1BE] = 0011, 1100, or 1111.
14:15	DV2M	Data Value Compare 2 Mode 00 Undefined 01 AND 10 OR 11 AND-OR	Type of data comparison used All bytes selected by DBCR1[DV2BE] must compare to the appropriate bytes of DVC2. One of the bytes selected by DBCR1[DV2BE] must compare to the appropriate bytes of DVC2. The upper halfword or lower halfword must compare to the appropriate halfword in DVC2. When performing halfword compares set DBCR1[DV2BE] = 0011, 1100, or 1111.
16:19	DV1BE	Data Value Compare 1 Byte 0 Disabled 1 Enabled	Selects which data bytes to use in data value comparison
20:23	DV2BE	Data Value Compare 2 Byte 0 Disabled 1 Enabled	Selects which data bytes to use in data value comparison
24:31		Reserved	

SPR 0x3F0 Read/Clear

See “Debug Status Register (DBSR)” on page 11-12.

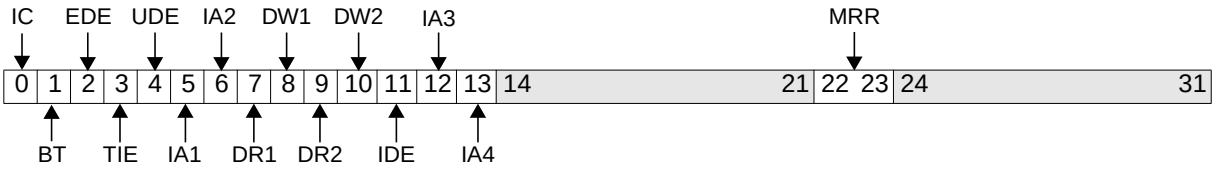


Figure 21-7. Debug Status Register (DBSR)

0	IC	Instruction Completion Debug Event 0 Event did not occur 1 Event occurred
1	BT	Branch Taken Debug Event 0 Event did not occur 1 Event occurred
2	EDE	Exception Debug Event 0 Event did not occur 1 Event occurred
3	TIE	Trap Instruction Debug Event 0 Event did not occur 1 Event occurred
4	UDE	Unconditional Debug Event 0 Event did not occur 1 Event occurred
5	IA1	IAC1 Debug Event 0 Event did not occur 1 Event occurred
6	IA2	IAC2 Debug Event 0 Event did not occur 1 Event occurred
7	DR1	DAC1 Read Debug Event 0 Event did not occur 1 Event occurred
8	DW1	DAC1 Write Debug Event 0 Event did not occur 1 Event occurred
9	DR2	DAC2 Read Debug Event 0 Event did not occur 1 Event occurred
10	DW2	DAC2 Write Debug Event 0 Event did not occur 1 Event occurred

DBSR (cont.)

Debug Status Register

11	IDE	Imprecise Debug Event 0 No circumstance that would cause a debug event (if MSR[DE] = 1) occurred 1 A debug event would have occurred, but debug exceptions were disabled (MSR[DE] = 1)	
12	IA3	IAC3 Debug Event 0 Event did not occur 1 Event occurred	
13	IA4	IAC4 Debug Event 0 Event did not occur 1 Event occurred	
14:21		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	

SPR 0x3FA

See “Real-mode Storage Attribute Control” on page 5-17.

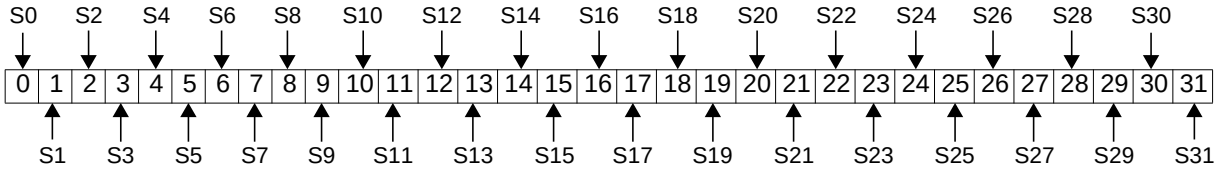


Figure 21-8. Data Cache Cachability Register (DCCR)

0	S0	0 Noncachable 1 Cachable	0x0000 0000–0x07FF FFFF
1	S1	0 Noncachable 1 Cachable	0x0800 0000–0x0FFF FFFF
2	S2	0 Noncachable 1 Cachable	0x1000 0000–0x17FF FFFF
3	S3	0 Noncachable 1 Cachable	0x1800 0000–0x1FFF FFFF
4	S4	0 Noncachable 1 Cachable	0x2000 0000–0x27FF FFFF
5	S5	0 Noncachable 1 Cachable	0x2800 0000–0x2FFF FFFF
6	S6	0 Noncachable 1 Cachable	0x3000 0000–0x37FF FFFF
7	S7	0 Noncachable 1 Cachable	0x3800 0000–0x3FFF FFFF
8	S8	0 Noncachable 1 Cachable	0x4000 0000–0x47FF FFFF
9	S9	0 Noncachable 1 Cachable	0x4800 0000–0x4FFF FFFF
10	S10	0 Noncachable 1 Cachable	0x5000 0000–0x57FF FFFF
11	S11	0 Noncachable 1 Cachable	0x5800 0000–0x5FFF FFFF
12	S12	0 Noncachable 1 Cachable	0x6000 0000–0x67FF FFFF
13	S13	0 Noncachable 1 Cachable	0x6800 0000–0x6FFF FFFF
14	S14	0 Noncachable 1 Cachable	0x7000 0000–0x77FF FFFF
15	S15	0 Noncachable 1 Cachable	0x7800 0000–0x7FFF FFFF

DCCR (cont.)

Data Cache Cacheability Register

16	S16	0 Noncachable 1 Cachable	0x8000 0000 – 0x87FF FFFF
17	S17	0 Noncachable 1 Cachable	0x8800 0000 – 0x8FFF FFFF
18	S18	0 Noncachable 1 Cachable	0x9000 0000 – 0x97FF FFFF
19	S19	0 Noncachable 1 Cachable	0x9800 0000 – 0x9FFF FFFF
20	S20	0 Noncachable 1 Cachable	0xA000 0000 – 0xA7FF FFFF
21	S21	0 Noncachable 1 Cachable	0xA800 0000 – 0xAFFF FFFF
22	S22	0 Noncachable 1 Cachable	0xB000 0000 – 0xB7FF FFFF
23	S23	0 Noncachable 1 Cachable	0xB800 0000 – 0xBFFF FFFF
24	S24	0 Noncachable 1 Cachable	0xC000 0000 – 0xC7FF FFFF
25	S25	0 Noncachable 1 Cachable	0xC800 0000 – 0xCFFF FFFF
26	S26	0 Noncachable 1 Cachable	0xD000 0000 – 0xD7FF FFFF
27	S27	0 Noncachable 1 Cachable	0xD800 0000 – 0xDFFF FFFF
28	S28	0 Noncachable 1 Cachable	0xE000 0000 – 0xE7FF FFFF
29	S29	0 Noncachable 1 Cachable	0xE800 0000 – 0xEFFF FFFF
30	S30	0 Noncachable 1 Cachable	0xF000 0000 – 0xF7FF FFFF
31	S31	0 Noncachable 1 Cachable	0xF800 0000 – 0xFFFF FFFF

SPR 0x3BA

See “Real-mode Storage Attribute Control” on page 5-17.

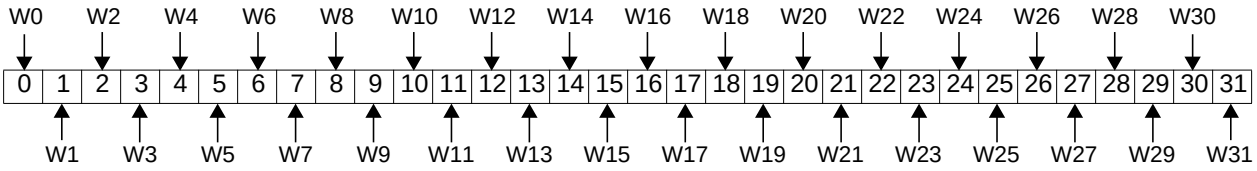


Figure 21-9. Data Cache Write-through Register (DCWR)

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

DCWR (cont.)

Data Cache Write-through Register

16	W16	0 Write-back 1 Write-through	0x8000 0000 – 0x87FF FFFF
17	W17	0 Write-back 1 Write-through	0x8800 0000 – 0x8FFF FFFF
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

SPR 0x3D5

See “Data Exception Address Register (DEAR)” on page 9-28.

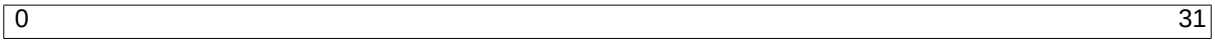


Figure 21-10. Data Exception Address Register (DEAR)

0:31		Address of Data Error (synchronous)

DVC1–DVC2

Data Value Compare Registers

SPR 0x3B6–0x3B7

See “Data Value Compare Registers (DVC1–DVC2)” on page 11-15.

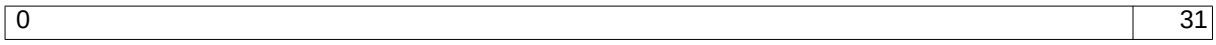


Figure 21-11. Data Value Compare Registers (DVC1–DVC2)

0:31		Data Value to Compare

SPR 0x3D4

See “Exception Syndrome Register (ESR)” on page 9-26.

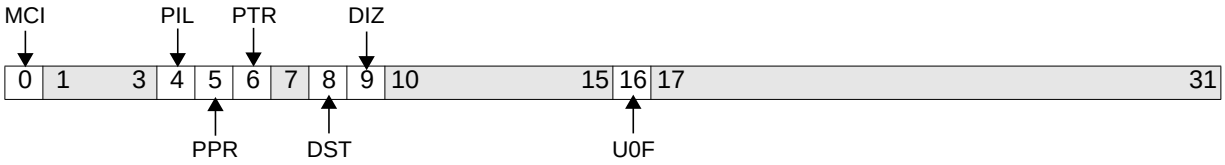


Figure 21-12. Exception Syndrome Register (ESR)

0	MCI	Machine check—instruction 0 Instruction machine check did not occur. 1 Instruction machine check occurred.
1:3		Reserved
4	PIL	Program interrupt—illegal 0 Illegal Instruction error did not occur. 1 Illegal Instruction error occurred.
5	PPR	Program interrupt—privileged 0 Privileged instruction error did not occur. 1 Privileged instruction error occurred.
6	PTR	Program interrupt—trap 0 Trap with successful compare did not occur. 1 Trap with successful compare occurred.
7		Reserved
8	DST	Data storage interrupt—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 interrupt—zone fault 0 Excepting condition was not a zone fault. 1 Excepting condition was a zone fault.
10:15		Reserved
16	U0F	Data storage interrupt—U0 fault 0 Excepting instruction did not cause a U0 fault. 1 Excepting instruction did cause a U0 fault.
17:31		Reserved

EVPR

Exception Vector Prefix Register

SPR 0x3D6

See “Exception Vector Prefix Register (EVPR)” on page 9-26.

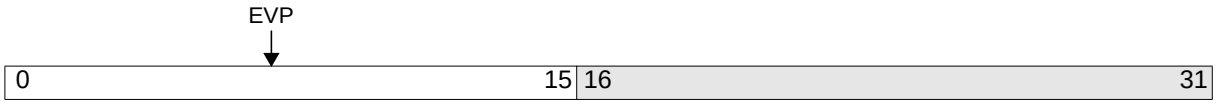


Figure 21-13. Exception Vector Prefix Register (EVPR)

0:15	EVP	Exception Vector Prefix
16:31		Reserved

See “General Purpose Registers (R0-R31)” on page 3-5.

Figure 21-14. General Purpose Registers (R0-R31)

0:31		General Purpose Register data

IAC1–IAC4

Instruction Address Compare Registers

SPR 0x3F4–0x3F5

See “Instruction Address Compare Registers (IAC1–IAC4)” on page 11-14.

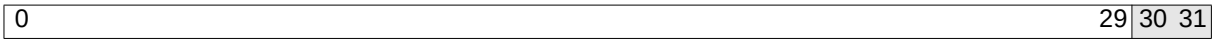


Figure 21-15. Instruction Address Compare Registers (IAC1–IAC4)

0:29		Instruction Address Compare word address	Omit two low-order bits of complete address.
30:31		Reserved	

SPR 0x3FB

See “Real-mode Storage Attribute Control” on page 5-17.

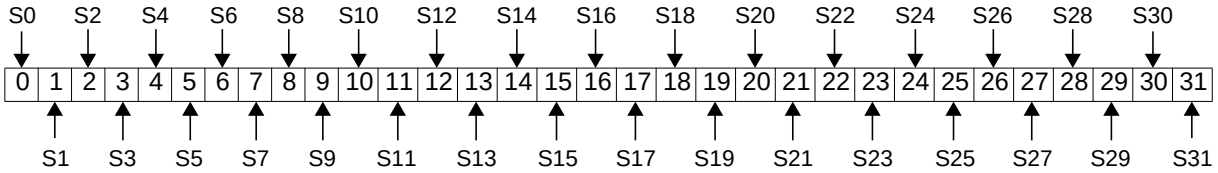


Figure 21-16. Instruction Cache Cachability Register (ICCR)

0	S0	0 Noncachable 1 Cachable	0x0000 0000–0x07FF FFFF
1	S1	0 Noncachable 1 Cachable	0x0800 0000–0x0FFF FFFF
2	S2	0 Noncachable 1 Cachable	0x1000 0000–0x17FF FFFF
3	S3	0 Noncachable 1 Cachable	0x1800 0000–0x1FFF FFFF
4	S4	0 Noncachable 1 Cachable	0x2000 0000–0x27FF FFFF
5	S5	0 Noncachable 1 Cachable	0x2800 0000–0x2FFF FFFF
6	S6	0 Noncachable 1 Cachable	0x3000 0000–0x37FF FFFF
7	S7	0 Noncachable 1 Cachable	0x3800 0000–0x3FFF FFFF
8	S8	0 Noncachable 1 Cachable	0x4000 0000–0x47FF FFFF
9	S9	0 Noncachable 1 Cachable	0x4800 0000–0x4FFF FFFF
10	S10	0 Noncachable 1 Cachable	0x5000 0000–0x57FF FFFF
11	S11	0 Noncachable 1 Cachable	0x5800 0000–0x5FFF FFFF
12	S12	0 Noncachable 1 Cachable	0x6000 0000–0x67FF FFFF
13	S13	0 Noncachable 1 Cachable	0x6800 0000–0x6FFF FFFF
14	S14	0 Noncachable 1 Cachable	0x7000 0000–0x77FF FFFF
15	S15	0 Noncachable 1 Cachable	0x7800 0000–0x7FFF FFFF

ICCR (cont.)

Instruction Cache Cacheability Register

16	S16	0 Noncachable 1 Cachable	0x8000 0000 – 0x87FF FFFF
17	S17	0 Noncachable 1 Cachable	0x8800 0000 – 0x8FFF FFFF
18	S18	0 Noncachable 1 Cachable	0x9000 0000 – 0x97FF FFFF
19	S19	0 Noncachable 1 Cachable	0x9800 0000 – 0x9FFF FFFF
20	S20	0 Noncachable 1 Cachable	0xA000 0000 – 0xA7FF FFFF
21	S21	0 Noncachable 1 Cachable	0xA800 0000 – 0xAFFF FFFF
22	S22	0 Noncachable 1 Cachable	0xB000 0000 – 0xB7FF FFFF
23	S23	0 Noncachable 1 Cachable	0xB800 0000 – 0xBFFF FFFF
24	S24	0 Noncachable 1 Cachable	0xC000 0000 – 0xC7FF FFFF
25	S25	0 Noncachable 1 Cachable	0xC800 0000 – 0xCFFF FFFF
26	S26	0 Noncachable 1 Cachable	0xD000 0000 – 0xD7FF FFFF
27	S27	0 Noncachable 1 Cachable	0xD800 0000 – 0xDFFF FFFF
28	S28	0 Noncachable 1 Cachable	0xE000 0000 – 0xE7FF FFFF
29	S29	0 Noncachable 1 Cachable	0xE800 0000 – 0xEFFF FFFF
30	S30	0 Noncachable 1 Cachable	0xF000 0000 – 0xF7FF FFFF
31	S31	0 Noncachable 1 Cachable	0xF800 0000 – 0xFFFF FFFF

SPR 0x3D3 Read-Only

See “ICU Debugging” on page 4-14.

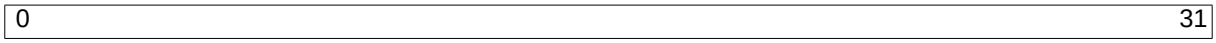


Figure 21-17. Instruction Cache Debug Data Register (ICDBDR)

0:31		Instruction cache information	See **icread** on page 20-68.

ICU tag information is placed into the ICDBDR as shown:

0:21	TAG	Cache Tag
22:26		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

Link Register

SPR 0x008

See “Link Register (LR)” on page 3-7.

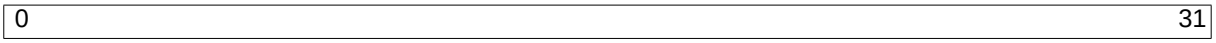


Figure 21-18. Link Register (LR)

0:31		Link Register contents	If (LR) represents an instruction address, $LR_{30:31}$ should be 0.

See “Machine State Register (MSR)” on page 9-23.

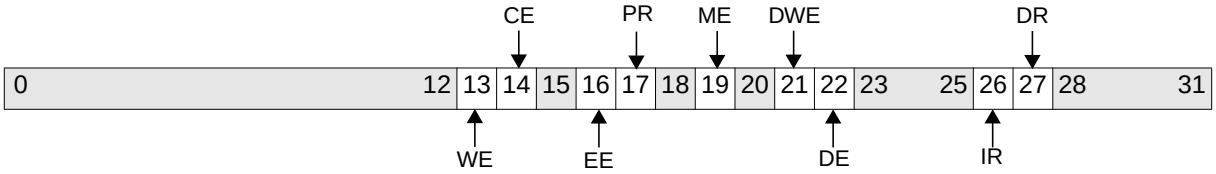


Figure 21-19. Machine State Register (MSR)

0:12		Reserved	
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 interrupt 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 time-out interrupts.
15		Reserved	
16	EE	External Interrupt Enable 0 Asynchronous interrupts (external to the processor core) are disabled. 1 Asynchronous interrupts are enabled.	Controls the non-critical external interrupt input, PIT, and FIT 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 interrupts are disabled. 1 Machine check interrupts are enabled.	
20		Reserved	
21	DWE	Debug Wait Enable 0 Debug wait mode is disabled. 1 Debug wait mode is enabled.	
22	DE	Debug Interrupts Enable 0 Debug interrupts are disabled. 1 Debug interrupts are enabled.	
23:25		Reserved	
26	IR	Instruction Relocate 0 Instruction address translation is disabled. 1 Instruction address translation is enabled.	

MSR (cont.)

Machine State Register

27	DR	Data Relocate 0 Data address translation is disabled. 1 Data address translation is enabled.
28:31		Reserved

SPR 0x3B1

See “Address Translation” on page 5-1.

Figure 21-20. Process ID (PID)

0:23		Reserved
24:31		Process ID

PIT

Programmable Interval Timer

SPR 0x3DB

See “Programmable Interval Timer (PIT)” on page 10-4.

Figure 21-21. Programmable Interval Timer (PIT)

0:31		Programmed interval remaining	Number of clocks remaining until the PIT event

SPR 0x11F Read-Only

See “Processor Version Register (PVR)” on page 3-11.

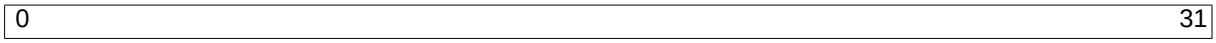


Figure 21-22. Processor Version Register (PVR)

0:31		Assigned PVR value

SGR

Storage Guarded Register

SPR 0x3B9

See “Real-mode Storage Attribute Control” on page 5-17.

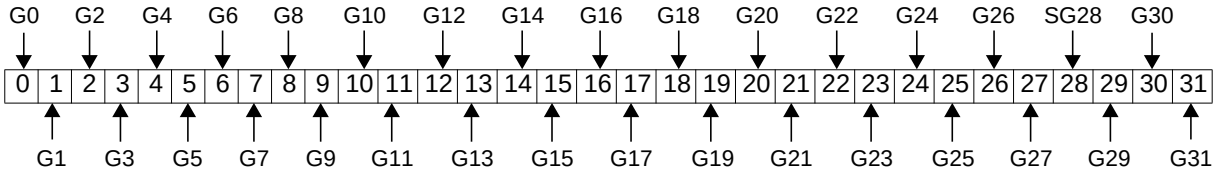


Figure 21-23. Storage Guarded Register (SGR)

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
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

SLER

Storage Little-Endian Register

SPR 0x3BB

See “Real-mode Storage Attribute Control” on page 5-17.

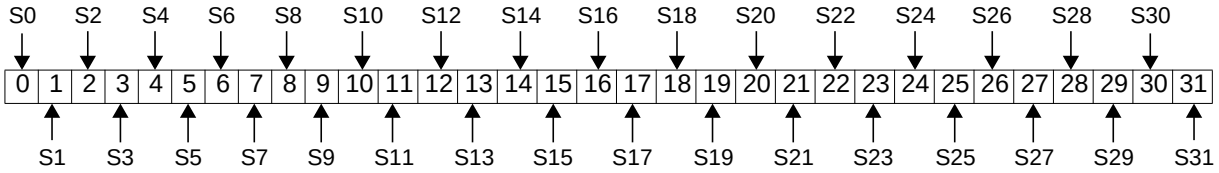


Figure 21-24. Storage Little-Endian Register (SLER)

0	S0	0 Big endian 1 Little endian	0x0000 0000–0x07FF FFFF
1	S1	0 Big endian 1 Little endian	0x0800 0000–0x0FFF FFFF
2	S2	0 Big endian 1 Little endian	0x1000 0000–0x17FF FFFF
3	S3	0 Big endian 1 Little endian	0x1800 0000–0x1FFF FFFF
4	S4	0 Big endian 1 Little endian	0x2000 0000–0x27FF FFFF
5	S5	0 Big endian 1 Little endian	0x2800 0000–0x2FFF FFFF
6	S6	0 Big endian 1 Little endian	0x3000 0000–0x37FF FFFF
7	S7	0 Big endian 1 Little endian	0x3800 0000–0x3FFF FFFF
8	S8	0 Big endian 1 Little endian	0x4000 0000–0x47FF FFFF
9	S9	0 Big endian 1 Little endian	0x4800 0000–0x4FFF FFFF
10	S10	0 Big endian 1 Little endian	0x5000 0000–0x57FF FFFF
11	S11	0 Big endian 1 Little endian	0x5800 0000–0x5FFF FFFF
12	S12	0 Big endian 1 Little endian	0x6000 0000–0x67FF FFFF
13	S13	0 Big endian 1 Little endian	0x6800 0000–0x6FFF FFFF
14	S14	0 Big endian 1 Little endian	0x7000 0000–0x77FF FFFF
15	S15	0 Big endian 1 Little endian	0x7800 0000–0x7FFF FFFF

16	S16	0 Big endian 1 Little endian	0x8000 0000 – 0x87FF FFFF
17	S17	0 Big endian 1 Little endian	0x8800 0000 – 0x8FFF FFFF
18	S18	0 Big endian 1 Little endian	0x9000 0000 – 0x97FF FFFF
19	S19	0 Big endian 1 Little endian	0x9800 0000 – 0x9FFF FFFF
20	S20	0 Big endian 1 Little endian	0xA000 0000 – 0xA7FF FFFF
21	S21	0 Big endian 1 Little endian	0xA800 0000 – 0xAFFF FFFF
22	S22	0 Big endian 1 Little endian	0xB000 0000 – 0xB7FF FFFF
23	S23	0 Big endian 1 Little endian	0xB800 0000 – 0xBFFF FFFF
24	S24	0 Big endian 1 Little endian	0xC000 0000 – 0xC7FF FFFF
25	S25	0 Big endian 1 Little endian	0xC800 0000 – 0xCFFF FFFF
26	S26	0 Big endian 1 Little endian	0xD000 0000 – 0xD7FF FFFF
27	S27	0 Big endian 1 Little endian	0xD800 0000 – 0xDFFF FFFF
28	S28	0 Big endian 1 Little endian	0xE000 0000 – 0xE7FF FFFF
29	S29	0 Big endian 1 Little endian	0xE800 0000 – 0xEFFF FFFF
30	S30	0 Big endian 1 Little endian	0xF000 0000 – 0xF7FF FFFF
31	S31	0 Big endian 1 Little endian	0xF800 0000 – 0xFFFF FFFF

SPRG0–SPRG7

Special Purpose Registers General

SPR 0x104–0x107 (User Read-only); 0x110–0x117 (Privileged Read/Write)

See “Special Purpose Register General (SPRG0–SPRG7)” on page 3-10.

Figure 21-25. Special Purpose Registers General (SPRG0–SPRG7)

0:31		General data	Software value; no hardware usage.

SPR 0x01A

See “Save/Restore Registers 0 and 1 (SRR0–SRR1)” on page 9-24.

.

Figure 21-26. 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

Save/Restore Register 1

SPR 0x01B

See “Save/Restore Registers 0 and 1 (SRR0–SRR1)” on page 9-24.

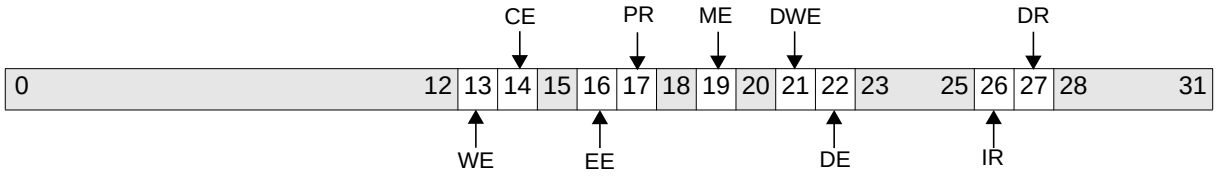


Figure 21-27. Save/Restore Register 1 (SRR1)

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

SPR 0x3DE

See “Save/Restore Registers 2 and 3 (SRR2–SRR3)” on page 9-25.

.

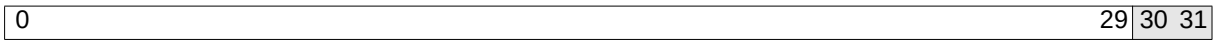


Figure 21-28. Save/Restore Register 2 (SRR2)

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

SRR3

Save/Restore Register 3

SPR 0x3DF

See “Save/Restore Registers 2 and 3 (SRR2–SRR3)” on page 9-25.

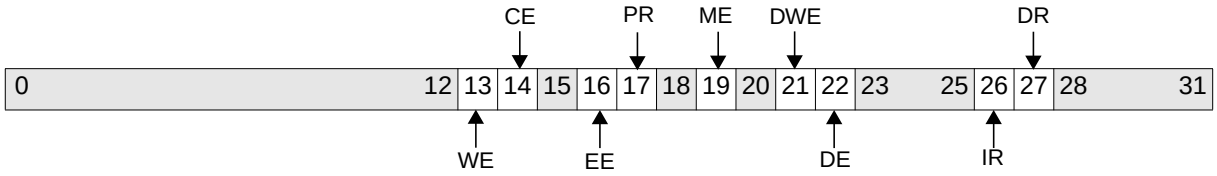


Figure 21-29. Save/Restore Register 3 (SRR3)

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

SPR 0x3BC

See “Real-mode Storage Attribute Control” on page 5-17.

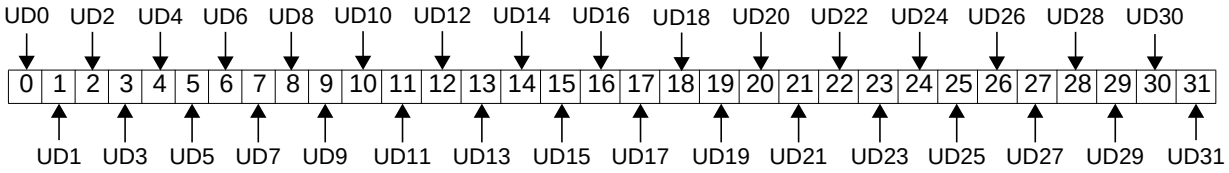


Figure 21-30. Storage User-defined 0 Register (SU0R)

0	UD0	0 Storage compression is off 1 Storage compression is on	0x0000 0000–0x07FF FFFF
1	UD1	0 Storage compression is off 1 Storage compression is on	0x0800 0000–0x0FFF FFFF
2	UD2	0 Storage compression is off 1 Storage compression is on	0x1000 0000–0x17FF FFFF
3	UD3	0 Storage compression is off 1 Storage compression is on	0x1800 0000–0x1FFF FFFF
4	UD4	0 Storage compression is off 1 Storage compression is on	0x2000 0000–0x27FF FFFF
5	UD5	0 Storage compression is off 1 Storage compression is on	0x2800 0000–0x2FFF FFFF
6	UD6	0 Storage compression is off 1 Storage compression is on	0x3000 0000–0x37FF FFFF
7	UD7	0 Storage compression is off 1 Storage compression is on	0x3800 0000–0x3FFF FFFF
8	UD8	0 Storage compression is off 1 Storage compression is on	0x4000 0000–0x47FF FFFF
9	UD9	0 Storage compression is off 1 Storage compression is on	0x4800 0000–0x4FFF FFFF
10	UD10	0 Storage compression is off 1 Storage compression is on	0x5000 0000–0x57FF FFFF
11	UD11	0 Storage compression is off 1 Storage compression is on	0x5800 0000–0x5FFF FFFF
12	UD12	0 Storage compression is off 1 Storage compression is on	0x6000 0000–0x67FF FFFF
13	UD13	0 Storage compression is off 1 Storage compression is on	0x6800 0000–0x6FFF FFFF
14	UD14	0 Storage compression is off 1 Storage compression is on	0x7000 0000–0x77FF FFFF
15	UD15	0 Storage compression is off 1 Storage compression is on	0x7800 0000–0x7FFF FFFF

SU0R (cont.)

Storage User-Defined 0 Register

16	UD16	0 Storage compression is off 1 Storage compression is on	0x8000 0000 – 0x87FF FFFF
17	UD17	0 Storage compression is off 1 Storage compression is on	0x8800 0000 – 0x8FFF FFFF
18	UD18	0 Storage compression is off 1 Storage compression is on	0x9000 0000 – 0x97FF FFFF
19	UD19	0 Storage compression is off 1 Storage compression is on	0x9800 0000 – 0x9FFF FFFF
20	UD20	0 Storage compression is off 1 Storage compression is on	0xA000 0000 – 0xA7FF FFFF
21	UD21	0 Storage compression is off 1 Storage compression is on	0xA800 0000 – 0xAFFF FFFF
22	UD22	0 Storage compression is off 1 Storage compression is on	0xB000 0000 – 0xB7FF FFFF
23	UD23	0 Storage compression is off 1 Storage compression is on	0xB800 0000 – 0xBFFF FFFF
24	UD24	0 Storage compression is off 1 Storage compression is on	0xC000 0000 – 0xC7FF FFFF
25	UD25	0 Storage compression is off 1 Storage compression is on	0xC800 0000 – 0xCFFF FFFF
26	UD26	0 Storage compression is off 1 Storage compression is on	0xD000 0000 – 0xD7FF FFFF
27	UD27	0 Storage compression is off 1 Storage compression is on	0xD800 0000 – 0xDFFF FFFF
28	UD28	0 Storage compression is off 1 Storage compression is on	0xE000 0000 – 0xE7FF FFFF
29	UD29	0 Storage compression is off 1 Storage compression is on	0xE800 0000 – 0xEFFF FFFF
30	UD30	0 Storage compression is off 1 Storage compression is on	0xF000 0000 – 0xF7FF FFFF
31	UD31	0 Storage compression is off 1 Storage compression is on	0xF800 0000 – 0xFFFF FFFF

TBR 0x10C (Read-only); SPR 0x11C (Privileged write-only)

See “Time Base” on page 10-2.

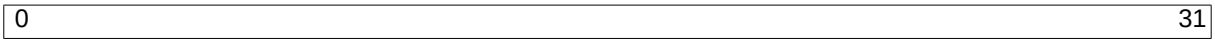


Figure 21-31. Time Base Lower (TBL)

0:31		Time Base Lower	Current count; low-order 32 bits of time base.

TBU

Time Base Upper

TBR 0x10D (Read-only); SPR 0x11D (Privileged write-only)

See “Time Base” on page 10-2.

Figure 21-32. Time Base Upper (TBU)

0:31		Time Base Upper	Current count, high-order 32 bits of time base.

SPR 0x3DA

See “Timer Control Register (TCR)” on page 10-9.

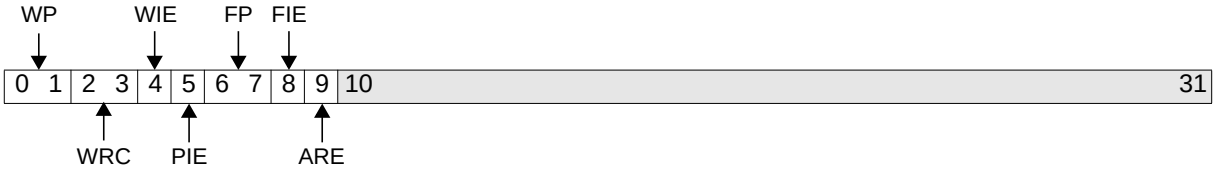


Figure 21-33. Timer Control Register (TCR)

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 will occur. 01 Core reset will be forced by the Watchdog. 10 Chip reset will be forced by the Watchdog. 11 System reset will be 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 watchdog interrupt. 1 Enable watchdog 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

Timer Status Register

SPR 0x3D8 Read/Clear

See “Timer Status Register (TSR)” on page 10-8.

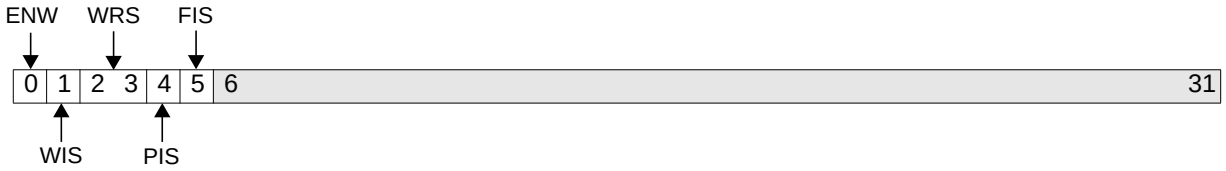


Figure 21-34. Timer Status Register (TSR)

0	ENW	Enable Next Watchdog 0 Action on next watchdog event is to set TSR[ENW] = 1. 1 Action on next watchdog event is governed by TSR[WIS].	Software must reset TSR[ENW] = 0 after each watchdog timer event.
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	

SPR 0x100 (User R/W)

See “Special Purpose Register General (SPRG0–SPRG7)” on page 3-10.

Figure 21-35. User SPR General 0 (USPRG0)

0:31		General data	Software value; no hardware usage.

XER

Fixed Point Exception Register

SPR 0x001

See “Fixed Point Exception Register (XER)” on page 3-7.

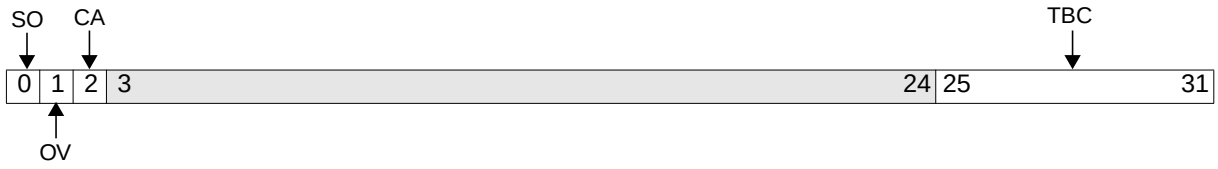


Figure 21-36. Fixed Point Exception Register (XER)

0	SO	Summary Overflow 0 No overflow has occurred. 1 Overflow has occurred.	Can be *set* by **mtspr** or by using “o” form instructions; 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 by using “o” form instructions; can be *reset* by **mtspr**, by **mcrxr**, or “o” form instructions.
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; 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**.

SPR 0x3B0

See “Zone Protection” on page 5-13.

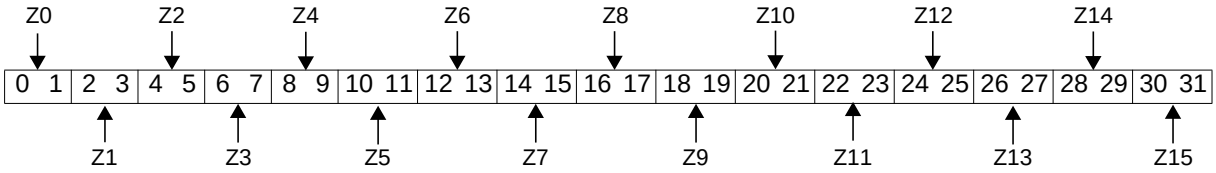


Figure 21-37. Zone Protection Register (ZPR)

0:1	Z0	TLB page access control for all pages in this zone.	
		In the problem state (MSR[PR] = 1): 00 No access 01 Access controlled by applicable TLB_entry[EX, WR] 10 Access controlled by applicable TLB_entry[EX, WR] 11 Accessed as if execute and write permissions (TLB_entry[EX, WR]) are granted.	In the supervisor state (MSR[PR] = 0): 00 Access controlled by applicable TLB_entry[EX, WR] 01 Access controlled by applicable TLB_entry[EX, WR] 10 Access controlled by applicable TLB_entry[EX, WR] 11 Accessed as if execute and write permissions (TLB_entry[EX, WR]) are granted.
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.	

21.9 Alphabetical Listing of Chip Control and Peripheral Registers

The following pages list the chip control and peripheral registers available in the PPC405CR. For each register, the following information is supplied:

Register mnemonic and name

Cross reference to a detailed register description

Register type

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, field bit location, field name, and the function associated with various field values.

CPC0_CR0

Chip Control Register 0

DCR 0x0B1

See “Chip Control Register 0 (CPC0_CR0)” on page 6-9.

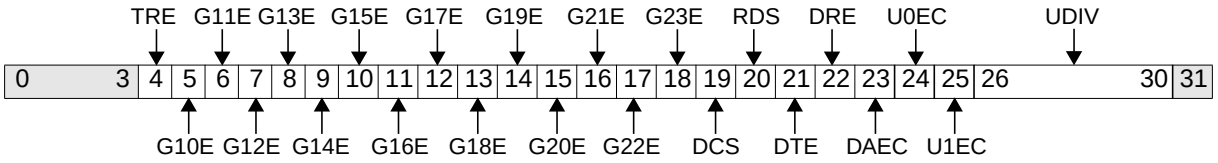


Figure 21-38. Chip Control Register 0 (CPC0_CR0)

0:3		Reserved	
4	TRE	CPU Trace Enable 0 GPIO1-9 are enabled 1 GPIO1-9 are disabled	Trace interface cannot be used when GPIO is enabled.
5	G10E	GPIO 10 Enable 0 Enable $\overline{\text{PerCS1}}$ as a chip select 1 Enable $\overline{\text{PerCS1}}$ as GPIO10	
6	G11E	GPIO 11 Enable 0 Enable $\overline{\text{PerCS2}}$ as a chip select 1 Enable $\overline{\text{PerCS2}}$ as GPIO11	
7	G12E	GPIO 12 Enable 0 Enable $\overline{\text{PerCS3}}$ as a chip select 1 Enable $\overline{\text{PerCS3}}$ as GPIO12	
8	G13E	GPIO 13 Enable 0 Enable $\overline{\text{PerCS4}}$ as a chip select 1 Enable $\overline{\text{PerCS4}}$ as GPIO13	
9	G14E	GPIO 14 Enable 0 Enable $\overline{\text{PerCS5}}$ as a chip select 1 Enable $\overline{\text{PerCS5}}$ as GPIO14	
10	G15E	GPIO 15 Enable 0 Enable $\overline{\text{PerCS6}}$ as a chip select 1 Enable $\overline{\text{PerCS6}}$ as GPIO15	
11	G16E	GPIO 16 Enable 0 Enable $\overline{\text{PerCS7}}$ as a chip select 1 Enable $\overline{\text{PerCS7}}$ as GPIO16	
12	G17E	GPIO 17 Enable 0 Enable interrupt IRQ0 as an interrupt 1 Enable interrupt IRQ0 as GPIO17	The purpose of GPIO_17_EN through GPIO_23_EN is to isolate the interrupt controller from activity on a shared pin when that pin is being used as a GPIO. For instance, when G17E is set to a 1, IRQ0 at the UIC will always be forced to a zero. Note: Setting G17E to a 0 will not prevent GPIO channel 17 (if configured as an output) from creating contention with the off-chip source of the IRQ input. Therefore, be sure to leave the shared GPIO channel disabled when using the pin as an interrupt input.

13	G18E	GPIO 18 Enable 0 Enable interrupt IRQ1 as an interrupt 1 Enable interrupt IRQ1 as GPIO18
14	G19E	GPIO 19 Enable 0 Enable interrupt IRQ2 as an interrupt 1 Enable interrupt IRQ2 as GPIO19
15	G20E	GPIO 20 Enable 0 Enable interrupt IRQ3 as an interrupt 1 Enable interrupt IRQ3 as GPIO20
16	G21E	GPIO 21 Enable 0 Enable interrupt IRQ4 as an interrupt 1 Enable interrupt IRQ4 as GPIO21
17	G22E	GPIO 22 Enable 0 Enable interrupt IRQ5 as an interrupt 1 Enable interrupt IRQ5 as GPIO22
18	G23E	GPIO 23 Enable 0 Enable interrupt IRQ6 as an interrupt 1 Enable interrupt IRQ6 as GPIO23
19	DCS	DSR/CTS select 0 DSR is selected. 1 CTS is selected.
20	RDS	RTS/DTR select 0 RTS is selected. 1 DTR is selected.
21	DTE	DMA Transmit Enable for UART0 0 DMA transmit channel is disabled. 1 DMA transmit channel is enabled.
22	DRE	DMA Receive Enable for UART0 0 DMA receive channel is disabled. 1 DMA receive channel is enabled.
23	DAEC	DMA Allow Enable Clear for UART0 0 DTE and DRE for UART0 are not cleared when the UART receives a corresponding terminal count. 1 DTE and DRE for UART0 are cleared when the UART receives a corresponding terminal count.
24	U0EC	Select External Clock for UART0 0 UART0 uses the internally derived serial clock 1 UART0 uses the external serial clock input
25	U1EC	Select External Clock for UART1 0 UART1 uses the internally derived serial clock 1 UART1 uses the external serial clock input

CPC0_CR0 (cont.)

Chip Control Register 0

26:30	UDIV	UART Internal Clock Divisor 00000 Divide by 1 00001 Divide by 2 00010 Divide by 3 . . . 11110 Divide by 31 11111 Divide by 32	UDIV specifies the divisor of the CPU clock frequency used to derive a UART serial clock frequency. For example, if the CPU is running at 200MHz, a divider value of 20 sets the serial clock frequency to 10MHz. **Note:** The maximum serial clock frequency is less than 1/2 × OPB frequency
31		Reserved	

DCR 0x0B2

See “Chip Control Register 1 (CPC0_CR1)” on page 6-11.

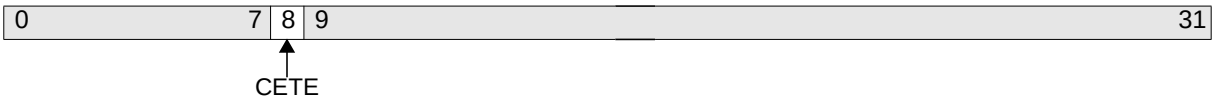


Figure 21-39. Chip Control Register 1 (CPC0_CR1)

0:7		Reserved
8	CETE	CPU External Timer Enable 0 CPU timers increment at CPU clock frequency. 1 CPU timers increment at frequency of TmrClk input.
9:31		Reserved

CPC0_ER

CPM Enable Register

DCR 0x0B9

See “CPM Enable Register (CPC0_ER)” on page 12-3.

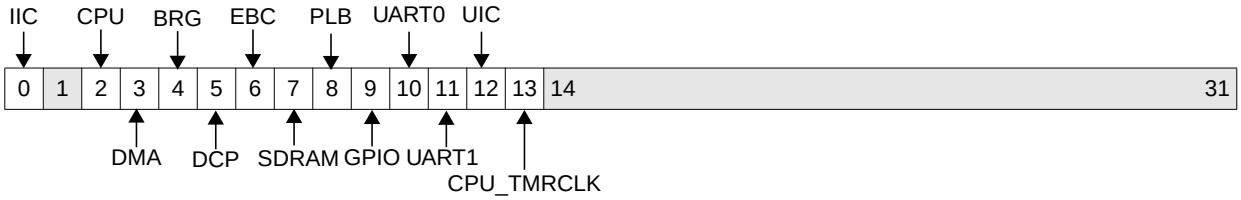


Figure 21-40. CPM Enable Register (CPC0_ER)

0	IIC	IIC Interface	Class 3
1		Reserved	
2	CPU	Processor Core	Class 2
3	DMA	DMA Controller	Class 2
4	BRG	PLB to OPB Bridge	Class 2
5	DCP	CodePack	Class 2
6	EBC	ROM/SRAM Peripheral Controller	Class 2
7	SDRAM	SDRAM Memory Controller	Class 2
8	PLB	PLB Bus Arbiter	Class 2
9	GPIO	General Purpose Interrupt Controller	Class 1
10	UART0	Serial Port 0	Class 1
11	UART1	Serial Port 1	Class 1
12	UIC	Universal Interrupt Controller	Class 1
13	CPU_TMRCLK	CPU Timers	Class 1
14:31		Reserved	

DCR 0x0BA

See “CPM Force Register (CPC0_FR)” on page 12-3.

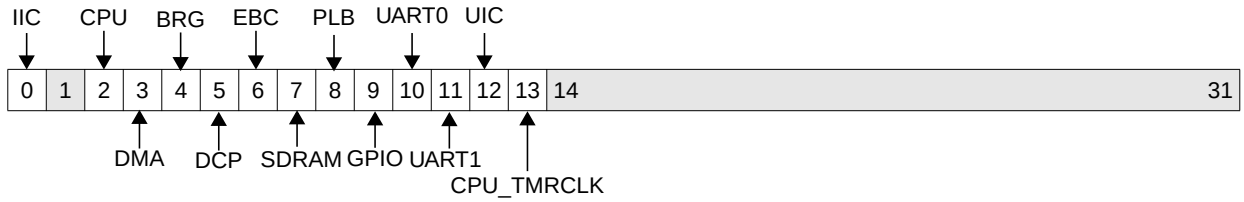


Figure 21-41. CPM Force Register (CPC0_FR)

0	IIC	IIC Interface	Class 3
1		Reserved	
2	CPU	Processor Core	Class 2
3	DMA	DMA Controller	Class 2
4	BRG	PLB to OPB Bridge	Class 2
5	DCP	CodePack	Class 2
6	EBC	ROM/SRAM Peripheral Controller	Class 2
7	SDRAM	SDRAM Memory Controller	Class 2
8	PLB	PLB Bus Arbiter	Class 2
9	GPIO	General Purpose Interrupt Controller	Class 1
10	UART0	Serial Port 0	Class 1
11	UART1	Serial Port 1	Class 1
12	UIC	Universal Interrupt Controller	Class 1
13	CPU_TMRCLK	CPU Timers	Class 1
14:31		Reserved	

CPC0_JTAGID

JTAG ID Register

DCR 0x0B5 Read-Only

See “JTAG ID Register (CPC0_JTAGID)” on page 11-4.

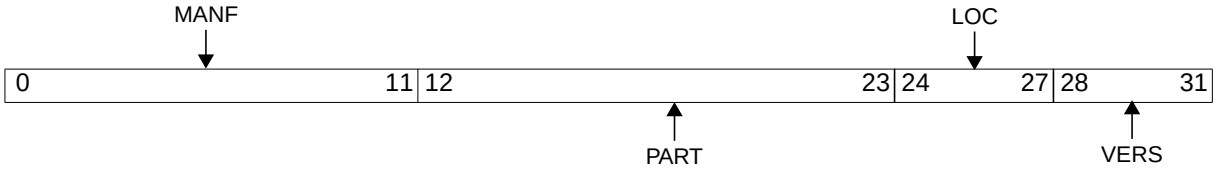


Figure 21-42. JTAG ID Register (CPC0_JTAGID)

0:11	MANF	Manufacturer Identifier
12:23	PART	Part Number
24:27	LOC	Developer Location
28:31	VERS	Version

DCR 0x0B

See “PLL Mode Register (CPC0_PLLMR)” on page 6-7.

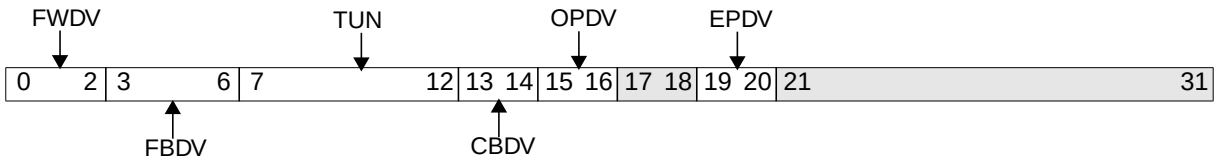


Figure 21-43. PLL Mode Register (CPC0_PLLMR)

0:2	FWDV	Forward Divisor 000 Forward divisor is 8. 001 Forward divisor is 7. 010 Forward divisor is 6. 011 Forward divisor is 5. 100 Forward divisor is 4. 101 Forward divisor is 3. 110 Forward divisor is 2. 111 Forward divisor is 1.	PLLOUTA Value: 50 MHz–100 MHz 58 MHz–114 MHz 66 MHz–134 MHz 80 MHz–160 MHz 100 MHz–200 MHz 133 MHz–267 MHz 200 MHz–400 MHz 400 MHz–800 MHz **Note:** PLLOUTA is the CPU clock.
3:6	FBDV	Feedback Divisor 0000 Feedback divisor is 16. 0001 Feedback divisor is 1. 0010 Feedback divisor is 2. 0011 Feedback divisor is 3. 0100 Feedback divisor is 4. 0101 Feedback divisor is 5. 0110 Feedback divisor is 6. 0111 Feedback divisor is 7. 1000 Feedback divisor is 8. 1001 Feedback divisor is 9. 1010 Feedback divisor is 10. 1011 Feedback divisor is 11. 1100 Feedback divisor is 12. 1101 Feedback divisor is 13. 1110 Feedback divisor is 14. 1111 Feedback divisor is 15.	
7:12	TUN	TUNE[5:0] Field	**Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs), visit the technical documents area of the IBM PowerPC web site.

CPC0_PLLMR (cont.)

PLL Mode Register

13:14	CBDV	CPU:PLB Frequency Divisor 00 CPU:PLB divisor is 1 01 CPU:PLB divisor is 2 10 CPU:PLB divisor is 3 11 CPU:PLB divisor is 4
15:16	OPDV	OPB–PLB Frequency Divisor 00 OPB–PLB divisor is 1 01 OPB–PLB divisor is 2 10 OPB–PLB divisor is 3 11 OPB–PLB divisor is 4
17:18		Reserved
19:20	EPDV	External Bus–PLB Frequency Divisor 00 External bus–PLB divisor ratio is 2:1 01 External bus–PLB divisor ratio is 3:1 10 External bus–PLB divisor ratio is 4:1 11 External bus–PLB divisor ratio is 5:1
21:31		Reserved

DCR 0x0B4

See “Chip Pin Strapping Register (CPC0_PSR)” on page 8-1.

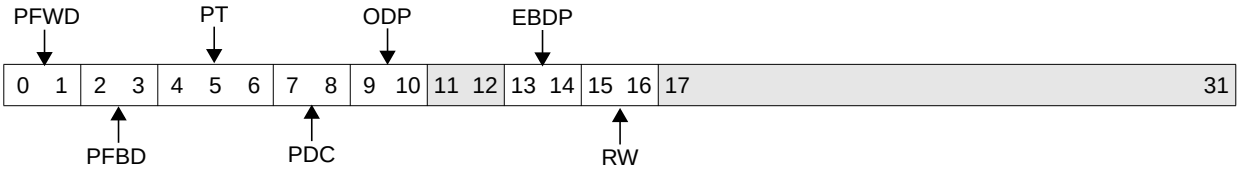


Figure 21-44. Chip Pin Strapping Register (CPC0_PSR)

0:1	PFWD	PLL Forward Divider 00 Bypass Mode 01 Divide by 3 10 Divide by 4 11 Divide by 6	
2:3	PFBD	PLL Feedback Divider 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4	
4:6	PT	PLL Tuning 000 Choice 1; TUNE[5:0] = 010001 001 Choice 2; TUNE[5:0] = 010010 010 Choice 3; TUNE[5:0] = 010011 011 Choice 4; TUNE[5:0] = 010100 100 Choice 5; TUNE[5:0] = 010101 101 Choice 6; TUNE[5:0] = 010110 110 Choice 7; TUNE[5:0] = 010111 111 Choice 8; TUNE[5:0] = 100100	**Note:** The tune bits adjust parameters that control PLL jitter. The recommended values minimize jitter for the PLL. These bits are shown for information only, and do not require modification except in special clocking circumstances, such as spread spectrum clocking. For details on the use of spread spectrum clock generators (SSCGs), visit the technical documents area of the IBM PowerPC web site.
7:8	PDC	PLB Divider from CPU 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4	
9:10	ODP	OPB Divider from PLB 00 Divide By 1 01 Divide By 2 10 Divide By 3 11 Divide By 4	
11:12		Reserved	
13:14	EBDP	External Bus Divider from PLB 00 Divide By 2 01 Divide By 3 10 Divide By 4 11 Divide By 5	

CPC0_PSR (cont.)

Chip Pin Strapping Register

15:16	RW	ROM Width 00 8-bit ROM 01 16-bit ROM 10 32-bit ROM 11 Reserved
17:31		Reserved

DCR 0x0B8 Read-Only

See “CPM Status Register (CPC0_SR)” on page 12-3.

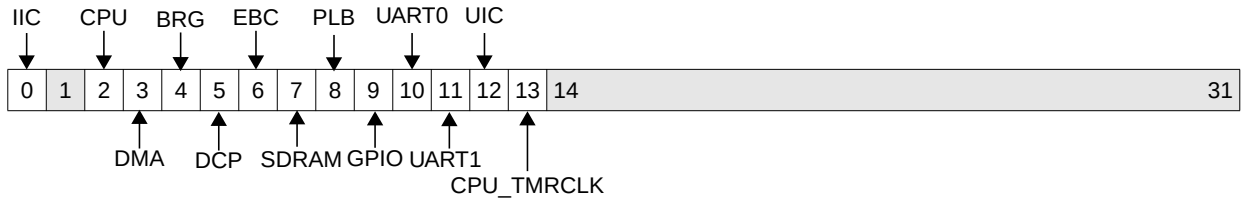


Figure 21-45. CPM Status Register (CPC0_SR)

0	IIC	IIC Interface	Class 3
1		Reserved	
2	CPU	Processor Core	Class 2
3	DMA	DMA Controller	Class 2
4	BRG	PLB to OPB Bridge	Class 2
5	DCP	CodePack	Class 2
6	EBC	ROM/SRAM Peripheral Controller	Class 2
7	SDRAM	SDRAM Memory Controller	Class 2
8	PLB	PLB Bus Arbiter	Class 2
9	GPIO	General Purpose Interrupt Controller	Class 1
10	UART0	Serial Port 0	Class 1
11	UART1	Serial Port 1	Class 1
12	UIC	Universal Interrupt Controller	Class 1
13	CPU_TMRCLK	CPU Timers	Class 1
14:31		Reserved	

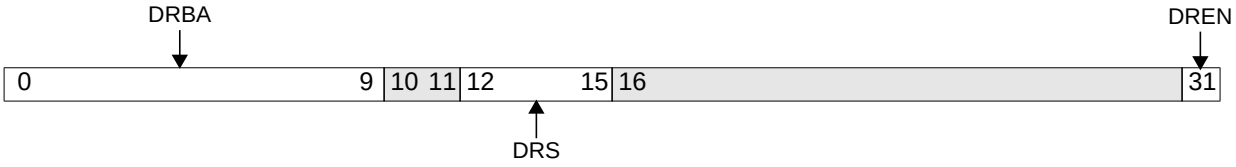
DCP0_ADDR0–DCP0_ADDR1

Address Decode Definition Register 0–1

Decompression Controller Registers

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x04–0x05

See “Decompression Address Decode Definition Registers (DCP0_ADDR0–DCP0_ADDR1)” on page 13-5.



Address Decode Definition Registers (DCP0_ADDR0–DCP0_ADDR1)

0:9	DRBA	Decode Region Base Address	High-order decode region address bits
10:11		Reserved	
12:15	DRS	Decode Region Size 0000–0100 Reserved 0101 4MB 0110 8MB 0111 16MB 1000 32MB 1001 64MB 1010 128MB 1011 256MB 1100 512MB 1101 1GB 1110 2GB 1111 4GB	
16:30		Reserved	
31	DREN	Enable Decode Region 0 Decoding is disabled for region 1 Decoding is enabled for region.	

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x40

See “Decompression Configuration Register (DCP0_CFG)” on page 13-5.

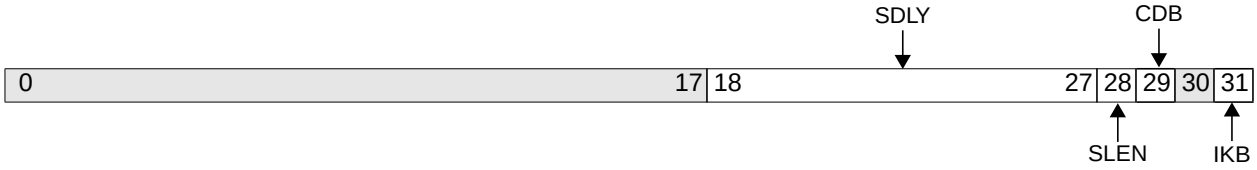


Figure 21-46. Decompression Controller Configuration Register (DCP0_CFG)

0:17		Reserved	
18:27	SLDY	Sleep Delay	Sleep delay
18:27	SLDY	Sleep Delay	Sleep delay
28	SLEN	Sleep Enable 0 Disable sleep 1 Enable sleep	
29	CDB	Clear Decompression Buffer 0 Normal operation 1 Clear decompression buffer	Self-clearing; always reads 0
30		Reserved	
31	IKB	Enable Decompression 0 Decompression is enabled; U0 storage attribute is recognized 1 Decompression is disabled; U0 storage attribute is ignored	

DCP0_CFGADDR

Decompression Controller Address Register

DCR 0x014

See “Decompression Controller Registers” on page 13-3.

This register is used to determine offsets for decompression controller DCRs.

DCR 0x015

See “Decompression Controller Registers” on page 13-3.

This register is used to indirectly access decompression controller DCRs.

DCP0_ESR

Decompression Controller Bus Error Status Register 0

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x52 Read/Clear

See “Decompression Controller Error Status Register 0 (DCP0_ESR)” on page 13-7.

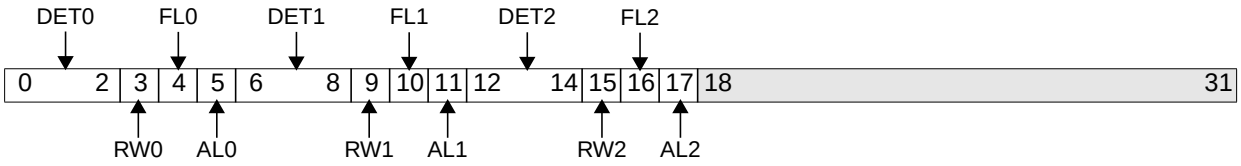


Figure 21-47. Decompression Controller Error Status Register 0 (DCP0_ESR)

0:2	DET0	Decompression Error Type for Master 0 000 Time-out during ITE fetch 001 Time-out during block fetch 010 Memory controller error during ITE fetch 011 Memory controller error during block fetch 100–111 Reserved	Master 0 is the processor core instruction cache unit (ICU).
3	RW0	Read/Write Status for Master 0 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
4	FL0	DCP0_ESR Field Lock for Master 0 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
5	AL0	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 0 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
6:8	DET1	Decompression Error Type for Master 1	See description for DCP0_ESR[DET0]. Master1 is the processor core data cache unit (DCU).
9	RW1	Read/Write Status for Master 1 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
10	FL1	DCP0_ESR Field Lock for Master 1 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
11	AL1	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 1 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.

12:14	DET2	Decompression Error Type for Master 2	See description for DCP0_ESR[DET0]. Master 2 is the external master.
15	RW2	Read/Write Status for Master 2 0 Error operation was a write 1 Error operation was a read	This implementation only reports errors for compressed reads.
16	FL2	DCP0_ESR Field Lock for Master 2 0 DCP0_ESR fields are unlocked 1 DCP0_ESR fields are locked	DCP0_ESR register fields are locked when the PLB_lockErr signal was active in the cycle in which the error occurred.
17	AL2	DCP0_MEMBEAR/DCP0_PLBBEAR Address Lock for Master 2 0 Master has not locked DCP0_MEMBEAR and DCP0_PLBBEAR 1 Master has locked DCP0_MEMBEAR and DCP0_PLBBEAR	DCP0_MEMBEAR and DCP0_PLBBEAR are locked to this master address when the PLB_lockErr signal was active in the cycle in which the error occurred.
18:31		Reserved	

DCP0_ID

Decompression Controller ID Register

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x41 Read-Only

See “Decompression Controller ID Register (DCP0_ID)” on page 13-6.

Figure 21-48. Decompression Controller ID Register (DCP0_ID)

0:31		Decompression Controller ID	Read-only, value is 0000504D

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x00–0x03

See “Index Table Origin Registers (DCP0_ITOR0–DCP0_ITOR3)” on page 13-4.

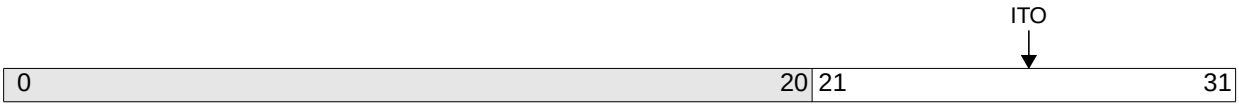


Figure 21-49. Decompression Index Table Origin Registers (DCP0_ITOR0–DCP0_ITOR3)

0:20		Reserved	
21:31	ITO	Index Table Origin	High-order index table address bits

DCP0_MEMBEAR

Decompression Controller Bus Error Address Register

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x51 Read-Only

See “Decompression Controller Bus Error Address Register (DCP0_MEMBEAR)” on page 13-7.

Figure 21-50. Decompression Controller Bus Error Address Register (DCP0_MEMBEAR)

0:31		Address of SDRAM or EBC Error

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x50 Read-Only

See “Decompression Controller PLB Error Address Register (DCP0_PLBBEAR)” on page 13-7.

Figure 21-51. Decompression Controller PLB Error Address Register (DCP0_PLBBEAR)

0:31		Address of PLB Error

DCP0_RAM0–DCP0_RAM3FF

Decompression Controller SRAM/ROM

DCR 0x400–0x7FF

See “Decompression Controller Registers” on page 13-3.

These registers store decode tables for the decompression controller.

DCR Accessed using DCP0_CFGADDR, DCP0_CFGDATA; Offset 0x42 Read-Only

See “Decompression Controller Version Register (DCP0_VER)” on page 13-6.

Figure 21-52. Decompression Controller Version Register (DCP0_VER)

0:31		Decompression ControllerVersion	Read-only, value is 0x00000200

DMA0_CR0–DMA0_CR3

DMA Channel Control Registers 0–3

DMA Registers

DCR 0x100, 0x108, 0x110, 0x118

See “DMA Channel Control Registers (DMA0_CR0–DMA0_CR3)” on page 16-8.

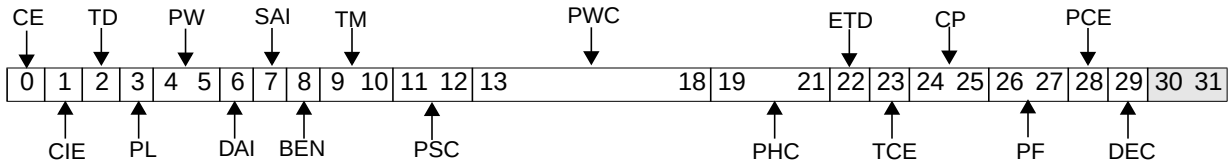


Figure 21-53. DMA Channel Control Registers (DMA0_CR0-DMA0_CR3)

0	CE	Channel Enable 0 Channel is disabled 1 Channel is enabled	This field is automatically cleared when the transfer completes or an error occurs.
1	CIE	Channel Interrupt Enable 0 Disable interrupts from this channel 1 Enable interrupts from this channel	When enabled, interrupts are generated for terminal count, end of transfer, and errors conditions. See “DMA Interrupts” on page 16-15.
2	TD	In peripheral mode: 0 Transfers are from memory-to-peripheral 1 Transfers are from peripheral-to-memory In device-paced memory-to-memory mode: 0 Peripheral is at the destination address 1 Peripheral is at the source address	TD is not used (don’t care) for software-initiated memory-to-memory transfers.
3	PL	Peripheral Location 0 External peripheral (EBC) bus 1 OPB (UART0)	
4:5	PW	Peripheral Width/Memory alignment 00 Byte (8 bits) 01 Halfword (16 bits) 10 Word (32 bits) 11 Doubleword (64 bits) memory-to-memory transfers only	Transfer width equals peripheral width for peripherals.
6	DAI	Destination Address Increment 0 Do not increment destination address 1 After each data transfer increment the destination address by: 1, if the transfer width is a byte (8 bits) 2, if the transfer width is a halfword (16 bits) 4, if the transfer width is a word (32 bits) 8, if the transfer width is a doubleword (64 bit)	
7	SAI	Source Address Increment 0 Do not increment source address 1 After each data transfer increment the source address by: 1, if the transfer width is a byte (8 bits) 2, if the transfer width is a halfword (16 bits) 4, if the transfer width is a word (32 bits) 8, if the transfer width is a doubleword (64 bit)	

8	BEN	Buffer Enable 0 Disable DMA 32-byte buffer 1 Enable DMA 32-byte buffer	If BEN=0 discrete read and write operations occur for each data transfer.
9:10	TM	Transfer mode 00 Peripheral 01 Reserved 10 Software-initiated memory-to-memory 11 Device-paced memory-to-memory	
11:12	PSC	Peripheral Setup Cycles 0-3	Number of PerClk cycles that the EBC peripheral bus is idle from the last peripheral bus transaction until DMAAckn is active. Used only for the peripheral side of peripheral mode transfers.
13:18	PWC	Peripheral Wait Cycles 0-63	DMAAckn remains active for PWC + 1 PerClk cycles. Used only for the peripheral side of peripheral mode transfers.
19:21	PHC	Peripheral Hold Cycles 0-7	The number of PerClk cycles from the time DMAAckn becomes inactive until the peripheral bus is available for the next bus access. Used only for the peripheral side of peripheral mode transfers.
22	ETD	End-of-Transfer/Terminal Count (EOTn[TCn]) Pin Direction 0 EOTn[TCn] is an EOT input 1 EOTn[TCn] is a TC output	ETD must be set to 1 if the channel is configured for software-initiated memory-to-memory transfers.
23	TCE	Terminal Count (TC) Enable 0 Channel does not stop when TC is reached 1 Channel stops when TC is reached	If TCE=1, it is required that ETD=1.
24:25	CP	Channel Priority 00 Low priority 01 Medium low priority 10 Medium high priority 11 High priority	Actively requesting channels of the same priority are prioritorized by channel number; channel 0 has the highest priority. See “Channel Priorities” on page 16-13 for more information.
26:27	PF	Memory Read Prefetch Transfer 00 Prefetch 1 doubleword 01 Prefetch 2 doublewords 10 Prefetch 4 doublewords 11 Reserved	Used only during memory-to-peripheral and deviced-paced memory-to-memory transfers. To enable prefetching it is required that BEN=1.
28	PCE	Parity Check Enable 0 Disable parity checking 1 Enable parity checking	Enables parity checking for peripheral mode transfers. See “Data Parity During DMA Peripheral Transfers” on page 16-13.
29	DEC	Address Decrement 0 SAI and DAI fields control memory address incrementing. 1 After each data transfer the memory address is decremented by the transfer width.	If DEC=1, it is required that BEN=0. This field is valid only for peripheral mode transfers (TM=00).
30:31		Reserved	

DMA0_CT0–DMA0_CT3

DMA Count Register 0–3

DCR 0x101, 0x109, 0x111, 0x119

See “DMA Count Registers (DMA0_CT0–DMA0_CT3)” on page 16-11.

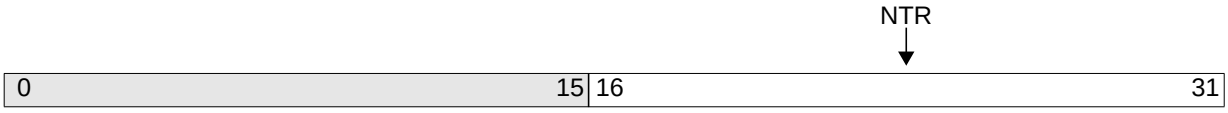


Figure 21-54. DMA Count Registers (DMA0_CT0-DMA0_CT3)

0:15		Reserved
16:31	NTR	Number of transfers remaining

DCR 0x102, 0x10A, 0x112, 0x11A

See “DMA Destination Address Registers (DMA0_DA0–DMA0_DA3)” on page 16-10.

Figure 21-55. DMA Destination Address Registers (DMA0_DA0-DMA0_DA3)

0:31		Destination address for memory-to-memory and peripheral-to-memory transfers.

DMA0_POL

DMA Polarity Configuration Register

DCR 0x126

See “DMA Polarity Configuration Register (DMA0_POL)” on page 16-5.

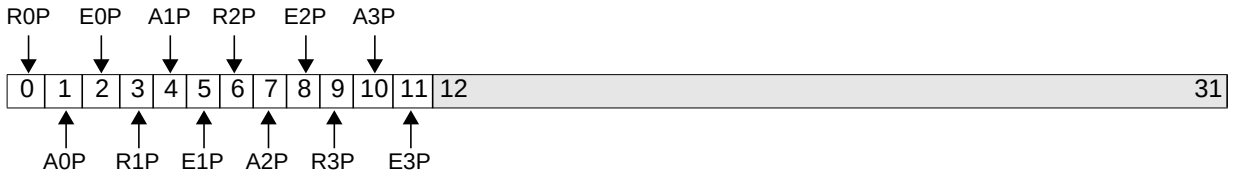


Figure 21-56. DMA Polarity Configuration Register (DMA0_POL)

0	R0P	DMAReq0 Polarity 0 DMAReq0 is active high 1 DMAReq0 is active low
1	A0P	DMAAck0 Polarity 0 DMAAck0 is active high 1 DMAAck0 is active low
2	E0P	EOT0[TC0] Polarity 0 EOT0[TC0] is active high 1 EOT0[TC0] is active low
3	R1P	DMAReq1 Polarity 0 DMAReq1 is active high 1 DMAReq1 is active low
4	A1P	DMAAck1 Polarity 0 DMAAck1 is active high 1 DMAAck1 is active low
5	E1P	EOT1[TC1] Polarity 0 EOT1[TC1] is active high 1 EOT1[TC1] is active low
6	R2P	DMAReq2 Polarity 0 DMAReq2 is active high 1 DMAReq2 is active low
7	A2P	DMAAck2 Polarity 0 DMAAck2 is active high 1 DMAAck2 is active low
8	E2P	EOT2[TC2] Polarity 0 EOT2[TC2] is active high 1 EOT2[TC2] is active low
9	R3P	DMAReq3 Polarity 0 DMAReq3 is active high 1 DMAReq3 is active low
10	A3P	DMAAck3 Polarity 0 DMAAck3 is active high 1 DMAAck3 is active low
11	E3P	EOT3[TC3] Polarity 0 EOT3[TC3] is active high 1 EOT3[TC3] is active low
12:31		Reserved

DCR 0x103, 0x10B, 0x113, 0x11B

See “DMA Source Address Registers (DMA0_SA0–DMA0_SA3)” on page 16-10.

Figure 21-57. DMA Source Address Registers (DMA0_SA0–DMA0_SA3)

0:31		Source address for memory-to-memory and memory-to-peripheral transfers.

DMA0_SG0–DMA0_SG3

DMA Scatter/Gather Descriptor Address Registers 0–3

DCR 0x104, 0x10C, 0x114, 0x11C

See “DMA Scatter/Gather Descriptor Address Registers (DMA0_SG0–DMA0_SG3)” on page 16-12.

Figure 21-58. DMA Scatter/Gather Descriptor Address Registers (DMA0_SG0–DMA0_SG3)

0:31		Address of next scatter/gather descriptor table.

DCR 0x123

See “DMA Scatter/Gather Command Register (DMA0_SGC)” on page 16-12.

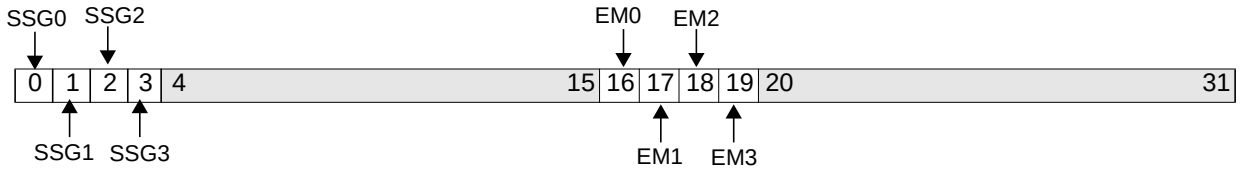


Figure 21-59. DMA Scatter/Gather Command Register (DMA0_SGC)

0:3	SSG[0:3]	Start Scatter/Gather for channels 0-3. 0 Scatter/gather support is disabled 1 Scatter/gather support is enabled	To start a scatter/gather operation for channel n, EM[n] must also be set.
4:15		Reserved	
16:19	EM[0:3]	Enable Mask for channels 0-3. 0 Writes to SSG[n] are ignored 1 Allow writing to SSG[n]	To write SSG[n], EM[n] must be set. Otherwise, writing SSG[n] has no effect.
20:31		Reserved	

DMA0_SLP

DMA Sleep Mode Register

DCR 0x125

See “DMA Sleep Mode Register (DMA0_SLP)” on page 16-6.

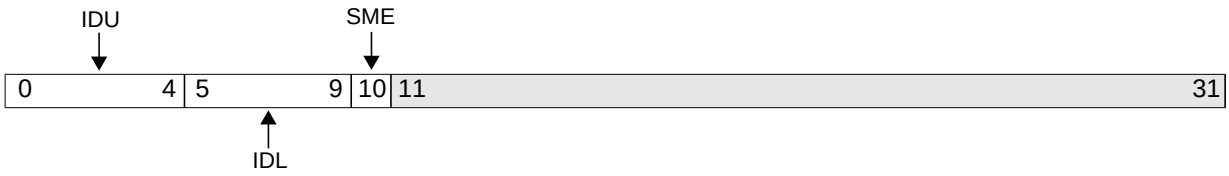


Figure 21-60. DMA Sleep Mode Register (DMA0_SLP)

0:4	IDU	Idle Timer Upper 0-31	Upper 5-bits of the idle timer.
5:9	IDL	Idle Timer Lower Hardcoded to 0b11111	Lower 5-bit portion of the idle timer. Writing this field has no effect.
10	SME	Sleep Mode Enable 0 Sleep disabled 1 Sleep enabled	If SME=1, also set CPM0_ER[DMA] to enable the Clock and Power Management macro to put the DMA controller to sleep.
11:31		Reserved	

DCR 0x120

See “DMA Scatter/Gather Command Register (DMA0_SGC)” on page 16-12.

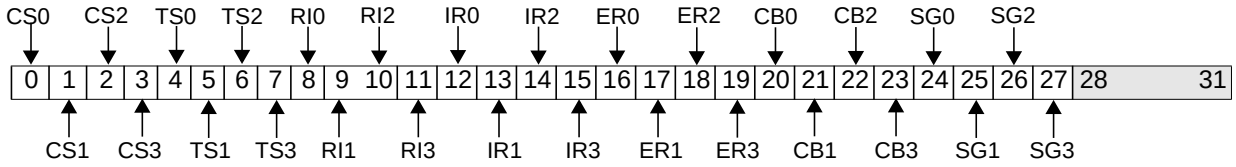


Figure 21-61. DMA Status Register (DMA0_SR)

0:3	CS[0:3]	Channel 0-3 Terminal Count Status 0 Terminal count has not occurred 1 Terminal count has been reached	Set when the transfer count reaches 0.
4:7	TS[0:3]	Channel 0-3 End of Transfer Status 0 End of transfer has not been requested 1 End of transfer has been requested	Only valid for channels with DMA0_CRn[ETD]=0.
8:11	RI[0:3]	Channel 0-3 Error Status 0 No error 1 Error occurred	See “Errors” on page 16-14 for more information.
12:15	IR[0:3]	Internal DMA Request 0 No internal DMA request pending 1 Internal DMA request is pending	
16:19	ER[0:3]	External DMA Request 0 No external DMA request pending 1 External DMA request is pending	
20:23	CB[0:3]	Channel Busy 0 Channel is idle 1 Channel currently active	
24:27	SG[0:3]	Scatter/Gather Status 0 No scatter/gather operation in progress 1 Scatter/gather operation in progress	
28:31		Reserved	

EBC0_BEAR

Peripheral Bus Error Address Register

EBC Registers

DCR Accessed using EBC0_CFGADDR; EBC0_CFGDATA; Offset 0x20

See “Peripheral Bus Error Address Register (EBC0_BEAR)” on page 15-29.

Figure 21-62. Peripheral Bus Error Address Register (EBC0_BEAR)

0:31		Address of Bus Error (asynchronous)

DCR Accessed using EBC0_CFGADDR; EBC0_CFGDATA; Offset 0x21

See “Peripheral Bus Error Status Register 0 (EBC0_BESR0)” on page 15-29.

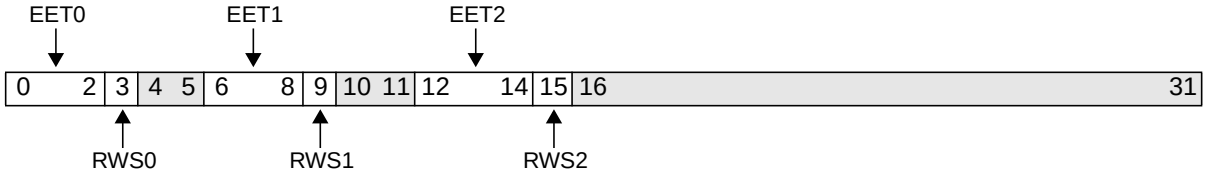


Figure 21-63. Peripheral Bus Error Status Register 0 (EBC0_BESR0)

0:2	EET0	Error type for master 0 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 0 is the processor instruction fetcher.
3	RWS0	Read/write status for master 0 0 Error operation was a write operation 1 Error operation was a read operation	
4:5		Reserved	
6:8	EET1	Error type for master 1 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 1 is the processor data side.
9	RWS1	Read/write status for master 1 0 Error operation was a write operation 1 Error operation was a read operation	
10:11		Reserved	
12:14	EET2	Error type for master 2 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error	Master 2 is the external bus master.
15	RWS2	Read/write status for master 2 0 Error operation was a write operation 1 Error operation was a read operation	
16:31		Reserved	

EBC0_BESR1

Peripheral Bus Error Status Register 1

DCR Accessed using EBC0_CFGADDR; EBC0_CFGDATA; Offset 0x22

See “Peripheral Bus Error Status Register 1 (EBC0_BESR1)” on page 15-30.

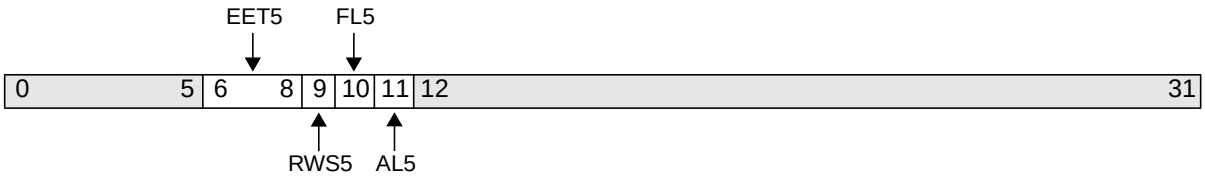


Figure 21-64. Peripheral Bus Error Status Register 1 (EBC0_BESR1)

0:5		Reserved
6:8	EET5	Error type for master 5 Master 5 is the DMA controller. 000 No error 001 Parity error 010 Reserved 011 Reserved 100 Protection error 101 Reserved 110 External bus input error 111 External bus timeout error
9	RWS5	Read/write status for master 5 0 Error operation was a write operation 1 Error operation was a read operation
10	FL5	Field lock for master 5 0 EET5 and RWS5 fields are unlocked 1 EET5 and RWS5 fields are locked
11	AL5	EBC0_BEAR address lock for master 5 0 EBC0_BEAR address unlocked 1 EBC0_BEAR address locked
12:31		Reserved

DCR Accessed using EBC0_CFGADDR; EBC0_CFGDATA; Offset 0x10–0x17

See “Peripheral Bank Access Parameters (EBC0_BnAP)” on page 15-26.

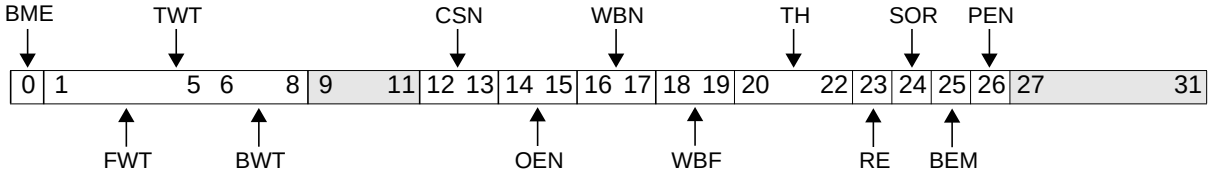


Figure 21-65. Peripheral Bank Access Parameters (EBC0_BnAP)

0	BME	Burst Mode Enable 0 Bursting is disabled 1 Bursting is enabled	
1:8	TWT	Transfer Wait 0-255 PerClk cycles	Wait states on all transfers when BME=0.
1:5	FWT	First Wait 0-31 PerClk cycles	If BME=1, number of wait states on the first transfer of a burst.
6:8	BWT	Burst Wait 0-7 PerClk cycles	If BME=1, number of wait states on non-first transfers of a burst.
9:11		Reserved	
12:13	CSN	Chip Select On Timing 0-3 PerClk cycles	Number of cycles from peripheral address driven to $\overline{\text{PerCSn}}$ low.
14:15	OEN	Output Enable On Timing 0-3 PerClk cycles	Number of cycles from $\overline{\text{PerCSn}}$ low to $\overline{\text{PerOE}}$ low.
16:17	WBN	Write Byte Enable On Timing 0-3 PerClk cycles	If BEM=0, number of cycles from $\overline{\text{PerCSn}}$ low to $\overline{\text{PerWBE0:3}}$ active.
18:19	WBF	Write Byte Enable Off Timing 0-3 PerClk cycles	If BEM=0 and RE=0, number of cycles $\overline{\text{PerWBEn}}$ becomes inactive prior to $\overline{\text{PerCSn}}$ inactive.
20:22	TH	Transfer Hold 0-7 PerClk cycles	Contains the number of hold cycles inserted at the end of a transfer.
23	RE	Ready Enable 0 PerReady is disabled 1 PerReady is enabled	
24	SOR	Sample on Ready 0 Data transfer occurs one PerClk cycle after PerReady is sampled active 1 Data transfer occurs in the same PerClk cycle that PerReady becomes active	
25	BEM	Byte Enable Mode 0 $\overline{\text{PerWBE0:3}}$ are only active for write cycles 1 $\overline{\text{PerWBE0:3}}$ are active for read and write cycles	If BEM=0, $\overline{\text{PerWBE0:3}}$ timing is controlled by WBN and WBF. If BEM=1, $\overline{\text{PerWBE0:3}}$ has the same timing as PerAddr0:31.
26	PEN	Parity Enable 0 Disable parity checking 1 Enable parity checking	The EBC implements odd parity.

EBC0_BnAP (cont.)

Peripheral Bank 0–7 Access Parameters

27:31		Reserved

DCR Accessed using EBC0_CFGADDR; EBC0_CFGDATA; Offset 0x00–0x07

See “Peripheral Bank Configuration Registers (EBC0_BnCR)” on page 15-25.

ROM presence and width are controlled by strapping pins in the PPC405CR.

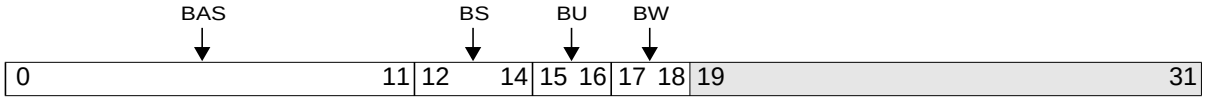


Figure 21-66. Peripheral Bank Configuration Registers (EBC0_BnCR)

0:11	BAS	Base Address Select	Specifies the bank starting address, which must be a multiple of the bank size.
12:14	BS	Bank Size 000 1 MB bank 001 2 MB bank 010 4 MB bank 011 8 MB bank 100 16 MB bank 101 32 MB bank 110 64 MB bank 111 128 MB bank	
15:16	BU	Bank Usage 00 Disabled 01 Read-only 10 Write-only 11 Read/Write	Specifies the type of accesses allowed for the bank. A protect error occurs if a write is attempted to a read-only bank or a read from a write-only bank.
17:18	BW	Bus Width 00 8-bit bus 01 16-bit bus 10 32-bit bus 11 Reserved	
19:31		Reserved	

EBC0_CFG

External Peripheral Control Register

DCR Accessed using EBC0_CFGADDR; EBC0_CFGDATA; Offset 0x23

See “EBC Configuration Register (EBC0_CFG)” on page 15-23.

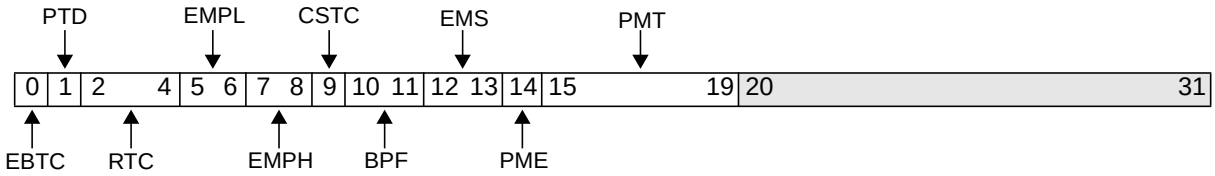


Figure 21-67. EBC Configuration Register (EBC0_CFG)

0	EBTC	External Bus Three-State Control 0 Address, data and control signals are high-Z between EBC transfers. 1 Between EBC transfers the peripheral data bus, address bus and control signals are driven.	Default after reset is EBTC=1. See “Effect of Driver Enable Programming on EBC Signal States” on page 15-5.
1	PTD	Device-Paced Time-out Disable 0 Enabled time-outs 1 Disable time-outs	If PTD=1, the EBC waits indefinitely for assertion of PerReady during device-paced accesses.
2:4	RTC	Ready Timeout Count 000 16 PerClk cycles 001 32 PerClk cycles 010 64 PerClk cycles 011 128 PerClk cycles 100 256 PerClk cycles 101 512 PerClk cycles 110 1024 PerClk cycles 111 2048 PerClk cycles	When PTD=0, the number of cycles from PerAddr0:31 changing until a time-out error occurs.
5:6	EMPL	External Master Priority Low 00 Low 01 Medium low 10 Medium high 11 High	The PLB priority for external master initiated transfers when the external master hold priority input is low (HoldPri=0).
7:8	EMPH	External Master Priority High 00 Low 01 Medium low 10 Medium high 11 High	The PLB priority for external master initiated transfers when the external master hold priority input is high (HoldPri=1).
9	CSTC	Chip Select Three-state Control 0 $\overline{\text{PerCS0:7}}$ are high-Z between EBC transfers and when an external master is active (HoldAck=1) 1 $\overline{\text{PerCS0:7}}$ are always driven.	Default after reset is CSTC=1. See “Effect of Driver Enable Programming on EBC Signal States” on page 15-5.
10:11	BPF	Burst Prefetch 00 Prefetch 1 doubleword 01 Prefetch 2 doublewords 10 Prefetch 4 doublewords 11 Reserved	Controls the amount of data prefetching when the EBC is servicing a PLB burst read. For most applications set this field to 0b00.

12:13	EMS	External Master Size 00 8-bit 01 16-bit 10 32-bit 11 No external master attached	Width of the attached external master.
14	PME	Power Management Enable 0 Disabled 1 Enabled	
15:19	PMT	Power Management Timer 0-31	The EBC makes a sleep request to the Clock and Power Management unit when PME=1 and the EBC has been idle for 32*PMT PerClk cycles.
20:31		Reserved	

EBC0_CFGADDR

Peripheral Controller Address Register

DCR 0x012

See “EBC Registers” on page 15-23.

This register is used to determine offsets for the indirectly-accessed EBC DCRs.

DCR 0x013

See “EBC Registers” on page 15-23.

This register is used to access the indirectly-accessed EBC DCRs.

GPIO0_IR

GPIO Input Register

GPIO Registers

MMIO 0xEF60071C Read-Only

See “GPIO Input Register (GPIO0_IR)” on page 19-7.

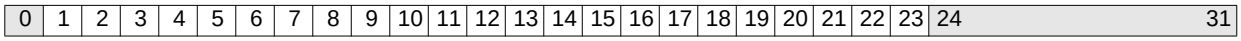


Figure 21-68. GPIO Input Register (GPIO0_IR)

0		Reserved
1:23		GPIO register bits
24:31		Reserved

MMIO 0xEF600718

See “GPIO Open Drain Register (GPIO0_ODR)” on page 19-6.

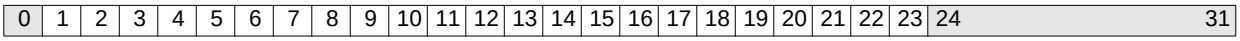


Figure 21-69. GPIO Open Drain Register (GPIO0_ODR)

0		Reserved
1:23		GPIO register bits
24:31		Reserved

GPIO0_OR

GPIO Output Register

MMIO 0xEF600700

See “GPIO Output Register (GPIO0_OR)” on page 19-6.

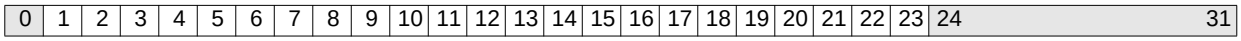


Figure 21-70. GPIO Output Register (GPIO0_OR)

0		Reserved
1:23		GPIO register bits
24:31		Reserved

MMIO 0xEF600704

See “GPIO Three-State Control Register (GPIO0_TCR)” on page 19-6.

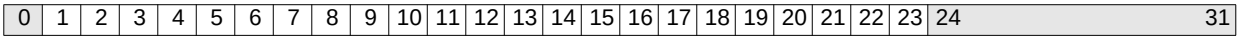


Figure 21-71. GPIO Three-State Register (GPIO0_TCR)

0		Reserved
1:23		GPIO register bits
24:31		Reserved

IIC0_CLKDIV

IIC0 Clock Divide

IIC Registers

MMIO 0xEF60050C

See “IIC0 Clock Divide Register” on page 18-15.

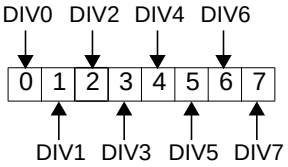


Figure 21-72. IIC0 Clock Divide Register (IIC0_CLKDIV)

0	DIV0	Divisor bit 0
1	DIV1	Divisor bit 1
2	DIV2	Divisor bit 2
3	DIV3	Divisor bit 3
4	DIV4	Divisor bit 4
5	DIV5	Divisor bit 5
6	DIV6	Divisor bit 6
7	DIV7	Divisor bit 7

MMIO 0xEF600506

See “IIC0 Control Register” on page 18-6.

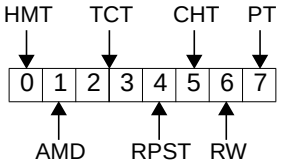


Figure 21-73. IIC0 Control Register (IIC0_CNTL)

0	HMT	Halt Master Transfer 0 Normal transfer operation. 1 Issue Stop signal on the IIC bus as soon as possible to halt master transfer.	If no transfer is in progress, no action is taken. IIC0_CNTL[PT] need not be set. If IIC0_MDCNTL[EINT] = 1, an interrupt is generated.
1	AMD	Addressing Mode 0 Use 7-bit addressing. 1 Use 10-bit addressing.	Does not affect slave transfers.
2:3	TCT	Transfer Count 00 Transfer one byte. 01 Transfer two bytes. 10 Transfer three bytes. 11 Transfer four bytes.	
4	RPST	Repeated Start 0 Normal start operation 1 Use repeated Start function to start transfer.	
5	CHT	Chain Transfer 0 Transfer is only or last transfer. 1 Transfer is one of a sequence of transfers (but not last in sequence).	Completion of a requested transfer causes a Stop signal to be issued on the IIC bus.
6	RW	Read/Write 0 Transfer is a write. 1 Transfer is a read.	
7	PT	Pending Transfer 0 Most recent requested transfer is complete. 1 Start transfer if bus is free.	

IIC0_DIRECTCNTL

IIC0 Direct Control

MMIO 0xEF600510

See “IIC0 Direct Control Register” on page 18-20.

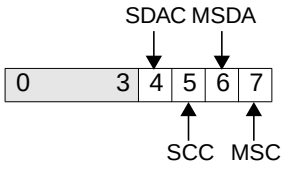


Figure 21-74. IIC0 Direct Control Register (IIC0_DIRECTCNTL)

0:3		Reserved	
4	SDAC	IICSDA Output Control Directly controls the IICSDA output. 0 IICSDA is a logic 0 1 IICSDA is a logic 1	
5	SCC	IICSCL Output Control Directly controls the IICSCL output 0 IICSCL is a logic 0 1 IICSCL is a logic 1	
6	MSDA	Monitor IICSDA Used to monitor the IICSDA input 0 IICSDA is a logic 0 1 IICSDA is a logic 1	Read-only
7	MSC	Monitor IICSCL. Used to monitor the IICSCL input. 0 IICSCL is a logic 0 1 IICSCL is a logic 1	Read-only

MMIO 0xEF600509

See “IIC0 Extended Control and Slave Status Register” on page 18-18.

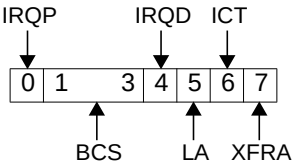


Figure 21-75. IIC0 Extended Status Register (IIC0_EXTSTS)

0	IRQP	IRQ Pending 0 No IRQ is pending. 1 An IRQ is active, another IRQ is on-deck, and another interrupt-generating condition has occurred.	• IIC0_EXTSTS[IRQP] might be set momentarily while an IRQ moves from the Pending to the On-deck state. • An interrupt remains pending, IIC0_EXTSTS[IRQP]=1, until the current on-deck interrupt becomes active, IIC0_EXTSTS[IRQD]=0 and IIC0_STS[IRQA]]=1. • Writing 1 to IIC0_EXTSTS[IRQP] clears the field. • When the IIC interrupt is disabled, IIC0_MDCNTL[IRQP] = 0, IIC0_EXTSTS[IRQP] should be ignored.
1:3	BCS	Bus Control State 000 Unused; if this value is read, a major IIC hardware problem occurred. 001 Slave-selected state; the IIC interface has detected and decoded a slave transfer request on the IIC bus. 010 Slave Transfer state; the IIC interface has detected but has not decoded a slave transfer request on the IIC bus. 011 Master Transfer state; entered after a master transfer request has started on the IIC bus. 100 Free Bus state; the bus is free and no transfer request is pending. 101 Busy Bus state; the bus is busy. 110 Unknown state; value after IIC reset. 111 Unused; if this value is read, a major IIC hardware problem occurred.	Read-only.

IIC0_EXTSTS (cont.)

IIC0 Extended Status

4	IRQD	IRQ On-Deck 0 No IRQ is on-deck. 1 An interrupt is active, and another interrupt-generating condition has occurred.	• IIC0_EXTSTS[IRQD] might be set momentarily while an IRQ moves from the On-deck to the Active state. • An interrupt remains on-deck, IIC0_EXTSTS[IRQD] = 1, until the current active interrupt is no longer active, IIC0_STS[IRQA] = 0. • If IIC0_EXTSTS[IRQP] = 1, IIC0_EXTSTS[IRQD] is set on the next OPB clock. • Writing 1 to IIC0_EXTSTS[IRQD] clears the field. • When the IIC interrupt is disabled, IIC0_EXTSTS[IRQP]=0, IIC0_EXTSTS[IRQD] should be ignored.
5	LA	Lost Arbitration 0 Normal operation. 1 Loss of arbitration has ended the requested master transfer.	• If arbitration is lost, any requested master transaction may have terminated prematurely. Read data may be incomplete and not all write data may have been written. • If arbitration is lost during a repeat start, the master may not own the IIC bus.
6	ICT	Incomplete Transfer 0 Normal operation. 1 Some of the bytes of the requested master transfer were not transferred.	For an incomplete transfer, read the transfer count, IIC0_XFRCNT, to determine how bytes were transferred.
7	XFRA	Transfer Aborted 0 No transfer is pending, or transfer is in progress. 1 A requested master transfer was aborted by a NACK during the transfer of the address byte, or was aborted because arbitration was lost. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.	Transfer aborted. When set to a 1, a requested master transfer was aborted by a NOT acknowledge during the transfer of the address byte. It is also set to a 1 when a requested master transfer loses data. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.

MMIO 0xEF600505

See “IIC0 High Master Address Register” on page 18-6.

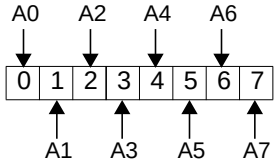


Figure 21-76. IIC0 High Master Address Register (IIC0_HMADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	MSb for 10-bit addresses
6	A6	Address bit 6	Next to MSb for 10-bit addresses
7	A7	Address bit 7	Don’t care for 10-bit addresses

IIC0_HSADR

IIC0 High Slave Address

MMIO 0xEF60050B

See “IIC0 High Slave Address Register” on page 18-14.

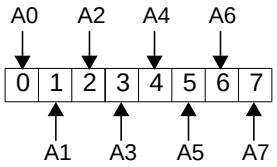


Figure 21-77. IIC0 High Slave Address Register (IIC0_HSADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	MSb for 10-bit addresses
6	A6	Address bit 6	Next to MSb for 10-bit addresses
7	A7	Address bit 7	Don’t care for 10-bit addresses

MMIO 0xEF60050D

See “IIC0 Interrupt Mask Register” on page 18-16.

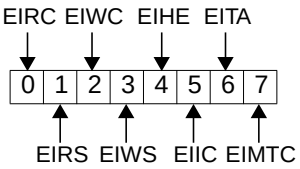


Figure 21-78. IIC0 Interrupt Mask Register (IIC0_INTRMSK)

0	EIRC	Enable IRQ on Slave Read Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave read on the IIC bus. IIC0_XTCNTLSS[SRC] = 1 indicates a Slave Read Complete.
1	EIRS	Enable IRQ on Slave Read Needs Service 0 Disable 1 Enable	The interrupt is activated upon receipt of a slave read on the IIC bus and the slave buffer was empty or went empty and more data was requested on the IIC bus. **Note:** IIC0_XTCNTLSS[SRS] = 1 indicates a Slave Read Needs Service.
2	EIWC	Enable IRQ on Slave Write Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave write on the IIC bus. **Note:** IIC0_XTCNTLSS[SWC] = 1 indicates a Slave Write Compete.
3	EIWS	Enable IRQ on Slave Write Needs Service 0 Disable 1 Enable	The interrupt is activated when the slave buffer becomes full during a slave write on the IIC bus. **Note:** IIC0_XTCNTLSS[SWS] = 1 indicates a Slave Write Needs Service.
4	EIHE	Enable IRQ on Halt Executed 0 Disable 1 Enable	
5	EIIC	Enable IRQ on Incomplete Transfer 0 Disable 1 Enable	
6	EITA	Enable IRQ on Transfer Aborted 0 Disable 1 Enable	
7	EIMTC	Enable IRQ on Requested Master Transfer Complete 0 Disable 1 Enable	

IIC0_LMADR

IIC0 Low Master Address

MMIO 0xEF600504

See “IIC0 Low Master Address Register” on page 18-5.

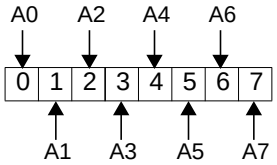


Figure 21-79. IIC0 Low Master Address Register (IIC0_LMADR)

0	A0	Address bit 0	
1	A1	Address bit 1	
2	A2	Address bit 2	
3	A3	Address bit 3	
4	A4	Address bit 4	
5	A5	Address bit 5	
6	A6	Address bit 6	LSb for 7-bit addresses
7	A7	Address bit 7	LSb for 10-bit addresses; don’t care for 7-bit addresses

MMIO 0xEF60050A

See “IIC0 Low Slave Address Register” on page 18-14.

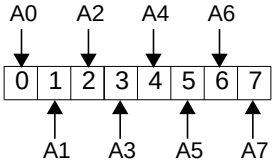


Figure 21-80. IIC0 Low Slave Address Register (IIC0_LSADR)

0	A0	Address bit 0	
1	A1	Address bit 1	
2	A2	Address bit 2	
3	A3	Address bit 3	
4	A4	Address bit 4	
5	A5	Address bit 5	
6	A6	Address bit 6	LSb for 7-bit addresses
7	A7	Address bit 7	LSb for 10-bit addresses; don’t care for 7-bit addresses

IIC0_MDBUF

IIC0 Master Data Buffer

MMIO 0xEF600500

See “IIC0 Master Data Buffer” on page 18-3.

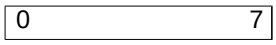


Figure 21-81. IIC0 Master Data Buffer (IIC0_MDBUF)

0		Data bit
1		Data bit
2		Data bit
3		Data bit
4		Data bit
5		Data bit
6		Data bit
7		Data bit

MMIO 0xEF600507

See “IIC0 Mode Control Register” on page 18-8.

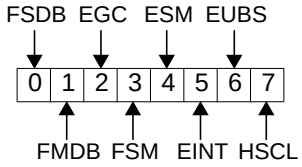


Figure 21-82. IIC0 Mode Control Register (IIC0_MDCNTL)

0	FSDB	Flush Slave Data Buffer 0 Normal operation 1 Set slave data buffer to empty.	Cleared after buffer is emptied.
1	FMDB	Flush Master Data Buffer 0 Normal operation 1 Set master data buffer to empty.	Cleared after buffer is emptied.
2	EGC	Enable General Call 0 Ignore general call on IIC bus. 1 Respond to general call on IIC bus.	IIC0_MDCNTL[ESM] overrides this field; if IIC0_MDCNTL[ESM] = 1, a general call is ignored.
3	FSM	Fast/Standard Mode 0 IIC transfers run at 100 kHz (standard mode). 1 IIC transfers run at 400 kHz (fast mode).	
4	ESM	Enable Slave Mode 0 Slave transfers are ignored. 1 Slave transfers are enabled.	Program IIC0_LSADR and IIC0_HSADR before setting this field.
5	EINT	Enable Interrupt 0 Interrupts are disabled. 1 Enables interrupts for interrupts enabled in IIC0_INTRMSK.	
6	EUBS	Exit Unknown IIC Bus State 0 Normal operation. 1 IIC bus control state machine exits unknown bus state, if in an unknown state.	If the IIC bus control state machine is in a known state, setting IIC0_MDCNTL[EUBS] = 1 has no effect.
7	HSCL	Hold IIC Serial Clock Low 0 If slave is not ready, issue a NACK in response to slave transfer request. 1 If slave is not ready, hold the IICSCL signal low until slave is ready.	This field is used only when in slave mode.

IIC0_SDBUF

IIC0 SLave Data Buffer

MMIO 0xEF600502

See “IIC0 Slave Data Buffer” on page 18-4.

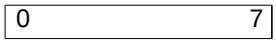


Figure 21-83. IIC0 Slave Data Buffer (IIC0_SDBUF)

0		Data bit
1		Data bit
2		Data bit
3		Data bit
4		Data bit
5		Data bit
6		Data bit
7		Data bit

MMIO 0xEF600508

See “IIC0 Status Register” on page 18-10.

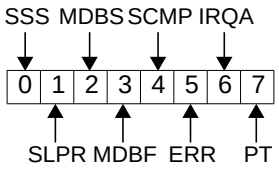


Figure 21-84. IIC0 Status Register (IIC0_STS)

0	SSS	Slave Status Set 0 No slave operations are in progress. 1 Slave operation is in progress.	Read-only; this field is set when any of the following fields are set: IIC0_XTCNTLSS[SRC, SRRS, SWC, SWRS].
1	SLPR	Sleep Request 0 Normal operation. 1 Sleep mode (CPC0_ER[IIC] = 1).	Read-only. The IIC interface is awakened when a start signal is detected on the IIC bus or when the CPC0_ER[IIC] is cleared.
2	MDBS	Master Data Buffer Status 0 Master data buffer is empty. 1 Master data buffer contains data.	Read-only.
3	MDBF	Master Data Buffer Full 0 Master data buffer is not full. 1 Master data buffer is full.	Read-only.
4	SCMP	Stop Complete 0 No request to halt transfer, or master data transfer, is complete. 1 Request to halt transfer, or master data transfer, is complete.	To clear IIC0_STS[SCMP], set IIC0_STS[SCMP] = 1.
5	ERR	Error 0 No error has occurred. 1 One of the following fields is set: IIC0_EXTSTS[LA, ICT, XFRA] = 1.	Read-only.
6	IRQA	IRQ Active 0 No IIC interrupt has been sent to the universal interrupt controller (UIC). 1 An IIC interrupt has been sent to the UIC.	To clear IIC0_STS[IRQA], set IIC0_STS[IRQA] = 1. If IIC0_MDCNTL[EINT] = 0, then IIC0_STS[IRQA] is not set.
7	PT	Pending Transfer 0 No transfer is pending, or transfer is in progress. 1 Transfer is pending.	Read-only.

IIC0_XFRCNT

IIC0 Transfer Count

MMIO 0xEF60050E

See “IIC0 Transfer Count Register” on page 18-17.

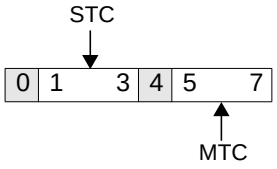


Figure 21-85. IIC0 Transfer Count Register (IIC0_XFRCNT)

0		Reserved
1:3	STC	Slave Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved
4		Reserved
5:7	MTC	Master Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved

MMIO 0xEF60050F

See “IIC0 Extended Control and Slave Status Register” on page 18-18.

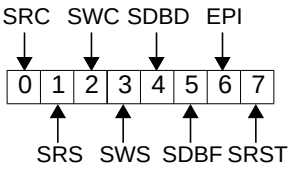


Figure 21-86. IIC0 Extended Control and Slave Status Register (IIC0_XTCNTLSS)

0	SRC	Slave Read Complete 0 Normal operation, or IIC0_MDCNTL[HSCL] = 0, IIC0_SDBUF is empty, and a read operation is in progress. 1 A NACK or Stop condition was received over the IIC bus, or a repeated Start condition ended a read operation.	Check whether the read operation emptied IIC0_SDBUF.
1	SRS	Slave Read Needs Service 0 Normal operation or slave read does not need service. 1 IIC0_SDBUF is empty, and a read operation was requested on the IIC bus. The set condition may also indicate that IIC0_SDBUF is empty due to a slave read and additional data is requested by the master.	1. If IIC0_MDCNTL[HSCL]=0 and IIC0_SDBUF contains no data, the slave will issue a NACK and IIC0_XTCNTLSS[SRS] is set. 2. If IIC0_MDCNTL[HSCL]=0, and IIC0_SDBUF contains data, the slave will send the data. IIC0_XTCNTLSS[SRS] is not set unless the master request additional data. 3. If IIC0_MDCNTL[HSCL]=0, and IIC0_SDBUF contains no data, the slave will hold IICSCL low to indicate the slave is busy. IIC0_XTCNTLSS[SRS] is set until the IIC0_SDBUF is filled. Once filled, IICSCL is released, IIC0_XTCNTLSS[SRS] is cleared, and the slave sends the data. 4. If IIC0_MDCNTL[HSCL]=1, and IIC0_SDBUF contains data, the slave will send the data. IIC0_XTCNTLSS[SRS] is not set unless the master requests additional data.
2	SWC	Slave Write Complete 0 Normal operation or slave write in progress. 1 A Stop signal was received during a write operation, or a repeated Start condition ended a write operation.	

IIC0_XTCNTLSS (cont.)

IIC0 Extended Control and Slave Status

3	SWS	Slave Write Needs Service 0 Normal operation or slave write does not need service. 1 IIC0_SDBUF is full during a slave write.	1. If IIC0_MDCNTL[HSCL] = 1 and IIC0_SDBUF is full, the slave will hold IICSCL low to indicate the slave is busy. IIC0_XTCNTLSS[SWS] is set until IIC0_SDBUF is empty. Once empty, IICSCL is released, IIC0_XTCNTLSS[SWS] is cleared, and the slave receives the data. 2. If IIC0_MDCNTL[HSCL] = 0 and IIC0_SDBUF is full, the slave will issue a NACK and IIC0_XTCNTLSS[SWS] is set.
4	SDBD	Slave Data Buffer Has Data 0 IIC0_SDBUF is empty 1 IIC0_SDBUF contains data	Read-only
5	SDBF	Slave Data Buffer Full 0 IIC0_SDBUF is not full 1 IIC0_SDBUF is full	Read-only
6	EPI	Enable Pulsed IRQ on Transfer Aborted 0 The internal IIC interrupt signal to the UIC remains active until the status is cleared, IIC0_STS[IRQA] =0. 1 The internal IIC interrupt signal to the PPC405CR UIC is active for one OPB clock cycle.	
7	SRST	Soft Reset 0 Normal operation 1 Soft reset	

OPB Arbiter Registers

MMIO 0xEF60 0600

See “OPB Arbiter Control Register (OPBA0_CR)” on page 2-11.

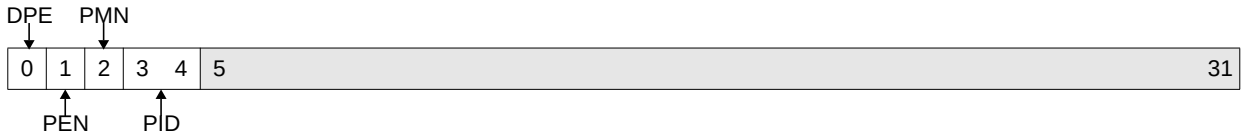


Figure 21-87. OPB Arbiter Control Register (OPBA0_CR)

0	DPE	Dynamic Priority Enable 0 Dynamic priority disabled 1 Dynamic priority enabled	
1	PEN	Park Enable 0 Park disabled 1 Park enabled	
2	PMN	Park on Master Not Last 0 Park on master last 1 Park on master not last	
3:4	PID	Parked Master ID 00 Master ID 0 01 Reserved 10 Master ID 2 11 Reserved	Master 0 is DMA; master 2 is the OPB to PLB bridge.
5:31		Reserved	

OPBA0_PR

OPB Arbiter Priority Register

MMIO 0xEF600601

See “OPB Arbiter Priority Register (OPBA0_PR)” on page 2-12.

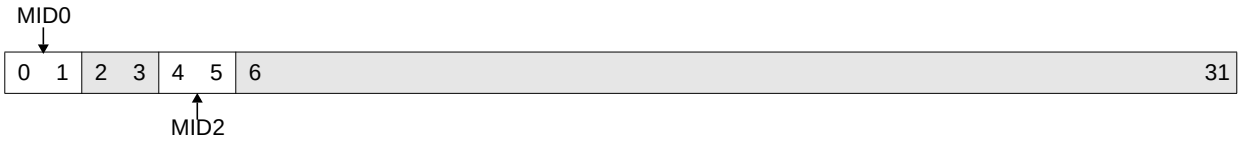


Figure 21-88. OPB Arbiter Priority Register (OPBA0_PR)

0:1	MID0	High Priority Master ID 00 Master ID 0 01 Reserved 10 Reserved 11 Reserved	At reset, this priority is assigned to DMA.
2:3		Reserved	
4:5	MID2	Low Priority master ID 00 Reserved 01 Reserved 10 Master ID 2 11 Reserved	At reset, this priority is assigned to the OPB to PLB bridge.
6:31		Reserved	

PLB Registers

DCR 0x087

See “PLB Arbiter Control Register (PLB0_ACR)” on page 2-5.

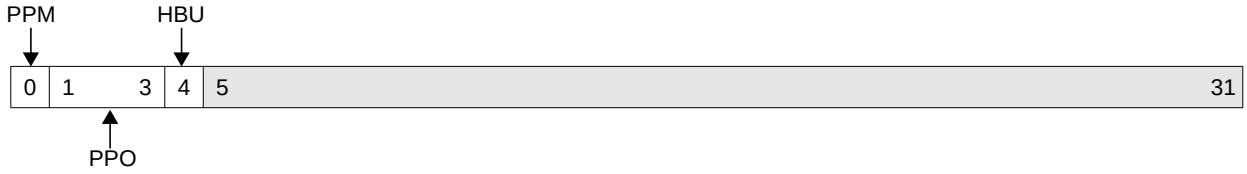


Figure 21-89. PLB Arbiter Control Register (PLB0_ACR)

0	PPM	PLB Priority Mode 0 Fixed 1 Fair
1:3	PPO	PLB Priority Order 000 Masters 0, 1, 2, 5 001 Masters 1, 2, 5, 0 010 Masters 2, 5, 0, 1 011 Masters 5,0, 1, 2 100 Reserved 101 Reserved 110 Reserved 111 Reserved
4	HBU	High Bus Utilization 0 Disabled 1 Enabled
5:31		Reserved

PLB0_BEAR

PLB Error Address Register

DCR 0x086

See “PLB Error Address Register (PLB0_BEAR)” on page 2-6.

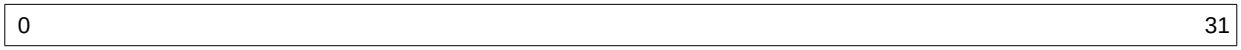


Figure 21-90. PLB Error Address Register (PLB0_BEAR)

0:31		Address of bus timeout error

DCR 0x084 Read/Clear

See “PLB Error Status Register (PLB0_BESR)” on page 2-6.

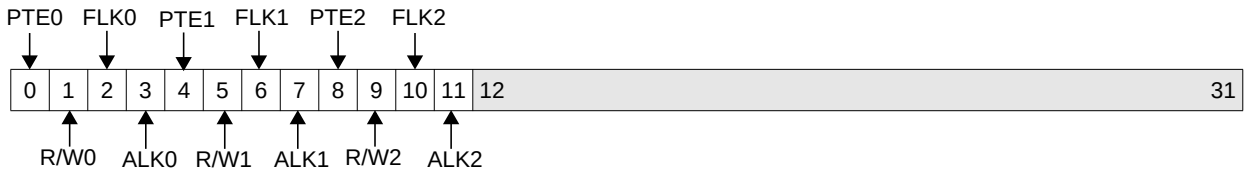


Figure 21-91. PLB Error Status Register (PLB0_BESR)

0	PTE0	Master 0 PLB Timeout Error Status 0 No master 0 timeout error 1 Master 0 timeout error	Master 0 is the processor core ICU.
1	R/W0	Master 0 Read/Write Status 0 Master 0 error operation was a write 1 Master 0 ICU error operation was a read	
2	FLK0	Master 0 PLB0_BESR Field Lock 0 Master 0 PLB0_BESR field is unlocked 1 Master 0 field is locked	
3	ALK0	Master 0 PLB0_BEAR Address Lock 0 Master 0 PLB0_BEAR is unlocked 1 Master 0 PLB0_BEAR is locked	
4	PTE1	Master 1 PLB Timeout Error Status 0 No master 1 timeout error 1 Master 1 timeout error	Master 1 is the processor core DCU.
5	R/W1	Master 1 Read/Write Status 0 Master 1 error operation was a write 1 Master 1 error operation was a read	
6	FLK1	Master 1PLB0_BESR Field Lock 0 Master 1 PLB0_BESR field is unlocked 1 Master 1 PLB0_BESR field is locked	
7	ALK1	Master 1 PLB0_BEAR Address Lock 0 Master 1 PLB0_BEAR is unlocked 1 Master 1 PLB0_BEAR is locked	
8	PTE2	Master 2 PLB Timeout Error Status 0 No master 2 timeout error 1 Master 2 timeout error	Master 2 is the external master.
9	R/W2	Master 2 Read/Write Status 0 Master 2 error operation was a write 1 Master 2 error operation was a read	
10	FLK2	Master 2 PLB0_BESR Field Lock 0 Master 2 PLB0_BESR field is unlocked 1 Master 2 PLB0_BESR field is locked	
11	AL2	Master 2 PLB0_BEAR Address Lock 0 Master 2 PLB0_BEAR is unlocked 1 Master 2 PLB0_BEAR is locked	
12:31		Reserved	

PLB0_BESR (cont.)

PLB Error Status Register

PLB-OPB Bridge Registers

DCR 0x0A2 Read-Only

See “Bridge Error Address Register (POB0_BEAR)” on page 2-8.

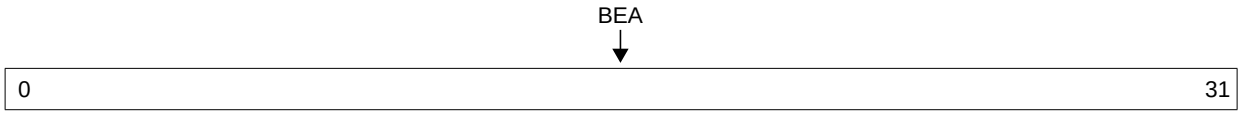


Figure 21-92. Bridge Error Address Register (POB0_BEAR)

0:31	BEA	Address of bus error

POB0_BESR0

Bridge Error Status Register 0

DCR 0x0A0 Read/Clear

See “Bridge Error Status Registers (POB0_BESR0–POB0_BESR1)” on page 2-8.

Figure 21-93. Bridge Error Status Register 0 (POB0_BESR0)

0:1	PTE0	PLB Timeout Error Status Master 0 00 No master 0 error occurred 01 Master 0 timeout error occurred 10 Master 0 slave error occurred 11 Reserved	Master 0 is the processor core ICU.
2	R/W0	Read Write Status Master 0 0 Master 0 error operation is a read 1 Master 0 error operation is a write	
3	FLK0	POB0_BESR0 Field Lock Master 0 0 Master 0 POB0_BESR0 field is unlocked 1 Master 0 POB0_BESR0 field is locked	
4	ALK0	POB0_BEAR Address Lock Master 0 0 Master 0 POB0_BEAR address is unlocked 1 Master 0 POB0_BEAR address is locked	
5:6	PTE1	PLB Timeout Error Status Master 1 00 No master 1 error occurred 01 Master 1 timeout error occurred 10 Master 1 slave error occurred 11 Reserved	Master 1 is the processor core DCU.
7	R/W1	Read/Write Status Master 1 0 Master 1 error operation is a read 1 Master 1 error operation is a write	
8	FLK1	POB0_BESR0 Field Lock Master 1 0 Master 1 POB0_BESR0 field is unlocked 1 Master 1 POB0_BESR0 field is locked	
9	ALK1	POB0_BEAR Address Lock Master 1 0 Master 1 POB0_BEAR address is unlocked 1 Master 1 POB0_BEAR address is locked	
10:11	PTE2	PLB Timeout Error Status Master 2 00 No master 2 error occurred 01 Master 2 timeout error occurred 10 Master 2 slave error occurred 11 Reserved	Master 2 is the external master.
12	R/W2	Read/Write Status Master 2 0 Master 2 error operation is a read 1 Master 2 error operation is a write	

13	FLK2	POB0_BESR0 Field Lock Master 2 0 Master 2 POB0_BESR0 field is unlocked 1 Master 2 POB0_BESR0 field is locked
14	ALK2	POB0_BEAR Address Lock Master 2 0 Master 2 POB0_BEAR address is unlocked 1 Master 2 POB0_BEAR address is locked
15:31		Reserved

POB0_BESR1

Bridge Error Status Register 1

DCR 0x0A4 Read/Clear

See “Bridge Error Status Registers (POB0_BESR0–POB0_BESR1)” on page 2-8.

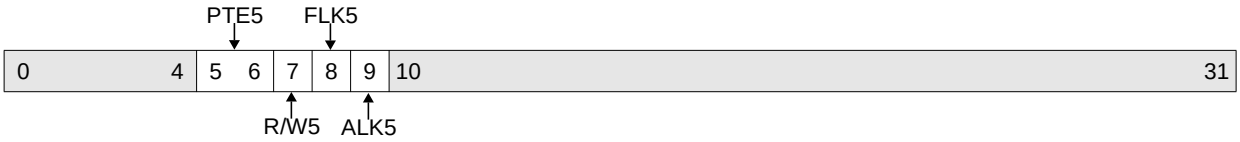


Figure 21-94. Bridge Error Status Register 1 (POB0_BESR1)

0:4		Reserved
5:6	PTE5	PLB Timeout Error Status Master 5 Master 5 is DMA. 00 No Master 5 error occurred 01 Master 5 timeout error occurred 10 Master 5 slave error occurred 11 Reserved
7	R/W5	Read/Write Status Master 5 0 Master 5 error operation is a read 1 Master 5 error operation is a write
8	FLK5	POB0_BESR1 Field Lock Master 5 0 Master 5 POB0_BESR1 field is unlocked 1 Master 5 POB0_BESR1 field is locked
9	ALK5	POB0_BEAR Address Lock Master 5 0 Master 5 POB0_BEAR address is unlocked 1 Master 5 POB0_BEAR address is locked
10:31		Reserved

SDRAM Registers

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x40–0x4C

See “Memory Bank 0–3 Configuration (SDRAM0_B0CR–SDRAM0_B3CR)” on page 14-5.

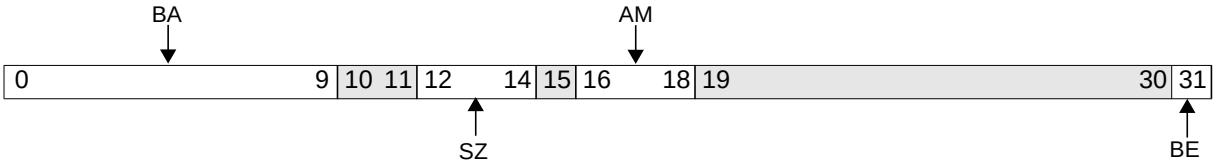


Figure 21-95. Memory Bank 0–3 Configuration Registers (SDRAM0_B0CR–SDRAM0_B3CR)

0:9	BA	Base Address	The base address must be aligned on a boundary that matches the size of the region as defined by the SZ field. For example, a 4MB region must begin on an address that is divisible by 4MB.
10:11		Reserved	
12:14	SZ	Size 000 4M byte 001 8M byte 010 16M byte 011 32M byte 100 64M byte 101 128M byte 110 256M byte 111 Reserved	
15		Reserved	
16:18	AM	Addressing Mode 000 Mode 1 001 Mode 2 010 Mode 3 011 Mode 4 100 Mode 5 101 Mode 6 110 Mode 7 111 Reserved	See “SDRAM Addressing Modes” on page 14-7.
19:30		Reserved	
31	BE	Memory Bank Enable 0 Bank is disabled 1 Bank is enabled	

SDRAM0_BEAR

PLB Master Bus Error Address Register

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x10

See “Bus Error Address Register (SDRAM0_BEAR)” on page 14-16.

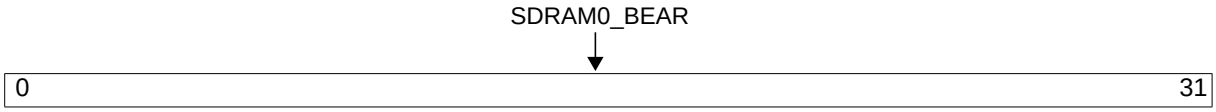


Figure 21-96. Bus Error Address Register (SDRAM0_BEAR)

0:31	SDRAM0_BEAR	Address of ECC Error.

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x00 Read/Clear

See “Bus Error Syndrome Register 0 (SDRAM0_BESR0)” on page 14-17.

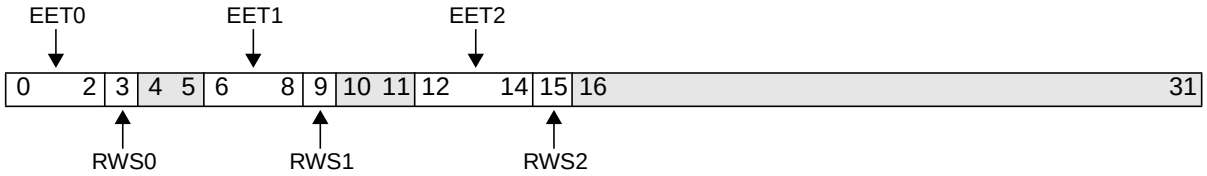


Figure 21-97. Bus Error Syndrome Register 0 (SDRAM0_BESR0)

0:2	EET0	Error type for master 0 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 0 is the processor instruction fetcher.
3	RWS0	Read/write status for master 0 0 Error operation was a write operation 1 Error operation was a read operation	
4:5		Reserved	
6:8	EET1	Error type for master 1 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 1 is the processor data side.
9	RWS1	Read/write status for master 1 0 Error operation was a write operation 1 Error operation was a read operation	
10:11		Reserved	
12:14	EET2	Error type for master 2 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 2 is the external bus master.
15	RWS2	Read/write status for master 2 0 Error operation was a write operation 1 Error operation was a read operation	
16:31		Reserved	

SDRAM0_BESR1

Bus Error Syndrome Register B

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x08 Read/Clear

See “Bus Error Syndrome Register 1 (SDRAM0_BESR1)” on page 14-17.

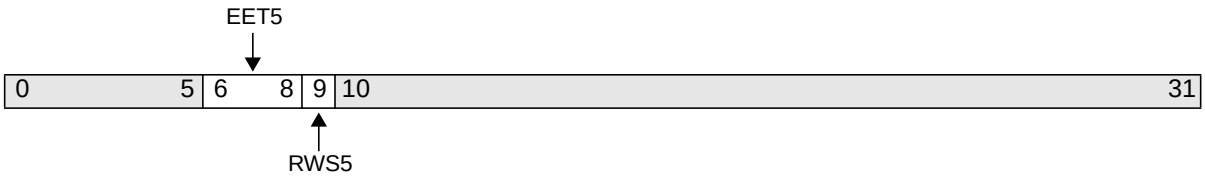


Figure 21-98. Bus Error Status Register 1 (SDRAM0_BESR1)

0:5		Reserved	
6:8	EET5	Error type for master 5 000 No error 001 Reserved 01X ECC uncorrectable error 1XX Reserved	Master 5 is the DMA controller.
9	RWS5	Read/write status for master 5 0 Error operation was a write operation 1 Error operation was a read operation	
10:31		Reserved	

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x20

See “Memory Controller Configuration Register (SDRAM0_CFG)” on page 14-3.

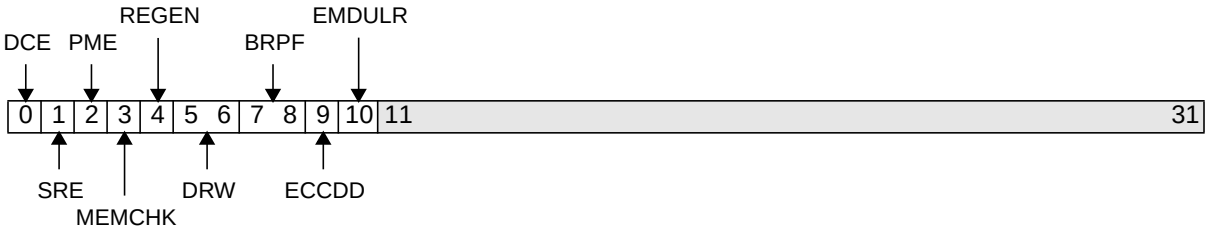


Figure 21-99. Memory Controller Configuration (SDRAM0_CFG)

0	DCE	SDRAM Controller Enable 0 Disable 1 Enable	All SDRAM controller configuration registers must be initialized and valid prior to setting DCE.
1	SRE	Self-Refresh Enable 0 Disable 1 Enable	See “Self-Refresh” on page 14-18.
2	PME	Power Management Enable 0 Disable 1 Enabled	See “Power Management” on page 14-19.
3	MEMCHK	Memory Data Error Checking 0 None 1 ECC	See “Error Checking and Correction (ECC)” on page 14-14.
4	REGEN	Registered Memory Enable 0 Disabled 1 Enabled	
5:6	DRW	SDRAM Width 00 32-bit 01 Reserved 10 Reserved 11 Reserved	Must be set to 0b00.
7:8	BRPF	Burst Read Prefetch Granularity 00 Reserved 01 16 bytes 10 32 bytes 11 Reserved	Most applications should set with field to 0b01.
9	ECCDD	ECC Driver Disable 0 Check bit data output on ECC0:7. 1 ECC0:7 are placed in high-Z state.	Regardless of whether ECC checking is enabled, SDRAM writes cause the chip to write check bit data on ECC0:7. If ECC is unused, setting ECCDD = 1 disables these drivers.
10	EMDULR	Enable Memory Data Unless Read 0 MemData0:31 are high-Z unless a memory write is being performed. 1 MemData0:31 are driven unless a memory read is being performed.	
11:31		Reserved	

SDRAM0_CFGADDR

Memory Controller Address Register

DCR 0x010

See “Accessing SDRAM Registers” on page 14-2.

This register is used to determine offsets for SDRAM controller DCRs.

DCR 0x011

See “Accessing SDRAM Registers” on page 14-2.

This register is used to indirectly access SDRAM controller DCRs.

SDRAM0_ECCCFG

ECC Configuration Register

Offset 0x94 DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x94

See “ECC Configuration Register (SDRAM0_ECCCFG)” on page 14-14.

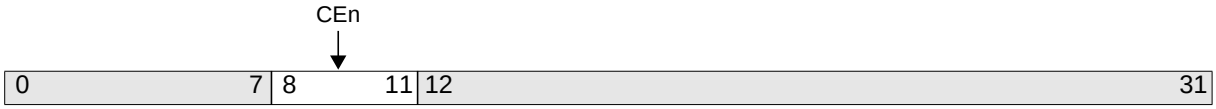


Figure 21-100. ECC Configuration Register (SDRAM0_ECCCFG)

0:7		Reserved	
8:11	CEn	ECC Correction Enable for bank n. 0 Disabled 1 Enabled	When CEn is set, ECC correction is enabled for bank n ($\overline{\text{BankSeln}}$). When cleared, the ECC logic ignores the check bits and passes read data along unmodified.
12:31		Reserved	

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x98 Read-Only

See “ECC Error Status Register (SDRAM0_ECCESR)” on page 14-16.

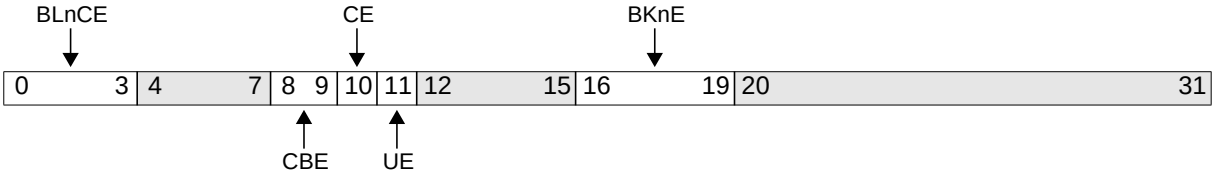


Figure 21-101. ECC Error Status Register (SDRAM0_ECCESR)

0:3	BLnCE	Byte Lane n Corrected Error 0 No error 1 Error occurred in byte lane n
4:7		Reserved
8:9	CBE	Error Detected in Check bits 00 No error 01 Error in lower check bits 10 Error in upper check bits 11 Error in both sets of check bits
10	CE	Correctable Error
11	UE	Uncorrectable Error
12:15		Reserved
16:19	BKnE	Bank n Error 0 No error 1 Error occurred in bank n
20:31		Reserved

SDRAM0_PMIT

Power Management Idle Timer

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x34

See “Power Management Idle Timer (SDRAM0_PMIT)” on page 14-19.

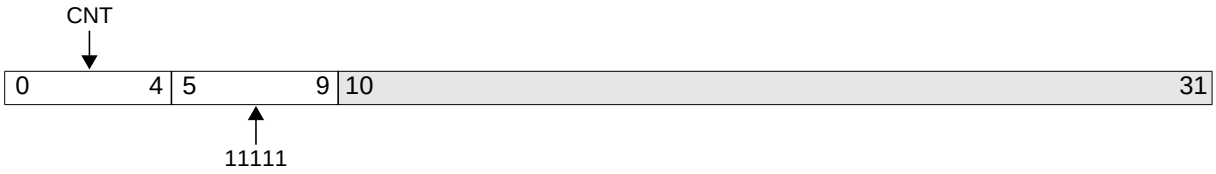


Figure 21-102. Power Management Idle Timer (SDRAM0_PMIT)

0:4	CNT	Count 0-31
5:9		Always 0b11111
10:31		Reserved

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x30

See “Refresh Timer Register (SDRAM0_RTR)” on page 14-13.

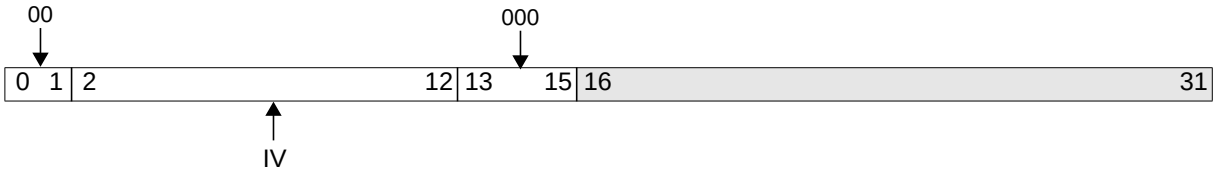


Figure 21-103. Refresh Timing Register (SDRAM0_RTR)

0:1		Always 0b00	
2:12	IV	Interval Programmable between 0b00000000000 and 0b11111111111 (that is, 0x0000 to 0x3FF8 for the 16-bit field).	Reset value is 0b00010111110 (that is, 0x05F0 for the complete 16-bit field)
13:15		Always 0b000	
16:31		Reserved	

SDRAM0_STATUS

SDRAM Controller Status

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x24

See “Memory Controller Status (SDRAM0_STATUS)” on page 14-5.

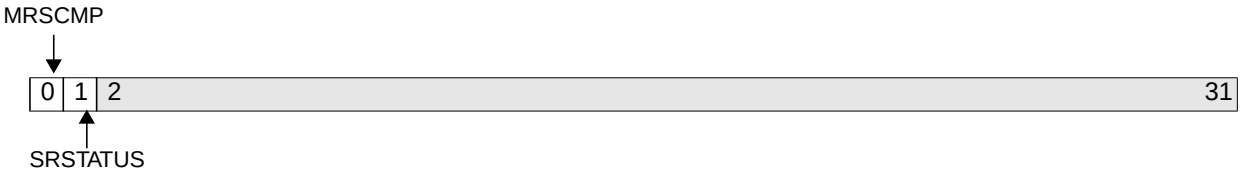


Figure 21-104. Memory Controller Status (SDRAM0_STATUS)

0	MRSCMP	Mode Register Set Complete 0 MRS not complete 1 MRS completed	Set to 1 when the SDRAM controller completes the Mode Register Set Command, which results from setting SDRAM0_CFG[DCE]. Clearing SDRAM0_CFG[DCE] causes this bit to clear in the following MemClkOut1:0 cycle.
1	SRSTATUS	Self-Refresh State 0 Not in Self-Refresh Mode 1 Self-Refresh Mode	See “Self-Refresh” on page 14-18.
2:31		Reserved	

DCR Accessed using SDRAM0_CFGADDR; SDRAM0_CFGDATA; Offset 0x80

See “SDRAM Timing Register (SDRAM0_TR)” on page 14-9.

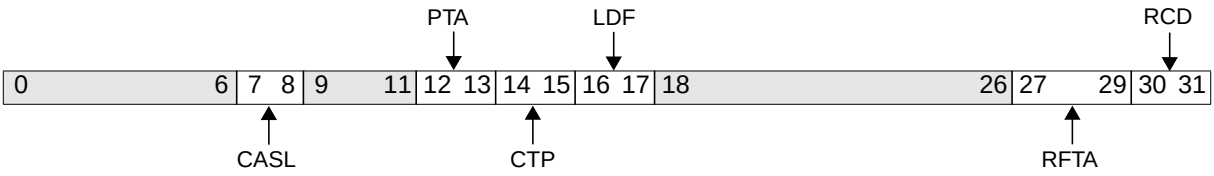


Figure 21-105. SDRAM Timing Register (SDRAM0_TR)

0:6		Reserved
7:8	CASL	SDRAM $\overline{CAS}$ latency. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
9:11		Reserved
12:13	PTA	SDRAM Precharge Command to next Activate Command minimum. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
14:15	CTP	SDRAM Read / Write Command to Precharge Command minimum. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
16:17	LDF	SDRAM Command Leadoff. 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles
18:26		Reserved
27:29	RFTA	SDRAM $\overline{CAS}$ before $\overline{RAS}$ Refresh Command to next Activate Command minimum. 000 4 MemClkOut1:0 cycles 001 5 MemClkOut1:0 cycles 010 6 MemClkOut1:0 cycles 011 7 MemClkOut1:0 cycles 100 8 MemClkOut1:0 cycles 101 9 MemClkOut1:0 cycles 110 10 MemClkOut1:0 cycles 111 Reserved

SDRAM0_TR

SDRAM Timing Register

30:31	RCD	SDRAM RAS to CAS Delay 00 Reserved 01 2 MemClkOut1:0 cycles 10 3 MemClkOut1:0 cycles 11 4 MemClkOut1:0 cycles

UART Registers

MMIO 0xEF600300 (UART0), 0xEF600400 (UART1)

See “UART Baud-Rate Divisor Latch (LSB) Registers (UARTx_DLL)” on page 17-15.

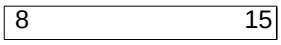


Figure 21-106. UART Baud-Rate Divisor Latch (LSB) Registers (UARTx_DLL)

8:15		Data bits

Note: UARTx_DLL is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

UARTx_DLM

UART Baud-Rate Divisor Latch MSB Registers

MMIO 0xEF600301 (UART0), 0xEF600401 (UART1)

See “UART Baud-Rate Divisor Latch (MSB) Registers (UARTx_DLM)” on page 17-14.

Figure 21-107. UART Baud-Rate Divisor Latch (MSB) Registers (UARTx_DLM)

0:7		Data bits

Note: UARTx_DLM is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

MMIO 0xEF60 0302 (UART0), 0xEF60 0402 (UART1) Write-Only

See “FIFO Control Registers (UARTx_FCR)” on page 17-8.

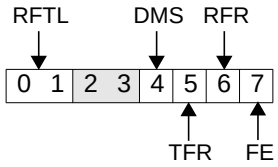


Figure 21-108. UART FIFO Control Registers (UARTx_FCR)

0:1	RFTL	Receiver FIFO Trigger Level 00 1 byte 01 4 bytes 10 8 bytes 11 14 bytes	
2:3		Reserved	
4	DMS	DMA Mode Select 0 Mode 0 = single transfer 1 Mode 1 = multiple transfers	Select single or multiple transfer mode if UARTx_FCR[7] = 1.
5	TFR	Transmitter FIFO Reset 0 Operation complete 1 Reset the transmitter FIFO	A 1 written to this bit clears all bytes in the transmitter FIFO and resets all of its counter logic to 0. The transmitter shift register is not cleared. This bit is self-clearing.
6	RFR	Receiver FIFO Reset 0 Operation complete 1 Reset the receiver FIFO	A 1 written to this bit clears all bytes in the receiver FIFO and resets all of its counter logic to 0. The receiver shift register is not cleared. This bit is self-clearing.
7	FE	FIFO Enable 0 Disable FIFOs 1 Enable FIFOs	When set to 1, both the receiver and transmitter FIFOs are enabled. When set to 0, both receiver and transmitter FIFOs are reset. Data is automatically cleared from both FIFOs when changing to and from FIFO and 16450 modes. Programming other bits will be ignored if this bit is not a 1.

Note: UARTx_FCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

UARTx_IER

UART Interrupt Enable Registers

MMIO 0xEF600301 (UART0), xEF600401 (UART1)

See “Interrupt Enable Registers (UARTx_IER)” on page 17-5.

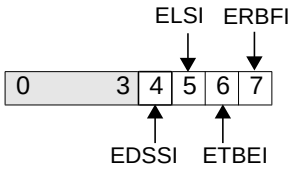


Figure 21-109. UART Interrupt Enable Registers (UARTx_IER)

0:3		Reserved	Always 0.
4	EDSSI	Enable Modem Status Interrupt 0 Disable modem status interrupt 1 Enable modem status interrupt	
5	ELSI	Enable Receiver Line Status Interrupt 0 Disable receiver line status interrupt 1 Enable receiver line status interrupt	
6	ETBEI	Enable Transmitter Holding Register Empty Interrupt 0 Disable transmitter holding register empty interrupt 1 Enable transmitter holding register empty interrupt	
7	ERBFI	Enable Received Data Available Interrupt 0 Disable received data available interrupt 1 Enable received data available interrupt	In FIFO mode, timeout interrupts follow the enable/disable state of ERDAI.

Note: UARTx_IER is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

MMIO 0xEF60 0302 (UART0), 0xEF60 0402 (UART1) Read-Only

See “Interrupt Identification Registers (UARTx_IIR)” on page 17-6.

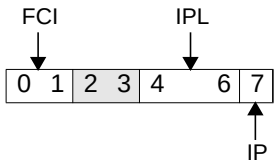


Figure 21-110. UART Interrupt Identification Registers (UARTx_IIR)

0:1	FCI	FIFO Control Indicator 00 FIFOs disabled (UARTx_FCR[FC] = 0) 01 Reserved 10 Reserved 11 FIFOs enabled (UARTx_FCR[FC] = 1)	
2:3		Reserved	
4:6	IPL	Interrupt Priority Level 000 Priority level 4 001 Priority level 3 010 Priority level 2 011 Priority level 1 100 Reserved 101 Reserved 110 Priority level 2 111 Reserved	See Table 17-3. Note: Priority 1 is highest priority.
7	IP	Interrupt Pending 0 Interrupt is pending 1 No interrupt pending	When set to 0, IIR contents can be used as a pointer to the appropriate interrupt service routine.

Note: UARTx_IIR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

UARTx_LCR

UART Line Control Registers

MMIO 0xEF600303 (UART0), 0xEF600403 (UART1)

See “Line Control Registers (UARTx_LCR)” on page 17-9.

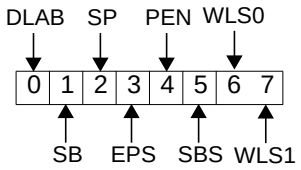


Figure 21-111. UART Line Control Registers (UARTx_LCR)

0	DLAB	Divisor Latch Access Bit 0 Address RBR, THR and IER with LTADR2–0 for read or write operation 1 Address Divisor Latches with LTADR2–0 for read or write operation	
1	SB	Set Break 0 Disable Break 1 Enable Break	Causes a break condition to be transmitted to the UART when the core is receiving. SOUT is forced to the spacing state (0). This bit acts only on SOUT and has no effect on the transmitter logic.
2	SP	Sticky Parity 0 Disable sticky parity 1 Enable sticky parity	If UARTx_LCR[EPS] = 1 and UARTx_LCR[PE] = 1, the parity bit is transmitted and checked as 0. If UARTx_LCR [EPS] = 0 and UARTx_LCR[PE] = 1,the parity bit is transmitted and checked as 1.
3	EPS	Even Parity Select 0 Generate odd parity 1 Generate even parity	This bit is significant only if UARTx_LCR[PE] = 1.
4	PEN	Parity Enable 0 Disable parity checking 1 Enable parity checking	
5	SBS	Stop Bit Select 0 Characters have 1 stop bit 1 Characters have 1.5 or 2 stop bits	If UARTx_LCR[CL] = 00, characters have 1.5 stop bits. For any other value of UARTx_LCR[CL], characters have 2 stop bits. The receiver checks the first stop bit only, regardless of how many stop bits are selected.
6:7	WLS0, WLS1	Word Length Select Bits 0,1 00 Use 5-bit characters 01 Use 6-bit characters 10 Use 7-bit characters 11 Use 8-bit characters	

Note: UARTx_LCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

MMIO 0xEF600305 (UART0), 0xEF600405 (UART1)

See “Line Status Registers (UARTx_LSR)” on page 17-11.

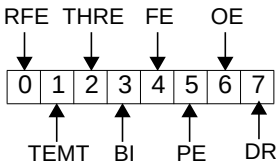


Figure 21-112. UART Line Status Registers (UARTx_LSR)

0	RFE	Receiver FIFO Error Indicator 0 In FIFO mode, reset to 0 when the processor reads the UARTx_LSR, provided there are no subsequent errors in the FIFO. 1 There are one or more instances of parity error, framing error or break indication in the FIFO.	Always 0 in 16450 mode.
1	TEMT	Transmitter Empty Indicator 0 Reset to 0 whenever the THR or the transmitter shift register contain a character. In FIFO mode, it is reset to 0 whenever the transmitter FIFO or the transmitter shift register contain a character. 1 Set to 1 when the THR and the Transmitter shift register are both empty. In FIFO mode, it is set to 1 when the transmitter FIFO and the transmitter shift register are both empty.	
2	THRE	Transmitter Holding Register Empty Indicator 0 Concurrent reset to 0 with the loading of the THR by the processor. In FIFO mode it is reset to 0 when at least one byte is written to the transmitter FIFO. 1 Set to 1 when the UART is ready to accept a new character for transmission. In FIFO mode, this bit is set when the transmitter FIFO is empty.	When UARTx_IER[THRE] = 1, the UART issues an interrupt to the PPC405CR interrupt controller. This bit is set to 1 when a character is transferred from the THR to the transmitter shift register.
3	BI	Break Interrupt Indicator. 0 Reset to 0 whenever processor reads Line Status Register (LSR). 1 Set to 1 whenever the received data input is held at the spacing level (0) for longer than a full word transmission time.	The word transmission time is the time required for the start bit, data bits (can be 5–8 bits), parity and stop bits. In FIFO mode, this error is reported to the processor when the character associated with the error is at the top of the FIFO. Only one 0 character is loaded into the receiver FIFO when a break occurs. After the next valid start bit is received and is in the marking state, the next character transfer is enabled. The error causes a Receiver Line Status Interrupt.

UARTx_LSR (cont.)

UART Line Status Registers

4	FE	Framing Error Indicator. 0 Reset to 0 whenever processor reads LSR. 1 Set to 1 whenever stop bit following the last data bit or parity bit is detected as 0 (spacing level).Indicates that a valid stop bit was not found in the received character.	Error causes a Receiver Line Status Interrupt.
5	PE	Parity Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Indicates that the received data character does not have the correct parity as determined by the even parity select bit (UARTx_LCR.[EPS]). Set to 1 upon detection of a parity error.	In FIFO mode, this error is revealed to the processor when the character this error is associated with is at the top of the FIFO. Error causes a Receiver Line Status Interrupt.
6	OE	Overrun Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Data in the RBR was read by the processor before the next character was transferred into the UARTx_RBR, hence the original data was lost.	In FIFO mode, if the incoming data continues to fill the FIFO beyond the trigger level, an OE occurs only after the FIFO is completely full and the entire next character has been received in the receiver shift register. The processor is informed of the OE immediately upon occurrence. The character in the shift register will be overwritten and will not be transferred to the FIFO. Error causes a Receiver Line Status Interrupt.
7	DR	Receiver Data Ready Indicator. 0 Reset to 0 when all data has been read from the receiver FIFO or the UARTx_RBR. 1 An entire incoming character has been received into the UARTx_RBR or receiver FIFO.	

Note: UARTx_LSR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

MMIO 0xEF600304 (UART0), 0xEF600404 (UART1)

See “Modem Control Registers (UARTx_MCR)” on page 17-10.

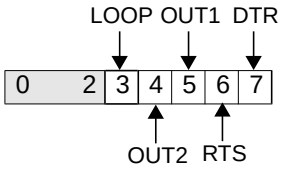


Figure 21-113. UART Modem Control Registers (UARTx_MCR)

0:2		Reserved	Always 0.
3	LOOP	Loopback Mode 0 Disabled 1 Enabled	Provides a local loopback feature for diagnostic testing of the UART. The following occurs: 1. SOUT is set to the marking state (logic 1) SIN is disconnected. 2. The output of the transmitter shift register feeds the input of the receiver shift register. 3. The four modem control inputs $\overline{DSR}$, $\overline{CTS}$, $\overline{RI}$, and $\overline{DCD}$ are disconnected. 4. The four modem control outputs $\overline{DTR}$, $\overline{RTS}$, $\overline{OUT1}$, and $\overline{OUT2}$ are set to a logic 1 (their inactive state). 5. The four modem control outputs are connected internally to the four modem control inputs. Transmitted data is immediately received to verify the UART transmit and receive data paths. Receiver and transmitter interrupts are operational. Their sources are external to the UART. Also operational are the modem control interrupts, but their source is the low-order 4 bits of UARTx_MCR instead of the modem control inputs to the UART. UARTx_IER still controls the interrupts.
4	OUT2	User Output 2 0 $\overline{OUT2}$ inactive (1) 1 $\overline{OUT2}$ active (0)	Auxiliary user designated output.
5	OUT1	User Output 1 0 $\overline{OUT1}$ inactive (1) 1 $\overline{OUT1}$ active (0)	Auxiliary user designated output.
6	RTS	Request To Send 0 $\overline{RTS}$ inactive (1) 1 $\overline{RTS}$ active (0)	
7	DTR	Data Terminal Ready 0 $\overline{DTR}$ inactive (1) 1 $\overline{DTR}$ active (0)	

Note: UARTx_MCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

UARTx_MSR

UART Modem Status Registers

MMIO 0xEF600306 (UART0), 0xEF600406 (UART1)

See “Modem Status Registers (UARTx_MSR)” on page 17-13.

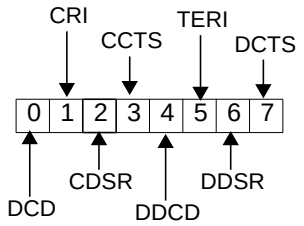


Figure 21-114. UART Modem Status Registers (UARTx_MSR)

0	DCD	Data Carrier Detect	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT2].
1	CRI	Complement of Ring Indicator	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT1].
2	CDSR	Complement of Data Set Ready	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[DTR].
3	CCTS	Complement of Clear To Send	In loopback mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[RTS].
4	DDCD	Delta Data Carrier Detect 0 Set when processor reads the Modem Status Register 1 $\overline{\text{DCD}}$ input changed state	Indicates that the $\overline{\text{DCD}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
5	TERI	Trailing Edge of Ring Indicator 0 Set when processor reads the Modem Status Register 1 $\overline{\text{RI}}$ input changed from 0 to 1	Indicates that the $\overline{\text{RI}}$ input to the UART changed from 0 to 1 since the processor last read the Modem Status Register. A modem status interrupt is generated.
6	DDSR	Delta Data Set Ready 0 Set when processor reads the Modem Status Register 1 $\overline{\text{DSR}}$ input changed state	Indicates that the $\overline{\text{DSR}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
7	DCTS	Delta Clear To Send 0 Set when processor reads the Modem Status Register 1 $\overline{\text{CTS}}$ input changed state	Indicates that the $\overline{\text{CTS}}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.

Note: UARTx_MSR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

MMIO 0xEF600300 (UART0), 0xEF600400 (UART1) Read-Only

See “Receiver Buffer Registers (UARTx_RBR)” on page 17-5.

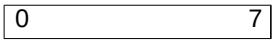


Figure 21-115. UART Receiver Buffer Registers (UARTx_RBR)

0:7		Data bit

Note: UARTx_RBR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

UARTx_SCR

UART Scratchpad Registers

MMIO 0xEF600307 (UART0), 0xEF600407 (UART1)

See “Scratchpad Registers (UARTx_SCR)” on page 17-14.

Figure 21-116. Scratchpad Registers (UARTx_SCR)

0:7		Data bits

Note: UARTx_SCR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

MMIO 0xEF600300 (UART0), 0xEF600400 (UART1) Write-Only

See “Transmitter Holding Registers (UARTx_THR)” on page 17-5.

Figure 21-117. UART Transmitter Holding Registers (UARTx_THR)

0:7		Data bit

Note: UARTx_THR is shown in standard PowerPC bit notation, where 0 is the MSb and 7 is the LSb.

UIC0_CR

UIC Critical Register

UIC Registers

DCR 0x0C3

See “UIC Critical Register (UIC0_CR)” on page 9-7.

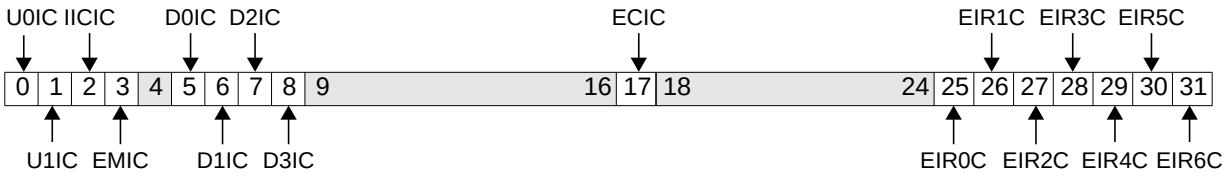


Figure 21-118. UIC Critical Register (UIC0_CR)

0	U0IC	UART0 Interrupt Class 0 UART0 interrupt is non-critical. 1 UART0 interrupt is critical.
1	U1IC	UART1 Interrupt Class 0 UART1 interrupt is non-critical. 1 UART1 interrupt is critical.
2	IICIC	IIC Interrupt Class 0 IIC interrupt is non-critical. 1 IIC interrupt is critical.
3	EMIC	External Master Interrupt Class 0 External master interrupt is non-critical. 1 External master interrupt is critical.
4		Reserved
5	D0IC	DMA Channel 0 Interrupt Class 0 DMA channel 0 interrupt is non-critical. 1 DMA channel 0 interrupt is critical.
6	D1IC	DMA Channel 1 Interrupt Class 0 DMA channel 1 interrupt is non-critical. 1 DMA channel 1 interrupt is critical.
7	D2IC	DMA Channel 2 Interrupt Class 0 DMA channel 2 interrupt is non-critical. 1 DMA channel 2 interrupt is critical.
8	D3IC	DMA Channel 3 Interrupt Class 0 DMA channel 3 interrupt is non-critical. 1 DMA channel 3 interrupt is critical.
9:16		Reserved
17	ECIC	ECC Correctable Error Interrupt Class 0 ECC correctable error interrupt is non-critical. 1 ECC correctable error interrupt is critical.
18:24		Reserved
25	EIR0C	External IRQ 0 Class 0 An external IRQ 0 interrupt is non-critical. 1 An external IRQ 0 interrupt is critical.

26	EIR1C	External IRQ 1 Class 0 An external IRQ 1 interrupt is non-critical. 1 An external IRQ 1 interrupt is critical.
27	EIR2C	External IRQ 2 Class 0 An external IRQ 2 interrupt is non-critical. 1 An external IRQ 2 interrupt is critical.
28	EIR3C	External IRQ 3 Class 0 An external IRQ 3 interrupt is non-critical. 1 An external IRQ 3 interrupt is critical.
29	EIR4C	External IRQ 4 Class 0 An external IRQ 4 interrupt is non-critical. 1 An external IRQ 4 interrupt is critical.
30	EIR5C	External IRQ 5 Class 0 An external IRQ 5 interrupt is non-critical. 1 An external IRQ 5 interrupt is critical.
31	EIR6C	External IRQ 6 Class 0 An external IRQ 6 interrupt is non-critical. 1 An external IRQ 6 interrupt is critical.

UIC0_ER

UIC Interrupt Enable Register

DCR 0x0C2

See “UIC Enable Register (UIC0_ER)” on page 9-5.

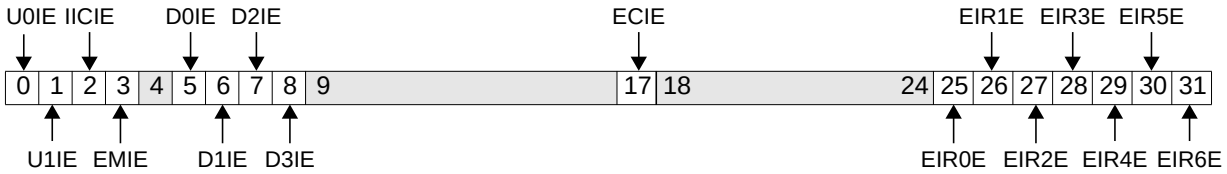


Figure 21-119. UIC Enable Register (UIC0_ER)

0	U0IE	UART0 Interrupt Enable 0 UART0 interrupt is disabled. 1 UART0 interrupt is enabled.
1	U1IE	UART1 Interrupt Enable 0 UART1 interrupt is disabled. 1 UART1 interrupt is enabled.
2	IICIE	IIC Interrupt Enable 0 IIC interrupt is disabled. 1 IIC interrupt is enabled.
3	EMIE	External Master Interrupt Enable 100 External master interrupt is disabled. 1 0xxExternal master interrupt is enabled.
4		Reserved
5	D0IE	DMA Channel 0 Interrupt Enable 0 DMA channel 0 interrupt is disabled. 1 DMA channel 0 interrupt is enabled.
6	D1IE	DMA Channel 1 Interrupt Enable 0 DMA channel 1 interrupt is disabled. 1 DMA channel 1 interrupt is enabled.
7	D2IE	DMA Channel 2 Interrupt Enable 0 DMA channel 2 interrupt is disabled. 1 DMA channel 2 interrupt is enabled.
8	D3IE	DMA Channel 3 Interrupt Enable 0 DMA channel 3 interrupt is disabled. 1 DMA channel 3 interrupt is enabled.
9:16		Reserved
17	ECIE	ECC Correctable Error Interrupt Enable 0 ECC correctable error interrupt is disabled. 1 ECC correctable error interrupt is enabled.
18:24		Reserved
25	EIR0E	External IRQ 0 Interrupt Enable 0 An external IRQ 0 interrupt is disabled. 1 An external IRQ 0 interrupt is enabled.

26	EIR1E	External IRQ 1 Interrupt Enable 0 An external IRQ 1 interrupt is disabled. 1 An external IRQ 1 interrupt is enabled.
27	EIR2E	External IRQ 2 Interrupt Enable 0 An external IRQ 2 interrupt is disabled. 1 An external IRQ 2 interrupt is enabled.
28	EIR3E	External IRQ 3 Interrupt Enable 0 An external IRQ 3 interrupt is disabled. 1 An external IRQ 3 interrupt is enabled.
29	EIR4E	External IRQ 4 Interrupt Enable 0 An external IRQ 4 interrupt is disabled. 1 An external IRQ 4 interrupt is enabled.
30	EIR5E	External IRQ 5 Interrupt Enable 0 An external IRQ 5 interrupt is disabled. 1 An external IRQ 5 interrupt is enabled.
31	EIR6E	External IRQ 6 Interrupt Enable 0 An external IRQ 6 interrupt is disabled. 1 An external IRQ 6 interrupt is enabled.

UIC0_MSR

UIC Masked Status Register

DCR 0x0C6 Read-Only

See “UIC Masked Status Register (UIC0_MSR)” on page 9-12.

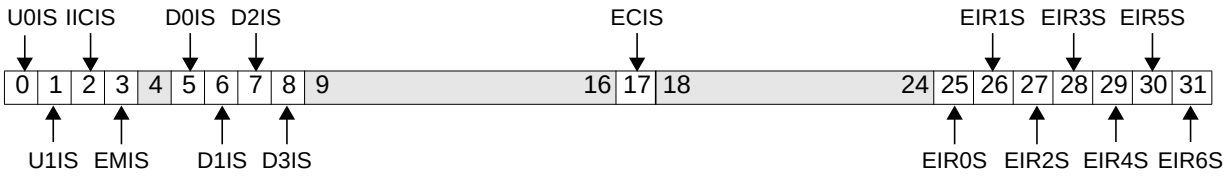


Figure 21-120. UIC Masked Status Register (UIC0_MSR)

0	U0IS	UART0 Masked Interrupt Status 0 A UART0 interrupt has not occurred. 1 A UART0 interrupt occurred.
1	U1IS	UART1 Masked Interrupt Status 0 A UART1 interrupt has not occurred. 1 A UART1 interrupt occurred.
2	IICIS	IIC Masked Interrupt Status 0 An IIC interrupt has not occurred. 1 An IIC interrupt occurred.
3	EMIS	External Master Masked Interrupt Status 0 An external master interrupt has not occurred. 1 An external master interrupt occurred.
4		Reserved
5	D0IS	DMA Channel 0 Masked Interrupt Status 0 A DMA channel 0 interrupt has not occurred. 1 A DMA channel 0 interrupt occurred.
6	D1IS	DMA Channel 1 Masked Interrupt Status 0 A DMA channel 1 interrupt has not occurred. 1 A DMA channel 1 interrupt occurred.
7	D2IS	DMA Channel 2 Masked Interrupt Status 0 A DMA channel 2 interrupt has not occurred. 1 A DMA channel 2 interrupt occurred.
8	D3IS	DMA Channel 3 Masked Interrupt Status 0 A DMA channel 3 interrupt has not occurred. 1 A DMA channel 3 interrupt occurred.
9:16		Reserved

17	ECIS	ECC Correctable Error Masked Interrupt Status 0 An ECC correctable error interrupt did not occur. 1 An ECC correctable error interrupt occurred.
18:24		Reserved
25	EIR0S	External IRQ 0 Masked Status 0 An external IRQ 0 interrupt has not occurred. 1 An external IRQ 0 interrupt occurred.
26	EIR1S	External IRQ 1 Masked Status 0 An external IRQ 1 interrupt has not occurred. 1 An external IRQ 1 interrupt occurred.
27	EIR2S	External IRQ 2 Masked Status 0 An external IRQ 2 interrupt has not occurred. 1 An external IRQ 2 interrupt occurred.
28	EIR3S	External IRQ 3 Masked Status 0 An external IRQ 3 interrupt has not occurred. 1 An external IRQ 3 interrupt occurred.
29	EIR4S	External IRQ 4 Masked Status 0 An external IRQ 4 interrupt has not occurred. 1 An external IRQ 4 interrupt occurred.
30	EIR5S	External IRQ 5 Masked Status 0 An external IRQ 5 interrupt has not occurred. 1 An external IRQ 5 interrupt occurred.
31	EIR6S	External IRQ 6 Masked Status 0 An external IRQ 6 interrupt has not occurred. 1 An external IRQ 6 interrupt occurred.

UIC0_PR

UIC Polarity Register

DCR 0x0C4

See “UIC Polarity Register (UIC0_PR))” on page 9-8.

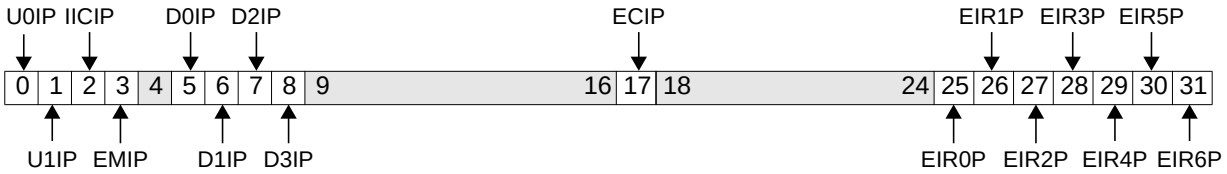


Figure 21-121. UIC Polarity Register (UIC0_PR)

0	U0IP	UART0 Interrupt Polarity 0 UART0 interrupt has negative polarity. 1 UART0 interrupt has positive polarity.	Must be set to 1.
1	U1IP	UART1 Interrupt Polarity 0 UART1 interrupt has negative polarity. 1 UART1 interrupt has positive polarity.	Must be set to 1.
2	IICIP	IIC Interrupt Polarity 0 IIC interrupt has negative polarity. 1 IIC interrupt has positive polarity.	Must be set to 1.
3	EMIP	External Master Interrupt Polarity 0 External master interrupt has negative polarity. 1 External master interrupt has positive polarity.	Must be set to 1.
4		Reserved	
5	D0IP	DMA Channel 0 Interrupt Polarity 0 DMA channel 0 interrupt has negative polarity. 1 DMA channel 0 interrupt has positive polarity.	Must be set to 1.
6	D1IP	DMA Channel 1 Interrupt Polarity 0 DMA channel 1 interrupt has negative polarity. 1 DMA channel 1 interrupt has positive polarity.	Must be set to 1.
7	D2IP	DMA Channel 2 Interrupt Polarity 0 DMA channel 2 interrupt has negative polarity. 1 DMA channel 2 interrupt has positive polarity.	Must be set to 1.
8	D3IP	DMA Channel 3 Interrupt Polarity 0 DMA channel 3 interrupt has negative polarity. 1 DMA channel 3 interrupt has positive polarity.	Must be set to 1.
9:16		Reserved	
17	ECIP	ECC Correctable Error Interrupt Polarity 0 ECC correctable error interrupt has negative polarity. 1 ECC correctable error interrupt has positive polarity.	Must be set to 1.
18:24		Reserved	
25	EIR0P	External IRQ 0 Polarity 0 An external IRQ 0 interrupt has negative polarity. 1 An external IRQ 0 interrupt has positive polarity.	

26	EIR1P	External IRQ 1 Polarity 0 An external IRQ 1 interrupt has negative polarity. 1 An external IRQ 1 interrupt has positive polarity.
27	EIR2P	External IRQ 2 Polarity 0 An external IRQ 2 interrupt has negative polarity. 1 An external IRQ 2 interrupt has positive polarity.
28	EIR3P	External IRQ 3 Polarity 0 An external IRQ 3 interrupt has negative polarity. 1 An external IRQ 3 interrupt has positive polarity.
29	EIR4P	External IRQ 4 Polarity 0 An external IRQ 4 interrupt has negative polarity. 1 An external IRQ 4 interrupt has positive polarity.
30	EIR5P	External IRQ 5 Polarity 0 An external IRQ 5 interrupt has negative polarity. 1 An external IRQ 5 interrupt has positive polarity.
31	EIR6P	External IRQ 6 Polarity 0 An external IRQ 6 interrupt has negative polarity. 1 An external IRQ 6 interrupt has positive polarity.

UIC0_SR

UIC Status Register

DCR 0x0C0 Read/Clear

See “UIC Status Register (UIC0_SR)” on page 9-3.

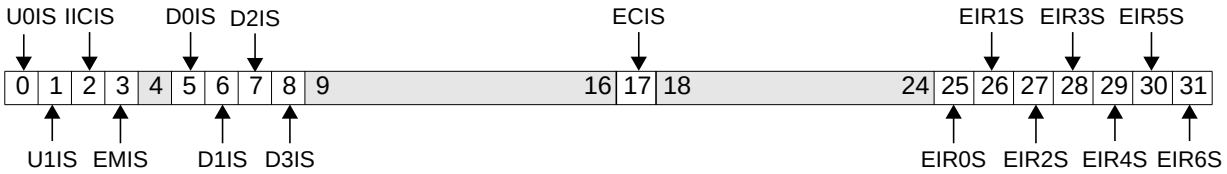


Figure 21-122. UIC Status Register (UIC0_SR)

0	U0IS	UART0 Interrupt Status 0 A UART0 interrupt has not occurred. 1 A UART0 interrupt occurred.
1	U1IS	UART1 Interrupt Status 0 A UART1 interrupt has not occurred. 1 A UART1 interrupt occurred.
2	IICIS	IIC Interrupt Status 0 An IIC interrupt has not occurred. 1 An IIC interrupt occurred.
3	EMIS	External Master Interrupt Status 0 An external master interrupt has not occurred. 1 An external master interrupt occurred.
4		Reserved
5	D0IS	DMA Channel 0 Interrupt Status 0 A DMA channel 0 interrupt has not occurred. 1 A DMA channel 0 interrupt occurred.
6	D1IS	DMA Channel 1 Interrupt Status 0 A DMA channel 1 interrupt has not occurred. 1 A DMA channel 1 interrupt occurred.
7	D2IS	DMA Channel 2 Interrupt Status 0 A DMA channel 2 interrupt has not occurred. 1 A DMA channel 2 interrupt occurred.
8	D3IS	DMA Channel 3 Interrupt Status 0 A DMA channel 3 interrupt has not occurred. 1 A DMA channel 3 interrupt occurred.
9:16		Reserved
17	ECIS	ECC Correctable Error Interrupt Status 0 An ECC correctable error interrupt did not occur. 1 An ECC correctable error interrupt occurred.

18:24		Reserved
25	EIR0S	External IRQ 0 Status 0 An external IRQ 0 interrupt has not occurred. 1 An external IRQ 0 interrupt occurred.
26	EIR1S	External IRQ 1 Status 0 An external IRQ 1 interrupt has not occurred. 1 An external IRQ 1 interrupt occurred.
27	EIR2S	External IRQ 2 Status 0 An external IRQ 2 interrupt has not occurred. 1 An external IRQ 2 interrupt occurred.
28	EIR3S	External IRQ 3 Status 0 An external IRQ 3 interrupt has not occurred. 1 An external IRQ 3 interrupt occurred.
29	EIR4S	External IRQ 4 Status 0 An external IRQ 4 interrupt has not occurred. 1 An external IRQ 4 interrupt occurred.
30	EIR5S	External IRQ 5 Status 0 An external IRQ 5 interrupt has not occurred. 1 An external IRQ 5 interrupt occurred.
31	EIR6S	External IRQ 6 Status 0 An external IRQ 6 interrupt has not occurred. 1 An external IRQ 6 interrupt occurred.

UIC0_TR

UIC Triggering Register

DCR 0x0C5

See “UIC Trigger Register (UIC0_TR)” on page 9-10.

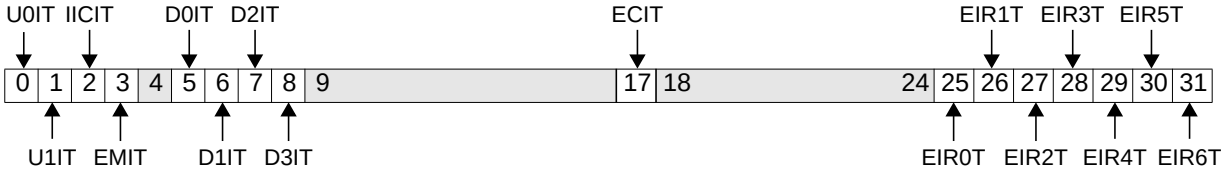


Figure 21-123. UIC Trigger Register (UIC0_TR)

0	U0IT	UART0 Interrupt Trigger 0 UART0 interrupt is level-sensitive. 1 UART0 interrupt is edge-sensitive.	Must be set to 0.
1	U1IT	UART1 Interrupt Trigger 0 UART1 interrupt is level-sensitive. 1 UART1 interrupt is edge-sensitive.	Must be set to 0.
2	IICIT	IIC Interrupt Trigger 0 IIC interrupt is level-sensitive. 1 IIC interrupt is edge-sensitive.	Must be set to 0.
3	EMIT	External Master Interrupt Trigger 0 External master interrupt is level-sensitive. 1 External master interrupt is edge-sensitive.	Must be set to 1.
4		Reserved	
5	D0IT	DMA Channel 0 Interrupt Trigger 0 DMA channel 0 interrupt is level-sensitive. 1 DMA channel 0 interrupt is edge-sensitive.	Must be set to 0.
6	D1IT	DMA Channel 1 Interrupt Trigger 0 DMA channel 1 interrupt is level-sensitive. 1 DMA channel 1 interrupt is edge-sensitive.	Must be set to 0.
7	D2IT	DMA Channel 2 Interrupt Trigger 0 DMA channel 2 interrupt is level-sensitive. 1 DMA channel 2 interrupt is edge-sensitive.	Must be set to 0.
8	D3IT	DMA Channel 3 Interrupt Trigger 0 DMA channel 3 interrupt is level-sensitive. 1 DMA channel 3 interrupt is edge-sensitive.	Must be set to 0.
9:16		Reserved	

17	ECIT	ECC Correctable Error Interrupt Trigger 0 ECC correctable error interrupt is level-sensitive. 1 ECC correctable error interrupt is edge-sensitive.	Must be set to 0.
18:24		Reserved	
25	EIR0T	External IRQ 0 Trigger 0 An external IRQ 0 interrupt is level-sensitive. 1 An external IRQ 0 interrupt is edge-sensitive.	
26	EIR1T	External IRQ 1 Trigger 0 An external IRQ 1 interrupt is level-sensitive. 1 An external IRQ 1 interrupt is edge-sensitive.	
27	EIR2T	External IRQ 2 Trigger 0 An external IRQ 2 interrupt is level-sensitive. 1 An external IRQ 2 interrupt is edge-sensitive.	
28	EIR3T	External IRQ 3 Trigger 0 An external IRQ 3 interrupt is level-sensitive. 1 An external IRQ 3 interrupt is edge-sensitive.	
29	EIR4T	External IRQ 4 Trigger 0 An external IRQ 4 interrupt is level-sensitive. 1 An external IRQ 4 interrupt is edge-sensitive.	
30	EIR5T	External IRQ 5 Trigger 0 An external IRQ 5 interrupt is level-sensitive. 1 An external IRQ 5 interrupt is edge-sensitive.	
31	EIR6T	External IRQ 6 Trigger 0 An external IRQ 6 interrupt is level-sensitive. 1 An external IRQ 6 interrupt is edge-sensitive.	

UIC0_VCR

UIC Vector Configuration Register

DCR 0x0C8 Write-Only

See “UIC Vector Configuration Register (UIC0_VCR)” on page 9-14.

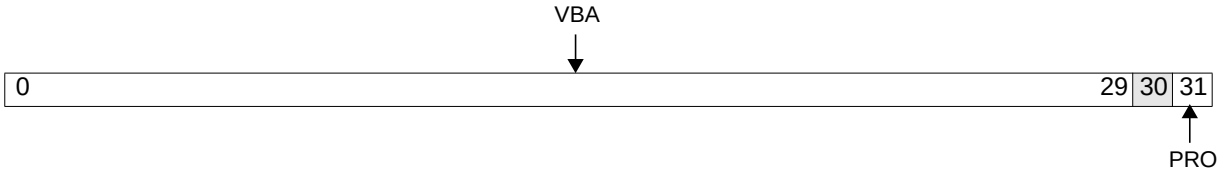


Figure 21-124. UIC Vector Configuration Register (UIC0_VCR)

0:29	VBA	Vector Base Address
30		Reserved
31	PRO	Priority Ordering 0 UIC0_SR[0] is the highest priority interrupt. 1 UIC0_SR[31] is the highest priority interrupt. Note: Vector generation is not performed for non-critical interrupts.

DCR 0x0C7 Read-Only

See “UIC Vector Register (UIC0_VR)” on page 9-15.

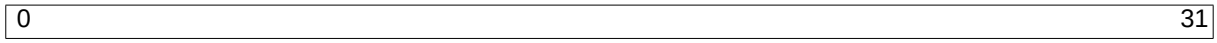


Figure 21-125. UIC Vector Register (UIC0_VR)

0:31		Interrupt Vector

UIC0_VR

UIC Vector Register

Chapter 22. Signal Summary

This chapter provides detailed information on the PPC405CR I/O signals.

22.1 Signals Listed Alphabetically

Table 22-1 lists the PPC405CR signals in alphabetical order. For each signal there is an indication of the interface group to which it belongs and a page reference to the description of the signal in Table 22-2, “Signal Descriptions,” on page 22-4.

Multiplexed signals are shown in brackets following the first signal name assigned to each multiplexed ball (for example, IRQ0:6[GPIO17:23]). Active-low signals are shown with an overbar on the signal name (for example, $\overline{\text{ExtAck}}$)..

Table 22-1. Alphabetical Signal List

Signal Name	Interface	Page
BA1:0	SDRAM	22-4
$\overline{\text{BankSel0:3}}$	SDRAM	22-4
BusReq	External Master Peripheral	22-5
$\overline{\text{CAS}}$	SDRAM	22-4
ClkEn0:1	SDRAM	22-4
DMAAck0:3	External Slave Peripheral	22-5
DMAReq0:3	External Slave Peripheral	22-5
DQM0:3	SDRAM	22-4
DQMCB	SDRAM	22-4
DrvrInh1:2	System	22-7
ECC0:7	SDRAM	22-4
EOT0:3[TC0:3]	External Slave Peripheral	22-5
$\overline{\text{ExtAck}}$	External Master Peripheral	22-5
$\overline{\text{ExtReq}}$	External Master Peripheral	22-5
$\overline{\text{ExtReset}}$	External Master Peripheral	22-5
GPIO1:2[TS1E:TS2E]	System	22-1
GPIO3:4[TS1O:TS2O]	System	22-1
GPIO5:8[TS3:6]	System	22-1
GPIO9[TrcClk]	System	22-1
$\overline{\text{Halt}}$	System	22-7

Table 22-1. Alphabetical Signal List

Signal Name	Interface	Page
HoldAck	External Master Peripheral	22-5
HoldPri	External Master Peripheral	22-5
HoldReq	External Master Peripheral	22-5
IICSCL	Internal Peripheral	22-6
IICSDA	Internal Peripheral	22-6
IRQ0:6[GPIO17:23]	Interrupts	22-2
MemAddr0:12	SDRAM	22-4
MemClkOut0:1	SDRAM	22-4
MemData0:31	SDRAM	22-4
PerAddr0:31	External Slave Peripheral	22-4
$\overline{\text{PerBLast}}$	External Slave Peripheral	22-5
PerClk	External Master Peripheral	22-5
$\overline{\text{PerCS0}}$	External Slave Peripheral	22-4
$\overline{\text{PerCS1:7}}$[GPIO10:16]	External Slave Peripheral	22-2
PerData0:31	External Slave Peripheral	22-4
PerErr	External Master Peripheral	22-5
$\overline{\text{PerOE}}$	External Slave Peripheral	22-5
PerPar0:3	External Slave Peripheral	22-4
PerReady	External Slave Peripheral	22-5
PerR/$\overline{\text{W}}$	External Slave Peripheral	22-5
$\overline{\text{PerWBE0:3}}$	External Slave Peripheral	22-4
$\overline{\text{PerWE}}$	External Slave Peripheral	22-4
$\overline{\text{RAS}}$	SDRAM	22-4
RcvrInh	System	22-7
SysClk	System	22-7
SysErr	System	22-7
$\overline{\text{SysReset}}$	System	22-7
TCK	JTAG	22-6
TDI	JTAG	22-6
TDO	JTAG	22-6
TestEn	System	22-7

Table 22-1. Alphabetical Signal List

Signal Name	Interface	Page
TmrClk	System	22-7
TMS	JTAG	22-6
$\overline{\text{TRST}}$	JTAG	22-7
$\overline{\text{UART0\_CTS}}$	Internal Peripheral	22-6
$\overline{\text{UART0\_DCD}}$	Internal Peripheral	22-6
$\overline{\text{UART0\_DSR}}$	Internal Peripheral	22-6
$\overline{\text{UART0\_DTR}}$	Internal Peripheral	22-6
$\overline{\text{UART0\_RI}}$	Internal Peripheral	22-6
$\overline{\text{UART0\_RTS}}$	Internal Peripheral	22-6
UART0_Rx	Internal Peripheral	22-6
UART0_Tx	Internal Peripheral	22-6
$\overline{\text{UART1\_DSR}}$[$\overline{\text{UART1\_CTS}}$]	Internal Peripheral	22-6
$\overline{\text{UART1 \_RTS}}$[$\overline{\text{UART1\_DTR}}$]	Internal Peripheral	22-6
UART1_Rx	Internal Peripheral	22-6
UART1_Tx	Internal Peripheral	22-6
UARTSerClk	Internal Peripheral	22-6
$\overline{\text{WE}}$	SDRAM	22-4

22.2 Signal Descriptions

Each I/O signal is listed with the other signals in the same interface group.

Multiplexed signals are shown in brackets following the first signal name assigned to each multiplexed ball (for example, IRQ0:6[GPIO17:23]). Active-low signals are shown with an overbar on the signal name (for example, $\overline{\text{ExtAck}}$).

Table 22-2. Signal Descriptions

Signal Name	I/O	Function
SDRAM Interface		
MemData0:31	I/O	Memory data bus
MemAddr0:12	O	Memory address bus
BA1:0	O	Bank Address supporting up to 4 internal banks
$\overline{\text{RAS}}$	O	Row Address Strobe
$\overline{\text{CAS}}$	O	Column Address Strobe
DQM0:3	O	DQM for byte lanes 0, 1, 2, and 3.
DQMCB	O	DQM for ECC check bits
ECC0:7	I/O	ECC check bits 0:7
$\overline{\text{BankSel0:3}}$	O	Select up to four external SDRAM banks
$\overline{\text{WE}}$	O	Write Enable
ClkEn0:1	O	ClkEn0:1 provide duplicate clock enables for MemClkOut0:1.
MemClkOut0:1	O	Duplicate SDRAM clock outputs. In limited cases, this allows glueless SDRAM attach without requiring the signal to be repowered by a PLL or zero-delay buffer.
External Slave Peripheral Interface		
PerData0:31	I/O	Peripheral data bus used by PPC405CR when not in external master mode, otherwise used by external master
PerAddr0:31	I/O	Peripheral address bus used by PPC405CR when not in external master mode, otherwise used by external master.
PerPar0:3	I/O	Peripheral byte parity signals
$\overline{\text{PerWBE0:3}}$	I/O	As outputs, these signals can act as byte-enables which are valid for an entire cycle or as write-byte-enables which are valid for each byte on each data transfer, allowing partial word transactions. As outputs, the signals are used by the peripheral controller or DMA controller, depending upon the type of transfer involved. These signals are used as inputs when an external bus master owns the external interface
$\overline{\text{PerWE}}$	I/O	Peripheral write enable. Logical AND of the four $\overline{\text{PerWBE0:3}}$ write byte enables
$\overline{\text{PerCS0}}$	O	Peripheral chip select 0

Table 22-2. Signal Descriptions (continued)

Signal Name	I/O	Function
PerCS1:7[GPIO10:16]	O [I/O]	Seven additional peripheral chip selects or General Purpose I/O - To access this function, software must toggle a DCR bit.
PerOE	O	Used by either peripheral controller or DMA controller depending upon the type of transfer involved.
PerR/W	I/O	Used by PPC405CR when not in external master mode, otherwise used by external master. As output, the signal is used by either peripheral controller or DMA controller depending upon the type of transfer involved.
PerReady	I	Used by a peripheral slave to indicate it is ready to transfer data
PerBLast	I/O	Used by PPC405CR when not in external master mode, otherwise used by external master.
DMAReq0:3	I	Used by slave peripherals to indicate that they are prepared to transfer data.
DMAAck0:3	O	Used by PPC405CR to indicate that data transfers have occurred.
EOT0:3[TC0:3]	I/O	End Of Transfer or Terminal Count
External Master Peripheral Interface		
PerClk	O	Peripheral clock to be used by an external master and by synchronous peripheral slaves
ExtReset	O	Peripheral reset to be used by an external master and by synchronous peripheral slaves
HoldReq	I	Hold Request, used by an external master to request ownership of the peripheral bus
HoldAck	O	Hold Acknowledge, used by PPC405CR to transfer ownership of peripheral bus to an external master
ExtReq	I	Used by an external master to indicate it is prepared to transfer data
ExtAck	O	Used by PPC405CR to indicate that a data transfer occurred.
HoldPri	I	Used by an external master to indicate the priority of a given transfer (0 = high, 1 = low)
BusReq	O	Used when PPC405CR needs to regain control of peripheral interface from an external Master
PerErr	I	Used to record external Master errors and external slave peripheral errors

Table 22-2. Signal Descriptions (continued)

Signal Name	I/O	Function
Internal Peripheral Interface		
UARTSerClk	I	Serial Clock provides an alternative clock to the internally generated serial clock. Used when internally generated baud rates are not satisfactory. This input can be connected to UART0, using CPC0_CR0[U0EC] = 1, or UART1, using CPC0_CR0[U1EC] = 1, or both (CPC0_CR0[U0EC, U1EC] = 1.
UART0_Rx	I	UART0 Serial in data
UART0_Tx	O	UART0 Serial out data
$\overline{\text{UART0\_DCD}}$	I	UART0 Data Carrier Detect
$\overline{\text{UART0\_DSR}}$	I	UART0 Data Set Ready
$\overline{\text{UART0\_CTS}}$	I	UART0 Clear To Send
$\overline{\text{UART0\_DTR}}$	O	UART0 Data Terminal Ready
$\overline{\text{UART0\_RTS}}$	O	UART0 Request To Send
$\overline{\text{UART0\_RI}}$	I	UART0 Ring Indicator
UART1_Rx	I	UART1 Serial In data.
UART1_Tx	O	UART1 Serial Out data
$\overline{\text{UART1\_DSR}}$[$\overline{\text{UART1\_CTS}}$]	I	UART1 Data Set Ready or UART1 Clear To Send. To access this function, software must toggle a DCR bit.
$\overline{\text{UART1\_RTS}}$[$\overline{\text{UART1\_DTR}}$]	O	UART1 Request To Send or UART1 Data Terminal Ready. To access this function, software must toggle a DCR bit.
IICSCL	I/O	IIC Serial Clock
IICSDA	I/O	IIC Serial Data
Interrupts Interface		
IRQ0:6[GPIO17:23]	I [I/O]	Interrupt requests 0–6 or General Purpose I/O. To access this function, software must toggle a DCR bit.
JTAG Interface		
TDI	I	Test data in
TMS	I	JTAG test mode select
TDO	O	Test data out
TCK	I	JTAG test clock

Table 22-2. Signal Descriptions (continued)

Signal Name	I/O	Function
$\overline{\text{TRST}}$	I	JTAG reset
System Interface		
SysClk	I	Main system clock input
$\overline{\text{SysReset}}$	I/O	Main system reset. This signal may be redriven by the PPC405CR to allow a system reset to occur.
SysErr	O	Asserted when a machine check exception is generated.
$\overline{\text{Halt}}$	I	Halt from external debugger.
GPIO1:2[TS1E:TS2E]	I/O [O]	General Purpose I/O or Even Trace execution status. To access this function, software must toggle a DCR bit.
GPIO3:4[TS1O:TS2O]	I/O [O]	General Purpose I/O or Odd Trace execution status. To access this function, software must toggle a DCR bit.
GPIO5:8[TS3:6]	I/O [O]	General Purpose I/O or Trace status. To access this function, software must toggle a DCR bit.
GPIO9[TrcClk]	I/O [O]	General Purpose I/O or Trace interface clock.
TestEn	I	Test Enable. Reserved for manufacturing test.
$\overline{\text{RcvrInh}}$	I	Receiver Inhibit. Reserved for manufacturing test.
$\overline{\text{DrvrInh1}}$ $\overline{\text{DrvrInh2}}$	I	Driver Inhibit 1 and 2. Reserved for manufacturing test.
TmrClk	I	Timer clock. TmrClk is an alternative clock source for the timer facilities. Used when the allowable timer clock intervals provided by the CPU clock are not satisfactory. Enabled when CPC0_CR1[CETE] = 1.

Appendix A. Instruction Summary

This appendix contains PPC405CR instructions summarized alphabetically and by opcode.

“Instruction Set and Extended Mnemonics – Alphabetical” lists all PPC405CR mnemonics, including extended mnemonics, alphabetically. A short functional description is included for each mnemonic.

“Instructions Sorted by Opcode,” on page A-33, lists all PPC405CR instructions, sorted by primary and secondary opcodes. Extended mnemonics are not included in the opcode list.

“Instruction Formats,” on page A-41, illustrates the PPC405CR instruction forms (allowed arrangements of fields within instructions).

A.1 Instruction Set and Extended Mnemonics – Alphabetical

Table A-1 summarizes the PPC405CR 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 Chapter 20, “Instruction Set,” which is also alphabetized under the root form. Chapter 20 also describes the instruction operands and notation.

Note the following for the branch conditional mnemonic:

Bit 4 of the BO field provides a hint about the most likely outcome of a conditional branch. (See “Branch Prediction” on page 3-32 for a detailed description 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 suffixes for the conditional branch mnemonics:

+ 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 “Branch Prediction” on page 3-32 for more information.

Table A-1. PPC405CR Instruction Syntax Summary

Mnemonic	Operands	Function	Other Registers Changed	Page
add	RT, RA, RB	Add (RA) to (RB). Place result in RT.		20-6
add.			CR[CR0]	
addo			XER[SO, OV]	
addo.			CR[CR0] XER[SO, OV]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
addc	RT, RA, RB	Add (RA) to (RB). Place result in RT. Place carry-out in XER[CA].		20-6
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].		20-8
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.		20-9
addic	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].		20-6
addic.	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].	CR[CR0]	20-11
addis	RT, RA, IM	Add (IM $\|\|$ $^{16}0$) to (RA|0). Place result in RT.		20-12
addme	RT, RA	Add XER[CA], (RA), (-1). Place result in RT. Place carry-out in XER[CA].		20-13
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].		20-14
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.		20-15
and.			CR[CR0]	
andc	RA, RS, RB	AND (RS) with ¬(RB). Place result in RA.		20-16
andc.			CR[CR0]	
andi.	RA, RS, IM	AND (RS) with ($^{16}0$ $\|\|$ IM). Place result in RA.	CR[CR0]	20-17
andis.	RA, RS, IM	AND (RS) with (IM $\|\|$ $^{16}0$). Place result in RA.	CR[CR0]	20-18

Table A-1. PPC405CR 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)		20-19
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.	20-20
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.	20-26
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.	20-30
bclrl			CTR if BO$_{2}$ = 0. (LR) ← CIA + 4.	
bctr		Branch unconditionally to address in CTR. *Extended mnemonic for* **bcctr 20,0**		20-26
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**		20-20
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. PPC405CR 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**		20-30
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**		20-20
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**		20-30
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**		20-20
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**		20-30
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**		20-20
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 A-1. PPC405CR Instruction Syntax Summary (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**		20-30
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**		20-20
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**		20-30
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**		20-20
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} = 1to address in LR. *Extended mnemonic for* **bclr 10,cr_bit**		20-30
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**		20-20
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.	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-26
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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**		20-26
bgectrl		*Extended mnemonic for* **bcctrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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**		20-26
blectrl		*Extended mnemonic for* **bcctrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-30
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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.	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-26
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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 A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-26
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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.	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-26
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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**		20-26
btctrl		*Extended mnemonic for* **bcctrl 12,cr_bit**	(LR) ← CIA + 4.	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
btlr	cr_bit	Branch if CR_{cr_bit} = 1, to address in LR. *Extended mnemonic for* **bclr 12,cr_bit**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-147
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**		20-147
clrlslwi.		*Extended mnemonic for* **rlwinm. RA,RS,n,b–n,31–n**	CR[CR0]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-147
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.		20-34
cmpi	BF, 0, RA, IM	Compare (RA) to EXTS(IM), signed. Results in CR[CRn], where n = BF.		20-35
cmpl	BF, 0, RA, RB	Compare (RA) to (RB), unsigned. Results in CR[CRn], where n = BF.		20-36
cmpli	BF, 0, RA, IM	Compare (RA) to ($^{16}0$ || IM), unsigned. Results in CR[CRn], where n = BF.		20-37
cmplw	[BF,] RA, RB	Compare Logical Word. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpl BF,0,RA,RB**		20-36
cmplwi	[BF,] RA, IM	Compare Logical Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpli BF,0,RA,IM**		20-37
cmpw	[BF,] RA, RB	Compare Word. Use CR0 if BF is omitted. *Extended mnemonic for* **cmp BF,0,RA,RB**		20-34
cmpwi	[BF,] RA, IM	Compare Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpi BF,0,RA,IM**		20-35
cntlzw	RA, RS	Count leading zeros in RS. Place result in RA.		20-38
cntlzw.			CR[CR0]	
crand	BT, BA, BB	AND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-39
crandc	BT, BA, BB	AND bit (CR_{BA}) with $\neg$(CR_{BB}). Place result in CR_{BT}.		20-40
crclr	bx	Condition register clear. *Extended mnemonic for* **crxor bx,bx,bx**		20-46
creqv	BT, BA, BB	Equivalence of bit CR_{BA} with CR_{BB}. $CR_{BT} \leftarrow \neg(CR_{BA} \oplus CR_{BB})$		20-41
crmove	bx, by	Condition register move. *Extended mnemonic for* **cror bx,by,by**		20-44
crnand	BT, BA, BB	NAND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-42

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
crnor	BT, BA, BB	NOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-43
crnot	bx, by	Condition register not. *Extended mnemonic for* **crnor bx,by,by**		20-43
cror	BT, BA, BB	OR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-44
crorc	BT, BA, BB	OR bit (CR_{BA}) with $\neg$(CR_{BB}). Place result in CR_{BT}.		20-45
crset	bx	Condition register set. *Extended mnemonic for* **creqv bx,bx,bx**		20-41
crxor	BT, BA, BB	XOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-46
dcba	RA, RB	Speculatively establish the data cache block which contains the effective address (RA|0) + (RB).		20-47
dcbf	RA, RB	Flush (store, then invalidate) the data cache block which contains the effective address (RA|0) + (RB).		20-49
dcbi	RA, RB	Invalidate the data cache block which contains the effective address (RA|0) + (RB).		20-50
dcbst	RA, RB	Store the data cache block which contains the effective address (RA|0) + (RB).		20-51
dcbt	RA, RB	Load the data cache block which contains the effective address (RA|0) + (RB).		20-52
dcbtst	RA,RB	Load the data cache block which contains the effective address (RA|0) + (RB).		20-53
dcbz	RA, RB	Zero the data cache block which contains the effective address (RA|0) + (RB).		20-54
dccci	RA, RB	Invalidate the data cache congruence class associated with the effective address (RA|0) + (RB).		20-56
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.		20-57
divw	RT, RA, RB	Divide (RA) by (RB), signed. Place result in RT.		20-59
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.		20-60
divwu.			CR[CR0]	
divwuo			XER[SO, OV]	
divwuo.			CR[CR0] XER[SO, OV]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

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.		20-61
eqv	RA, RS, RB	Equivalence of (RS) with (RB). (RA) ← ¬((RS) ⊕ (RB))		20-62
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**		20-147
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**		20-147
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.		20-63
extsb.			CR[CR0]	
extsh	RA, RS	Extend the sign of halfword $(RS)_{16:31}$. Place the result in RA.		20-64
extsh.			CR[CR0]	
icbi	RA, RB	Invalidate the instruction cache block which contains the effective address (RA|0) + (RB).		20-65
icbt	RA, RB	Load the instruction cache block which contains the effective address (RA|0) + (RB).		20-63
iccci	RA, RB	Invalidate instruction cache.		20-67
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.		20-68
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**		20-146
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**		20-146
insrwi.		*Extended mnemonic for* **rlwimi. RA,RS,32–b–n,b,b+n–1**	CR[CR0]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
isync		Synchronize execution context by flushing the prefetch queue.		20-70
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**		20-9
lbz	RT, D(RA)	Load byte from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		20-71
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.		20-72
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.		20-73
lbzx	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		20-74
lha	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)).		20-75
lhau	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		20-76
lhaux	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		20-77
lhax	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)).		20-78
lhbrx	RT, RA, RB	Load halfword from EA = (RA|0) + (RB), then reverse byte order and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA+1,1) || MS(EA,1).		20-79
lhz	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		20-80

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lhzu	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		20-81
lhzux	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		20-82
lhzx	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		20-83
li	RT, IM	Load immediate. (RT) ← EXTS(IM) *Extended mnemonic for* **addi RT,0,value**		20-9
lis	RT, IM	Load immediate shifted. (RT) ← (IM || $^{16}0$) *Extended mnemonic for* **addis RT,0,value**		20-12
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).		20-84
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}.		20-85
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.		20-87
lwarx	RT, RA, RB	Load word from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Set the Reservation bit.		20-89
lwbrx	RT, RA, RB	Load word 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).		20-90

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lwz	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4).		20-91
lwzu	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		20-92
lwzux	RT, RA, RB	Load word from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		20-93
lwzx	RT, RA, RB	Load word from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4).		20-94
macchw	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31}$ + (RT) (RT) ← $temp_{1:32}$		20-95
macchw.			CR[CR0]	
macchwo			XER[SO, OV]	
macchwo.			CR[CR0] XER[SO, OV]	
macchws	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31}$ + (RT) if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then (RT) ← $(RT_0 \| {}^{31}(\neg RT_0))$ else (RT) ← $temp_{1:32}$		20-96
macchws.			CR[CR0]	
macchwso			XER[SO, OV]	
macchwso.			CR[CR0] XER[SO, OV]	
macchwsu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31}$ + (RT) (RT) ← $(temp_{1:32} \vee {}^{32}temp_0)$		20-97
macchwsu.			CR[CR0]	
macchwsuo			XER[SO, OV]	
macchwsuo.			CR[CR0] XER[SO, OV]	
macchwu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31}$ + (RT) (RT) ← $temp_{1:32}$		20-98
macchwu.			CR[CR0]	
macchwuo			XER[SO, OV]	
macchwuo.			CR[CR0] XER[SO, OV]	
machhw	RT, RA, RB	$prod_{0:15} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31}$ + (RT) (RT) ← $temp_{1:32}$		20-99
machhw.			CR[CR0]	
machhwo			XER[SO, OV]	
machhwo.			CR[CR0] XER[SO, OV]	
machhws	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{0:15}$ x $(RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31}$ + (RT) if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then (RT) ← $(RT_0 \| {}^{31}(\neg RT_0))$ else (RT) ← $temp_{1:32}$		20-100
machhws.			CR[CR0]	
machhwso			XER[SO, OV]	
machhwso.			CR[CR0] XER[SO, OV]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
machhwsu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_0)$		20-101
machhwsu.			CR[CR0]	
machhwsuo			XER[SO, OV]	
machhwsuo.			CR[CR0] XER[SO, OV]	
machhwu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-102
machhwu.			CR[CR0]	
machhwuo			XER[SO, OV]	
machhwuo.			CR[CR0] XER[SO, OV]	
maclhw	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-103
maclhw.			CR[CR0]	
maclhwo			XER[SO, OV]	
maclhwo.			CR[CR0] XER[SO, OV]	
maclhws	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| {}^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-104
maclhws.			CR[CR0]	
maclhwso			XER[SO, OV]	
maclhwso.			CR[CR0] XER[SO, OV]	
maclhwsu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_0)$		20-105
maclhwsu.			CR[CR0]	
maclhwsuo			XER[SO, OV]	
maclhwsuo.			CR[CR0] XER[SO, OV]	
maclhwu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-106
maclhwu.			CR[CR0]	
maclhwuo			XER[SO, OV]	
maclhwuo.			CR[CR0] XER[SO, OV]	
mcrf	BF, BFA	Move CR field, (CR[CRn]) $\leftarrow$ (CR[CRm]) where m $\leftarrow$ BFA and n $\leftarrow$ BF.		20-107
mcrxr	BF	Move XER[0:3] into field CRn, where n$\leftarrow$BF. CR[CRn] $\leftarrow$ (XER[SO, OV, CA]). (XER[SO, OV, CA]) $\leftarrow$ ^{3}0.		20-108
mfcr	RT	Move from CR to RT, (RT) $\leftarrow$ (CR).		20-109
mfdcr	RT, DCRN	Move from DCR to RT, (RT) $\leftarrow$ (DCR(DCRN)).		20-110
mfmsr	RT	Move from MSR to RT, (RT) $\leftarrow$ (MSR).		20-111

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mfccr0 **mfctr** **mfdac1** **mfdac2** **mfdear** **mfdbcr0** **mfdbcr1** **mfdbsr** **mfdccr** **mfdcwr** **mfdvc1** **mfdvc2** **mfesr** **mfevpr** **mfiac1** **mfiac2** **mfiac3** **mfiac4** **mficcr** **mficdbdr** **mflr** **mfpid** **mfpit** **mfpvr** **mfsgr** **mfsler** **mfsprg0** **mfsprg1** **mfsprg2** **mfsprg3** **mfsprg4** **mfsprg5** **mfsprg6** **mfsprg7** **mfsrr0** **mfsrr1** **mfsrr2** **mfsrr3** **mfsu0r** **mftcr** **mftsr** **mfxer** **mfzpr**	RT	Move from special purpose register (SPR) SPRN. *Extended mnemonic for* **mfspr RT,SPRN** See Table 21.5, “Special Purpose Registers,” on page 21-1 for listing of valid SPRN values.		20-112
mfspr	RT, SPRN	Move from SPR to RT, (RT) ← (SPR(SPRN)).		20-112
mftb	RT, TBRN	Move from TBR to RT, (RT) ← (TBR(TBRN)).		20-114
mftb	RT	Move the contents of TBL into RT, (RT) ← (TBL) *Extended mnemonic for* **mftb RT,TBL**		20-114

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mftbu	RT	Move the contents of TBU into RT, (RT) $\leftarrow$ (TBU) *Extended mnemonic for* **mftb RT,TBU**		20-114
mr	RT, RS	Move register. (RT) $\leftarrow$ (RS) *Extended mnemonic for* **or RT,RS,RS**		20-140
mr.		*Extended mnemonic for* **or. RT,RS,RS**	CR[CR0]	
mtcr	RS	Move to Condition Register. *Extended mnemonic for* **mtcrf 0xFF,RS**		20-116
mtcrf	FXM, RS	Move some or all of the contents of RS into CR as specified by FXM field, mask $\leftarrow$ $^4(FXM_0)$ || $^4(FXM_1)$ || ... || $^4(FXM_6)$ || $^4(FXM_7)$. (CR)$\leftarrow$((RS) $\wedge$ mask) $\vee$ (CR) $\wedge$ $\neg$mask).		20-116
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) $\leftarrow$ (RS).		20-117
mtmsr	RS	Move to MSR from RS, (MSR) $\leftarrow$ (RS).		20-118

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mtccr0 **mtctr** **mtdac1** **mtdac2** **mtdbcr0** **mtdbcr1** **mtdbsr** **mtdccr** **mtdear** **mtdcwr** **mtdvc1** **mtdvc2** **mtesr** **mtevpr** **mtiac1** **mtiac2** **mtiac3** **mtiac4** **mticcr** **mticdbdr** **mtlr** **mtpid** **mtpit** **mtpvr** **mtsgr** **mtsler** **mtsprg0** **mtsprg1** **mtsprg2** **mtsprg3** **mtsprg4** **mtsprg5** **mtsprg6** **mtsprg7** **mtsrr0** **mtsrr1** **mtsrr2** **mtsrr3** **mtsu0r** **mttbl** **mttbu** **mttcr** **mttsr** **mtxer** **mtzpr**	RS	Move to SPR SPRN. *Extended mnemonic for* **mtspr SPRN,RS** See Table 21.5, “Special Purpose Registers,” on page 21-1 for listing of valid SPRN values.		20-119
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) ← (RS).		20-119
mulchw	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ signed		20-121
mulchw.			CR[CR0]	
mulchwu	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{0:15}$ unsigned		20-122
mulchwu.			CR[CR0]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mulhhw	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{0:15}$ x $(RB)_{0:15}$ signed		20-123
mulhhw.			CR[CR0]	
mulhhwu	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{0:15}$ x $(RB)_{0:15}$ unsigned		20-124
mulhhwu.			CR[CR0]	
mullhw	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{16:31}$ x $(RB)_{16:31}$ signed		20-127
mullhw.			CR[CR0]	
mullhwu	RT, RA, RB	$(RT)_{16:31} \leftarrow (RA)_{16:31}$ x $(RB)_{16:31}$ unsigned		20-128
mullhwu.			CR[CR0]	
mulhw	RT, RA, RB	Multiply (RA) and (RB), signed. Place high-order result in RT. $prod_{0:63} \leftarrow (RA) \times (RB)$ (signed). $(RT) \leftarrow prod_{0:31.}$		20-125
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). $(RT) \leftarrow prod_{0:31.}$		20-126
mulhwu.			CR[CR0]	
mulli	RT, RA, IM	Multiply (RA) and IM, signed. Place low-order result in RT. $prod_{0:47} \leftarrow (RA) \times IM$ (signed) $(RT) \leftarrow prod_{16:47}$		20-129
mullw	RT, RA, RB	Multiply (RA) and (RB), signed. Place low-order result in RT. $prod_{0:63} \leftarrow (RA) \times (RB)$ (signed). $(RT) \leftarrow prod_{32:63.}$		20-130
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.		20-131
nand.			CR[CR0]	
neg	RT, RA	Negative (twos complement) of RA. $(RT) \leftarrow \neg(RA) + 1$		20-132
neg.			CR[CR0]	
nego			XER[SO, OV]	
nego.			CR[CR0] XER[SO, OV]	
nmacchw	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{16:31}$ x $(RB)_{0:15})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-133
nmacchw.			CR[CR0]	
nmacchwo			XER[SO, OV]	
nmacchwo.			CR[CR0] XER[SO, OV]	
nmacchws	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{16:31}$ x $(RB)_{0:15})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ if $((nprod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| {}^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-134
nmacchws.			CR[CR0]	
nmacchwso			XER[SO, OV]	
nmacchwso.			CR[CR0] XER[SO, OV]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
nmachhw	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{0:15} \times (RB)_{0:15})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-135
nmachhw.			CR[CR0]	
nmachhwo			XER[SO, OV]	
nmachhwo.			CR[CR0] XER[SO, OV]	
nop		Preferred no-op, triggers optimizations based on no-ops. *Extended mnemonic for* **ori 0,0,0**		20-142
nor	RA, RS, RB	NOR (RS) with (RB). Place result in RA.		20-139
nor.			CR[CR0]	
not	RA, RS	Complement register. $(RA) \leftarrow \neg(RS)$ *Extended mnemonic for* **nor RA,RS,RS**		20-139
not.		*Extended mnemonic for* **nor. RA,RS,RS**	CR[CR0]	
or	RA, RS, RB	OR (RS) with (RB). Place result in RA.		20-140
or.			CR[CR0]	
orc	RA, RS, RB	OR (RS) with $\neg$(RB). Place result in RA.		20-141
orc.			CR[CR0]	
ori	RA, RS, IM	OR (RS) with ($^{16}0 \parallel$ IM). Place result in RA.		20-142
oris	RA, RS, IM	OR (RS) with (IM $\parallel$ $^{16}0$). Place result in RA.		20-143
rfci		Return from critical interrupt $(PC) \leftarrow (SRR2)$. $(MSR) \leftarrow (SRR3)$.		20-144
rfi		Return from interrupt. $(PC) \leftarrow (SRR0)$. $(MSR) \leftarrow (SRR1)$.		20-145
rlwimi	RA, RS, SH, MB, ME	Rotate left word immediate, then insert according to mask. $r \leftarrow ROTL((RS), SH)$ $m \leftarrow MASK(MB, ME)$ $(RA) \leftarrow (r \wedge m) \vee ((RA) \wedge \neg m)$		20-146
rlwimi.			CR[CR0]	
rlwinm	RA, RS, SH, MB, ME	Rotate left word immediate, then AND with mask. $r \leftarrow ROTL((RS), SH)$ $m \leftarrow MASK(MB, ME)$ $(RA) \leftarrow (r \wedge m)$		20-147
rlwinm.			CR[CR0]	
rlwnm	RA, RS, RB, MB, ME	Rotate left word, then AND with mask. $r \leftarrow ROTL((RS), (RB)_{27:31})$ $m \leftarrow MASK(MB, ME)$ $(RA) \leftarrow (r \wedge m)$		20-150
rlwnm.			CR[CR0]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
rotlw	RA, RS, RB	Rotate left. (RA) ← ROTL((RS), $(RB)_{27:31}$) *Extended mnemonic for* **rlwnm RA,RS,RB,0,31**		20-150
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**		20-147
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**		20-147
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		20-151
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.		20-152
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**		20-147
slwi.		*Extended mnemonic for* **rlwinm. RA,RS,n,0,31–n**	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).		20-153
sraw.			CR[CR0]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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).		20-154
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.		20-155
srw.			CR[CR0]	
srwi	RA, RS, n	Shift right immediate. (n < 32) $(RA)_{n:31}$ ← $(RS)_{0:31–n}$ $(RA)_{0:n–1}$ ← ^{n}0 *Extended mnemonic for* **rlwinm RA,RS,32–n,n,31**		20-147
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).		20-156
stbu	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		20-157
stbux	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		20-158
stbx	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB).		20-159
sth	RS, D(RA)	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D).		20-160
sthbrx	RS, RA, RB	Store halfword $(RS)_{16:31}$ byte-reversed in memory at EA = (RA|0) + (RB). MS(EA, 2) ← $(RS)_{24:31}$ ‖ $(RS)_{16:23}$		20-161
sthu	RS, D(RA)	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		20-162
sthux	RS, RA, RB	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		20-163
sthx	RS, RA, RB	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB).		20-164

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
stmw	RS, D(RA)	Store consecutive words from RS through GPR(31) in memory starting at EA = (RA|0) + EXTS(D).		20-165
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).		20-166
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).		20-167
stw	RS, D(RA)	Store word (RS) in memory at EA = (RA|0) + EXTS(D).		20-169
stwbrx	RS, RA, RB	Store word (RS) byte-reversed in memory at EA = (RA|0) + (RB). MS(EA, 4) $\leftarrow$ $(RS)_{24:31}$ || $(RS)_{16:23}$ || $(RS)_{8:15}$ || $(RS)_{0:7}$		20-170

stwcx.	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB) only if reservation bit is set.
		if RESERVE = 1 then
		MS(EA, 4) $\leftarrow$ (RS)
		RESERVE $\leftarrow$ 0
		(CR[CR0]) $\leftarrow$ $^{2}0$ || 1 || XER_{so}
		else
		(CR[CR0]) $\leftarrow$ $^{2}0$ || 0 || $XER_{so.}$
				20-171

stwu	RS, D(RA)	Store word (RS) in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) $\leftarrow$ EA.		20-173
stwux	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB). Update the base address, (RA) $\leftarrow$ EA.		20-174
stwx	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB).		20-175
sub	RT, RA, RB	Subtract (RB) from (RA). (RT) $\leftarrow$ $\neg$(RB) + (RA) + 1. *Extended mnemonic for* **subf RT,RB,RA**		20-176
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]	

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-177
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.		20-176
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].		20-177
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].		20-178
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].		20-179
subfme	RT, RA, RB	Subtract (RA) from (–1) with carry-in. (RT) ← ¬(RA) + (–1) + XER[CA]. Place carry-out in XER[CA].		20-180
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].		20-181
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**		20-9

Table A-1. PPC405CR Instruction Syntax Summary (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**		20-10
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**	CR[CR0]	20-11
subis	RT, RA, IM	Subtract (IM || $^{16}0$) from (RA|0). Place result in RT. *Extended mnemonic for* **addis RT,RA,–IM**		20-12
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** will have completed.		20-182
tlbia		All TLB entries 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 TLB fields unmodified.		20-183
tlbre	RT, RA,WS	If WS = 0: Load TLBHI of the selected TLB entry into RT. Load PID 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)]		20-184
tlbrehi	RT, RA	Load TLBHI of the selected TLB entry into RT. Load PID 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**		20-184
tlbrelo	RT, RA	Load TLBLO of the selected TLB entry into RT. (RT) ← TLBLO[(RA)] *Extended mnemonic for* **tlbre RT,RA,1**		20-184

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbsx	RT,RA,RB	Search the TLB for a valid entry that translates the EA. EA = (RA|0) + (RB). If found, (RT) ← Index of TLB entry. If not found, (RT) Undefined.		20-186
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 the PPC405CR, **tlbsync** is a no-op.		20-187
tlbwe	RS, RA,WS	If WS = 0: Write TLBHI 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)		20-188
tlbwehi	RS, RA	Write TLBHI 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**		20-188
tlbwelo	RS, RA	Write TLBLO of the selected TLB entry from RS. TLBLO[(RA)] ← (RS) *Extended mnemonic for* **tlbwe RS,RA,1**		20-188

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
trap		Trap unconditionally. *Extended mnemonic for* **tw 31,0,0**		20-190
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.		20-190

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tweqi	RA, IM	Trap if (RA) equal to EXTS(IM). *Extended mnemonic for* **wi 4,RA,IM**		20-193
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* **wi 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.		20-193
wrtee	RS	Write value of RS_{16} to MSR[EE].		20-196
wrteei	E	Write value of E to MSR[EE].		20-197

Table A-1. PPC405CR Instruction Syntax Summary (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
xor	RA, RS, RB	XOR (RS) with (RB). Place result in RA.		20-198
xor.			CR[CR0]	
xori	RA, RS, IM	XOR (RS) with ($^{16}0$ || IM). Place result in RA.		20-199
xoris	RA, RS, IM	XOR (RS) with (IM || $^{16}0$). Place result in RA.		20-200

A.2 Instructions Sorted by Opcode

All instructions are four bytes long and word aligned. All instructions have a primary opcode field (shown as field OPCD in Figure A-1 through Figure A-9, beginning on page A-44) in bits 0:5. Some instructions also have a secondary opcode field (shown as field XO in Figure A-1 through Figure A-9). PPC405CR instructions, sorted by primary and secondary opcode, are listed in Table A-2.

The “Form” indicated in the table refers to the arrangement of valid field combinations within the four-byte instruction. See “Instruction Formats,” on page A-41, for the field layouts of 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 the table below.

Table A-2. PPC405CR Instructions by Opcode

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
3		D	**twi**	TO, RA, IM	20-193
4	8	X	**mulhhwu**	RT, RA, RB	20-124
			mulhhwu.		
4	12 (524)	XO	**machhwu**	RT, RA, RB	20-102
			machhwu.		
			machhwuo		
			machhwuo.		
4	40	X	**mulhhw**	RT, RA, RB	20-123
			mulhhw.		
4	44 (556)	XO	**machhw**	RT, RA, RB	20-99
			machhw.		
			machhwo		
			machhwo.		
4	46 (558)	XO	**nmachhw**	RT, RA, RB	20-135
			nmachhw.		
			nmachhwo		
			nmachhwo		

Table A-2. PPC405CR Instructions by Opcode (continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
4	76 (588)	XO	**machhwsu**	RT, RA, RB	20-101
			machhwsu.		
			machhwsuo		
			machhwsuo.		
4	108 (620)	XO	**machhws**	RT, RA, RB	20-100
			machhws.		
			machhwso		
			machhwso.		
4	110 (622)	XO	**nmachhws**	RT, RA, RB	20-136
			nmachhws.		
			nmachhwso		
			nmachhwso.		
4	136	X	**mulchwu**	RT, RA, RB	20-122
			mulchwu.		
4	140 (652)	XO	**macchwu**	RT, RA, RB	20-98
			macchwu.		
			macchwuo		
			machhwuo.		
4	168	X	**mulchw**	RT, RA, RB	20-121
			mulchw.		
4	172 (684)	XO	**macchw**	RT, RA, RB	20-95
			macchw.		
			macchwo		
			macchwo.		
4	174 (686)	XO	**nmacchw**	RT, RA, RB	20-133
			nmacchw.		
			nmacchwo		
			nmacchwo.		
4	204 (716)	XO	**macchwsu**	RT, RA, RB	20-101
			macchwsu.		
			macchwsuo		
			macchwsuo.		
4	236 (748)	XO	**macchws**	RT, RA, RB	20-96
			macchws.		
			macchwso		
			macchwso.		
4	238 (750)	XO	**nmacchws**	RT, RA, RB	20-134
			nmacchws.		
			nmacchwso		
			nmacchwso.		

Table A-2. PPC405CR Instructions by Opcode (continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
4	392	X	**mullhwu**	RT, RA, RB	20-128
			mullhwu.		
4	396 (908)	XO	**maclhwu**	RT, RA, RB	20-106
			maclhwu.		
			maclhwuo		
			maclhwuo.		
4	424	X	**mullhw**	RT, RA, RB	20-126
			mullhw.		
4	428 (940)	XO	**maclhw**	RT, RA, RB	20-103
			maclhw.		
			maclhwo		
			maclhwo.		
4	430 (942)	XO	**nmaclhw**	RT, RA, RB	20-137
			nmaclhw.		
			nmaclhwo		
			nmaclhwo.		
4	492 (972)	XO	**maclhws**	RT, RA, RB	20-104
			maclhws.		
			maclhwso		
			maclhwso.		
4	460 (1004)	XO	**maclhwsu**	RT, RA, RB	20-105
			maclhwsu.		
			maclhwsuo		
			maclhwsuo.		
4	494 (1006)	XO	**nmaclhws**	RT, RA, RB	20-138
			nmaclhws.		
			nmaclhwso		
			nmaclhwso.		
7		D	**mulli**	RT, RA, IM	20-129
8		D	**subfic**	RT, RA, IM	20-179
10		D	**cmpli**	BF, 0, RA, IM	20-37
11		D	**cmpi**	BF, 0, RA, IM	20-35
12		D	**addic**	RT, RA, IM	20-10
13		D	**addic.**	RT, RA, IM	20-11
14		D	**addi**	RT, RA, IM	20-9
15		D	**addis**	RT, RA, IM	20-12
16		B	**bc**	BO, BI, target	20-20
			bca		
			bcl		
			bcla		
17		SC	**sc**		20-151

Table A-2. PPC405CR Instructions by Opcode (continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
18		I	**b**	target	20-19
			ba		
			bl		
			bla		
19	0	XL	**mcrf**	BF, BFA	20-107
19	16	XL	**bclr**	BO, BI	20-30
			bclrl		
19	33	XL	**crnor**	BT, BA, BB	20-43
19	50	XL	**rfi**		20-145
19	51	XL	**rfci**		20-144
19	129	XL	**crandc**	BT, BA, BB	20-40
19	150	XL	**isync**		20-70
19	193	XL	**crxor**	BT, BA, BB	20-46
19	225	XL	**crnand**	BT, BA, BB	20-42
19	257	XL	**crand**	BT, BA, BB	20-39
19	289	XL	**creqv**	BT, BA, BB	20-41
19	417	XL	**crorc**	BT, BA, BB	20-45
19	449	XL	**cror**	BT, BA, BB	20-44
19	528	XL	**bcctr**	BO, BI	20-26
			bcctrl		
20		M	**rlwimi**	RA, RS, SH, MB, ME	20-146
			rlwimi.		
21		M	**rlwinm**	RA, RS, SH, MB, ME	20-147
			rlwinm.		
23		M	**rlwnm**	RA, RS, RB, MB, ME	20-150
			rlwnm.		
24		D	**ori**	RA, RS, IM	20-142
25		D	**oris**	RA, RS, IM	20-143
26		D	**xori**	RA, RS, IM	20-199
27		D	**xoris**	RA, RS, IM	20-200
28		D	**andi.**	RA, RS, IM	20-17
29		D	**andis.**	RA, RS, IM	20-18
31	0	X	**cmp**	BF, 0, RA, RB	20-34
31	4	X	**tw**	TO, RA, RB	20-190
31	8 (520)	XO	**subfc**	RT, RA, RB	20-177
			subfc.		
			subfco		
			subfco.		

Table A-2. PPC405CR Instructions by Opcode (continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	10 (522)	XO	**addc**	RT, RA, RB	20-7
			addc.		
			addco		
			addco.		
31	11	XO	**mulhwu**	RT, RA, RB	20-126
			mulhwu.		
31	19	X	**mfcr**	RT	20-109
31	20	X	**lwarx**	RT, RA, RB	20-89
31	23	X	**lwzx**	RT, RA, RB	20-94
31	24	X	**slw**	RA, RS, RB	20-152
			slw.		
31	26	X	**cntlzw**	RA, RS	20-38
			cntlzw.		
31	28	X	**and**	RA, RS, RB	20-15
			and.		
31	32	X	**cmpl**	BF, 0, RA, RB	20-36
31	40 (552)	XO	**subf**	RT, RA, RB	20-176
			subf.		
			subfo		
			subfo.		
31	54	X	**dcbst**	RA, RB	20-53
31	55	X	**lwzux**	RT, RA, RB	20-93
31	60	X	**andc**	RA, RS, RB	20-16
			andc.		
31	75	XO	**mulhw**	RT, RA, RB	20-125
			mulhw.		
31	83	X	**mfmsr**	RT	20-111
31	86	X	**dcbf**	RA, RB	20-49
31	87	X	**lbzx**	RT, RA, RB	20-74
31	104 (616)	XO	**neg**	RT, RA	20-132
			neg.		
			nego		
			nego.		
31	119	X	**lbzux**	RT, RA, RB	20-73
31	124	X	**nor**	RA, RS, RB	20-139
			nor.		
31	131	X	**wrtee**	RS	20-196
31	136 (648)	XO	**subfe**	RT, RA, RB	20-178
			subfe.		
			subfeo		
			subfeo.		

Table A-2. PPC405CR Instructions by Opcode (continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	138 (650)	XO	**adde**	RT, RA, RB	20-8
			adde.		
			addeo		
			addeo.		
31	144	XFX	**mtcrf**	FXM, RS	20-116
31	146	X	**mtmsr**	RS	20-118
31	150	X	**stwcx.**	RS, RA, RB	20-171
31	151	X	**stwx**	RS, RA, RB	20-175
31	163	X	**wrteei**	E	20-197
31	183	X	**stwux**	RS, RA, RB	20-174
31	200 (712)	XO	**subfze**	RT, RA, RB	20-181
			subfze.		
			subfzeo		
			subfzeo.		
31	202 (714)	XO	**addze**	RT, RA	20-14
			addze.		
			addzeo		
			addzeo.		
31	215	X	**stbx**	RS, RA, RB	20-159
31	232 (744)	XO	**subfme**	RT, RA, RB	20-180
			subfme.		
			subfmeo		
			subfmeo.		
31	234 (746)	XO	**addme**	RT, RA	20-13
			addme.		
			addmeo		
			addmeo.		
31	235 (747)	XO	**mullw**	RT, RA, RB	20-130
			mullw.		
			mullwo		
			mullwo.		
31	246	X	**dcbtst**	RA,RB	20-53
31	247	X	**stbux**	RS, RA, RB	20-157
31	262	X	**icbt**	RA, RB	20-66
31	266 (778)	XO	**add**	RT, RA, RB	20-6
			add.		
			addo		
			addo.		
31	278	X	**dcbt**	RA, RB	20-51
31	279	X	**lhzx**	RT, RA, RB	20-83

Table A-2. PPC405CR Instructions by Opcode (continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	284	X	**eqv**	RA, RS, RB	20-62
			eqv.		
31	311	X	**lhzux**	RT, RA, RB	20-82
31	316	X	**xor**	RA, RS, RB	20-198
			xor.		
31	323	XFX	**mfdcr**	RT, DCRN	20-110
31	339	XFX	**mfspr**	RT, SPRN	20-112
31	343	X	**lhax**	RT, RA, RB	20-78
31	370	X	**tlbia**		20-183
31	371	XFX	**mftb**	RT, TBRN	20-114
31	375	X	**lhaux**	RT, RA, RB	20-77
31	407	X	**sthx**	RS, RA, RB	20-164
31	412	X	**orc**	RA, RS, RB	20-141
			orc.		
31	439	X	**sthux**	RS, RA, RB	20-163
31	444	X	**or**	RA, RS, RB	20-140
			or.		
31	451	XFX	**mtdcr**	DCRN, RS	20-117
31	454	X	**dccci**	RA, RB	20-56
31	459 (971)	XO	**divwu**	RT, RA, RB	20-60
			divwu.		
			divwuo		
			divwuo.		
31	467	XFX	**mtspr**	SPRN, RS	20-119
31	470	X	**dcbi**	RA, RB	20-50
31	476	X	**nand**	RA, RS, RB	20-131
			nand.		
31	486	X	**dcread**	RT, RA, RB	20-57
31	491 (1003)	XO	**divw**	RT, RA, RB	20-59
			divw.		
			divwo		
			divwo.		
31	512	X	**mcrxr**	BF	20-108
31	533	X	**lswx**	RT, RA, RB	20-87
31	534	X	**lwbrx**	RT, RA, RB	20-90
31	536	X	**srw**	RA, RS, RB	20-155
			srw.		
31	566	X	**tlbsync**		20-187
31	597	X	**lswi**	RT, RA, NB	20-85
31	598	X	**sync**		20-182
31	661	X	**stswx**	RS, RA, RB	20-167

Table A-2. PPC405CR Instructions by Opcode (continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	662	X	**stwbrx**	RS, RA, RB	20-170
31	725	X	**stswi**	RS, RA, NB	20-166
31	758	X	**dcba**	RA, RB	20-47
31	790	X	**lhbrx**	RT, RA, RB	20-79
31	792	X	**sraw**	RA, RS, RB	20-153
			sraw.		
31	824	X	**srawi**	RA, RS, SH	20-154
			srawi.		
31	854	X	**eieio**		20-61
31	914	X	**tlbsx**	RT,RA,RB	20-186
			tlbsx.		
31	918	X	**sthbrx**	RS, RA, RB	20-161
31	922	X	**extsh**	RA, RS	20-64
			extsh.		
31	946	X	**tlbre**	RT, RA,WS	20-184
31	954	X	**extsb**	RA, RS	20-63
			extsb.		
31	966	X	**iccci**	RA, RB	20-67
31	978	X	**tlbwe**	RS, RA,WS	20-188
31	982	X	**icbi**	RA, RB	20-65
31	998	X	**icread**	RA, RB	20-68
31	1014	X	**dcbz**	RA, RB	20-54
32		D	**lwz**	RT, D(RA)	20-91
33		D	**lwzu**	RT, D(RA)	20-92
34		D	**lbz**	RT, D(RA)	20-71
35		D	**lbzu**	RT, D(RA)	20-72
36		D	**stw**	RS, D(RA)	20-169
37		D	**stwu**	RS, D(RA)	20-173
38		D	**stb**	RS, D(RA)	20-156
39		D	**stbu**	RS, D(RA)	20-157
40		D	**lhz**	RT, D(RA)	20-80
41		D	**lhzu**	RT, D(RA)	20-81
42		D	**lha**	RT, D(RA)	20-75
43		D	**lhau**	RT, D(RA)	20-76
44		D	**sth**	RS, D(RA)	20-160
45		D	**sthu**	RS, D(RA)	20-162
46		D	**lmw**	RT, D(RA)	20-84
47		D	**stmw**	RS, D(RA)	20-165

A.3 Instruction Formats

Instructions are four bytes long. Instruction addresses are always word-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions have an extended opcode in another field. 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 GPR selectors and immediate values, that can vary from execution to execution. The instruction format diagrams specify the operands in the variable fields.

- Reserved

  Bits in reserved fields should be set to 0. In the instruction format diagrams, /, //, or /// indicate reserved fields.

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; its result is architecturally undefined. The PPC405CR executes all invalid instruction forms without causing an illegal instruction exception.

A.3.1 Instruction Fields

PPC405CR instructions contain various combinations of the following fields, as indicated in the instruction format diagrams that follow the field definitions. Numbers, enclosed in parentheses, that follow the field names indicate 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 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 “BO Field on Conditional Branches” on page 3-31.

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 “Condition Register (CR)” on page 3-11 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 instruction formats (also called “forms”) illustrated in Figure A-1 through Figure A-9 are valid combinations of instruction fields. Table A-2 on page A-33 indicates which “form” is utilized by each PPC405CR opcode. Fields indicated by slashes (/, //, or ///) are reserved. The figures are adapted from the PowerPC User Instruction Set Architecture.

A.3.2.1 I-Form

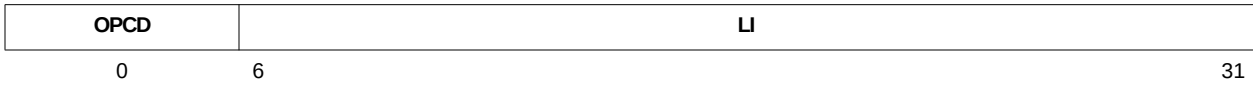


Figure A-1. I Instruction Format

A.3.2.2 B-Form

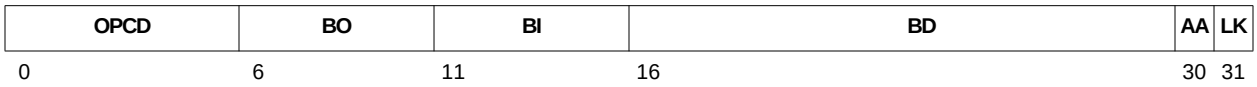


Figure A-2. B Instruction Format

A.3.2.3 SC-Form

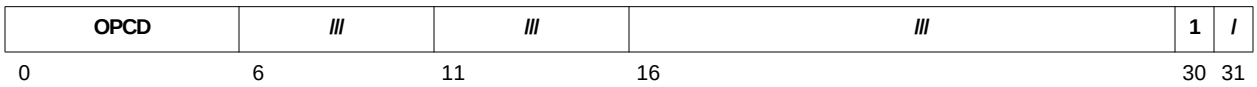


Figure A-3. SC Instruction Format

A.3.2.4 D-Form

OPCD	RT		RA	D
OPCD	RS		RA	SI
OPCD	RS		RA	D
OPCD	RS		RA	UI
OPCD	BF / L		RA	SI
OPCD	BF / L		RA	UI
OPCD	TO		RA	SI
0	6	11	16	31

Figure A-4. D Instruction Format

A.3.2.5 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	Rc
OPCD	BF //	///	///	XO	/
OPCD	BF //	///	U	XO	Rc
OPCD	BF //	///	///	XO	/
OPCD	TO	RA	RB	XO	/
OPCD	BT	///	///	XO	Rc
OPCD	///	RA	RB	XO	/
OPCD	///	///	///	XO	/
OPCD	///	///	E //	XO	/

0 6 11 16 21 31

Figure A-5. X Instruction Format

A.3.2.6 XL-Form

OPCD	BT	BA	BB	XO	/
OPCD	BC	BI	///	XO	LK
OPCD	BF //	BFA //	///	XO	/
OPCD	///	///	///	XO	/

0 6 11 16 21 31

Figure A-6. XL Instruction Format

A.3.2.7 XFX-Form

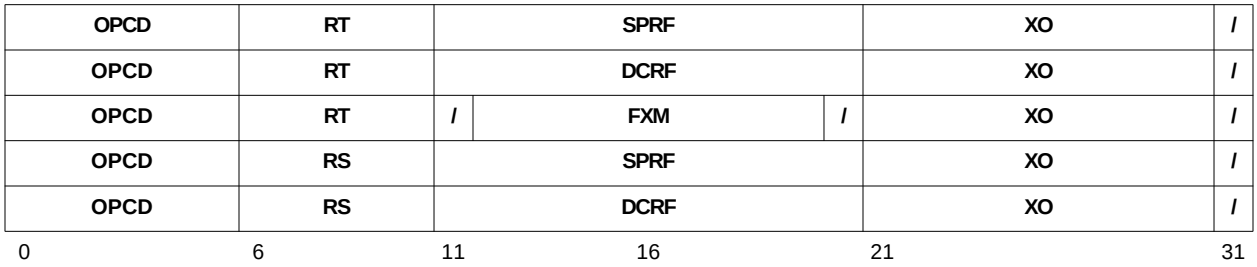


Figure A-7. XFX Instruction Format

A.3.2.8 X0-Form

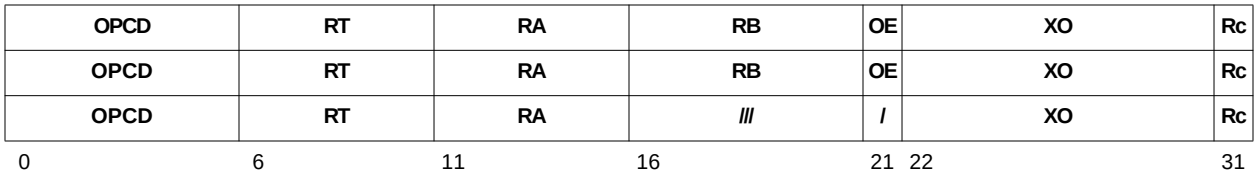


Figure A-8. XO Instruction Format

A.3.2.9 M-Form

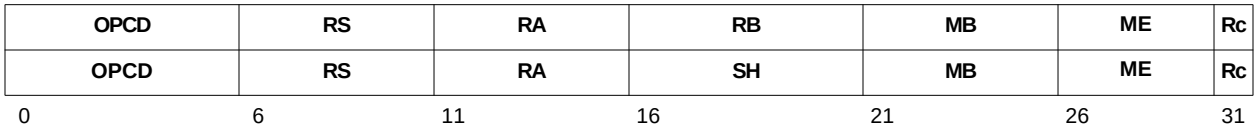


Figure A-9. M Instruction Format

Appendix B. Instructions by Category

Chapter 20, “Instruction Set,” contains detailed descriptions of the instructions, their operands, and notation.

Table B-1 summarizes the instruction categories in the PPC405CR instruction set. The instructions within each category are listed in subsequent tables.

Table B-1. PPC405CR 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, multiply accumulate
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.1 Implementation-Specific Instructions

To meet the functional requirements of processors for embedded systems and real-time applications, the PPC405CR defines the implementation-specific instructions summarized in Table B-2.

Table B-2. Implementation-specific Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
dccci	RA, RB	Invalidate the data cache congruence class associated with the effective address (EA) (RA|0) + (RB).		20-56
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the EA (RA|0) + (RB). Place the results in RT.		20-57
iccci	RA, RB	Invalidate instruction cache.		20-67
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the EA (RA|0) + (RB). Place the results in ICDBDR.		20-68

Table B-2. Implementation-specific Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
macchw	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-95
macchw.			CR[CR0]	
macchwo			XER[SO, OV]	
macchwo.			CR[CR0] XER[SO, OV]	
macchws	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| \| {}^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-96
macchws.			CR[CR0]	
macchwso			XER[SO, OV]	
macchwso.			CR[CR0] XER[SO, OV]	
macchwsu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_0)$		20-97
macchwsu.			CR[CR0]	
macchwsuo			XER[SO, OV]	
macchwsuo.			CR[CR0] XER[SO, OV]	
macchwu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-98
macchwu.			CR[CR0]	
macchwuo			XER[SO, OV]	
macchwuo.			CR[CR0] XER[SO, OV]	
machhw	RT, RA, RB	$prod_{0:15} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-99
machhw.			CR[CR0]	
machhwo			XER[SO, OV]	
machhwo.			CR[CR0] XER[SO, OV]	
machhws	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| \| {}^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-100
machhws.			CR[CR0]	
machhwso			XER[SO, OV]	
machhwso.			CR[CR0] XER[SO, OV]	
machhwsu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_0)$		20-101
machhwsu.			CR[CR0]	
machhwsuo			XER[SO, OV]	
machhwsuo.			CR[CR0] XER[SO, OV]	

Table B-2. Implementation-specific Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
machhwu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-102
machhwu.			CR[CR0]	
machhwuo			XER[SO, OV]	
machhwuo.			CR[CR0] XER[SO, OV]	
maclhw	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-103
maclhw.			CR[CR0]	
maclhwo			XER[SO, OV]	
maclhwo.			CR[CR0] XER[SO, OV]	
maclhws	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ if $((prod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| \| {}^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-104
maclhws.			CR[CR0]	
maclhwso			XER[SO, OV]	
maclhwso.			CR[CR0] XER[SO, OV]	
maclhwsu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow (temp_{1:32} \vee {}^{32}temp_0)$		20-105
maclhwsu.			CR[CR0]	
maclhwsuo			XER[SO, OV]	
maclhwsuo.			CR[CR0] XER[SO, OV]	
maclhwu	RT, RA, RB	$prod_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ unsigned $temp_{0:32} \leftarrow prod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-106
maclhwu.			CR[CR0]	
maclhwuo			XER[SO, OV]	
maclhwuo.			CR[CR0] XER[SO, OV]	
mulchw	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ signed		20-121
mulchw.			CR[CR0]	
mulchwu	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{0:15}$ unsigned		20-122
mulchwu.			CR[CR0]	
mulhhw	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ signed		20-123
mulhhw.			CR[CR0]	
mulhhwu	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{0:15} \times (RB)_{0:15}$ unsigned		20-124
mulhhwu.			CR[CR0]	
mullhw	RT, RA, RB	$(RT)_{0:31} \leftarrow (RA)_{16:31} \times (RB)_{16:31}$ signed		20-127
mullhw.			CR[CR0]	

Table B-2. Implementation-specific Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mullhwu	RT, RA, RB	$(RT)_{16:31} \leftarrow (RA)_{0:15} \times (RB)_{16:31}$ unsigned		20-128
mullhwu.			CR[CR0]	
nmacchw	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{0:15})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-133
nmacchw.			CR[CR0]	
nmacchwo			XER[SO, OV]	
nmacchwo.			CR[CR0] XER[SO, OV]	
nmacchws	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{0:15})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ if $((nprod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| \| ^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-134
nmacchws.			CR[CR0]	
nmacchwso			XER[SO, OV]	
nmacchwso.			CR[CR0] XER[SO, OV]	
nmachhw	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{0:15} \times (RB)_{0:15})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-135
nmachhw.			CR[CR0]	
nmachhwo			XER[SO, OV]	
nmachhwo.			CR[CR0] XER[SO, OV]	
nmachhws	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{0:15} \times (RB)_{0:15})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ if $((nprod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| \| ^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-136
nmachhws.			CR[CR0]	
nmachhwso			XER[SO, OV]	
nmachhwso.			CR[CR0] XER[SO, OV]	
nmaclhw	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{16:31})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ $(RT) \leftarrow temp_{1:32}$		20-137
nmaclhw.			CR[CR0]	
nmaclhwo			XER[SO, OV]	
nmaclhwo.			CR[CR0] XER[SO, OV]	
nmaclhws	RT, RA, RB	$nprod_{0:31} \leftarrow -((RA)_{16:31} \times (RB)_{16:31})$ signed $temp_{0:32} \leftarrow nprod_{0:31} + (RT)$ if $((nprod_0 = RT_0) \wedge (RT_0 \neq temp_1))$ then $(RT) \leftarrow (RT_0 \| \| ^{31}(\neg RT_0))$ else $(RT) \leftarrow temp_{1:32}$		20-138
nmaclhws.			CR[CR0]	
nmaclhwso			XER[SO, OV]	
nmaclhwso.			CR[CR0] XER[SO, OV]	

B.2 Instructions in the IBM PowerPC Embedded Environment

To meet the functional requirements of processors for embedded systems and real-time applications, the IBM PowerPC Embedded Environment defines instructions that are not part of the PowerPC Architecture.

Table B-3 summarizes the PPC405CR instructions in the PowerPC Embedded Environment.

Table B-3. Instructions in the IBM PowerPC Embedded Environment

Mnemonic	Operands	Function	Other Registers Changed	Page
dcba	RA, RB	Speculatively establish the data cache block which contains the EA (RA|0) + (RB).		20-47
dcbf	RA, RB	Flush (store, then invalidate) the data cache block which contains the EA (RA|0) + (RB).		20-49
dcbi	RA, RB	Invalidate the data cache block which contains the EA (RA|0) + (RB).		20-50
dcbst	RA, RB	Store the data cache block which contains the EA (RA|0) + (RB).		20-51
dcbt	RA, RB	Load the data cache block which contains the EA (RA|0) + (RB).		20-52
dcbtst	RA,RB	Load the data cache block which contains the EA (RA|0) + (RB).		20-53
dcbz	RA, RB	Zero the data cache block which contains the EA (RA|0) + (RB).		20-54
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.		20-61
icbi	RA, RB	Invalidate the instruction cache block which contains the EA (RA|0) + (RB).		20-65
icbt	RA, RB	Load the instruction cache block which contains the EA (RA|0) + (RB).		20-66
isync		Synchronize execution context by flushing the prefetch queue.		20-70
mfdcr	RT, DCRN	Move from DCR to RT, (RT) $\leftarrow$ (DCR(DCRN)).		20-110
mftb	RT	Move the contents of a Time Base Register (TBR) into RT, TBRN $\leftarrow$ $TBRF_{5:9}$ || $TBRF_{0:4}$ (RT) $\leftarrow$ (TBR(TBRN))		20-114
mfmsr	RT	Move from MSR to RT, (RT) $\leftarrow$ (MSR).		20-118

Table B-3. Instructions in the IBM PowerPC Embedded Environment (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mfspr	RT, SPRN	Move from SPR to RT, (RT) ← (SPR(SPRN)). Privileged for all SPRs except LR, CTR, TBHU, TBLU, and XER.		20-112
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) ← (RS).		20-117
mtmsr	RS	Move to MSR from RS, (MSR) ← (RS).		20-118
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) ← (RS). Privileged for all SPRs except LR, CTR, and XER.		20-119
rfci		Return from critical interrupt (PC) ← (SRR2). (MSR) ← (SRR3).		20-144
rfi		Return from interrupt. (PC) ← (SRR0). (MSR) ← (SRR1).		20-145
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.		20-183
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)]		20-184

Table B-3. Instructions in the IBM PowerPC Embedded Environment (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbsx	RT,RA,RB	Search the TLB array for a valid entry which translates the EA EA = (RA|0) + (RB). If found, (RT) ← Index of TLB entry. If not found, (RT) Undefined.		20-186
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 the PPC405CR, **tlbsync** is a no-op.		20-187
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)		20-188
wrtee	RS	Write value of RS_{16} to MSR[EE].		20-196
wrteei	E	Write value of E to MSR[EE].		20-197

B.3 Privileged Instructions

Table B-4 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-4. Privileged Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
dcbi	RA, RB	Invalidate the data cache block which contains the EA (RA|0) + (RB).		20-50
dccci	RA, RB	Invalidate the data cache congruence class associated with the EA (RA|0) + (RB).		20-56

Table B-4. Privileged Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the EA (RA|0) + (RB). Place the results in RT.		20-57
iccci	RA, RB	Invalidate instruction cache.		20-67
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the EA (RA|0) + (RB). Place the results in ICDBDR.		20-68
mfdcr	RT, DCRN	Move from DCR to RT, (RT) ← (DCR(DCRN)).		20-110
mfmsr	RT	Move from MSR to RT, (RT) ← (MSR).		20-118
mfspr	RT, SPRN	Move from SPR to RT, (RT) ← (SPR(SPRN)). Privileged for all SPRs except LR, CTR, TBHU, TBLU, and XER.		20-112
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) ← (RS).		20-117
mtmsr	RS	Move to MSR from RS, (MSR) ← (RS).		20-118
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) ← (RS). Privileged for all SPRs except LR, CTR, and XER.		20-119
rfci		Return from critical interrupt (PC) ← (SRR2). (MSR) ← (SRR3).		20-144
rfi		Return from interrupt. (PC) ← (SRR0). (MSR) ← (SRR1).		20-145
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)]		20-184

Table B-4. Privileged Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
tlbsx	RT,RA,RB	Search the TLB array for a valid entry which translates the EA EA = (RA|0) + (RB). If found, (RT) ← Index of TLB entry. If not found, (RT) Undefined.		20-186
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)		20-188
wrtee	RS	Write value of RS_{16} to the External Enable bit (MSR[EE]).		20-196
wrteei	E	Write value of E to the External Enable bit (MSR[EE]).		20-197

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-5 lists the extended mnemonics required for the PPC405CR.

Note for every Branch Conditional mnemonic:

Bit 4 of the BO field provides a hint about the most likely outcome of a conditional branch. (“Branch Prediction” on page 3-32 describes branch prediction). Assemblers should set BO_4 = 0 unless a specific reason exists otherwise. In the BO field values specified in the following table, BO_4 = 0 has always been assumed. The assembler must allow the programmer to specify branch prediction. To do this, the assembler will support 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 “Branch Prediction” on page 3-32).

Table B-5. Extended Mnemonics for PPC405CR

Mnemonic	Operands	Function	Other Registers Changed	Page
bctr		Branch unconditionally to address in CTR. *Extended mnemonic for* **bcctr 20,0**		20-26
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**		20-20
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**		20-30
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**		20-20
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**		20-30
bdnzflrl		*Extended mnemonic for* **bclrl 0,cr_bit**	(LR) ← CIA + 4.	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bdnzt	cr_bit, target	Decrement CTR. Branch if CTR ≠ 0 AND CR_{cr_bit} = 1. *Extended mnemonic for* **bc 8,cr_bit,target**		20-20
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**		20-30
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**		20-20
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**		20-30
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**		20-20
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 B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-30
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**		20-20
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**		20-30
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**		20-20
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**		20-26
beqctrl		*Extended mnemonic for* **bcctrl 12,4∗cr_field+2**	(LR) ← CIA + 4.	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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**		20-26
bgectrl		*Extended mnemonic for* **bcctrl 4,4∗cr_field+0**	(LR) ← CIA + 4.	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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.	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-26
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**		20-30
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**		20-30
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**		20-20
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**		20-26
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**		20-30
bltlrl		*Extended mnemonic for* **bclrl 12,4∗cr_field+0**	(LR) ← CIA + 4.	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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**		20-26
bngctrl		*Extended mnemonic for* **bcctrl 4,4∗cr_field+1**	(LR) ← CIA + 4.	

Table B-5. Extended Mnemonics for PPC405CR (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**		20-30
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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 B-5. Extended Mnemonics for PPC405CR (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**		20-26
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**		20-30
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**		20-20
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**		20-26
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**		20-30
bnulrl		*Extended mnemonic for* **bclrl 4,4∗cr_field+3**	(LR) ← CIA + 4.	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-20
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**		20-26
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**		20-30
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**		20-20
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**		20-26
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**		20-30
btlrl		*Extended mnemonic for* **bclrl 12,cr_bit**	(LR) ← CIA + 4.	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
bun	[cr_field,] target	Branch if unordered. Use CR0 if cr_field is omitted. *Extended mnemonic for* **bc 12,4∗cr_field+3,target**		20-20
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**		20-26
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**		20-30
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**		20-147
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**		20-147
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**		20-147
clrrwi.		*Extended mnemonic for* **rlwinm. RA,RS,0,0,31–n**	CR[CR0]	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
cmplw	[BF,] RA, RB	Compare Logical Word. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpl BF,0,RA,RB**		20-36
cmplwi	[BF,] RA, IM	Compare Logical Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpli BF,0,RA,IM**		20-37
cmpw	[BF,] RA, RB	Compare Word. Use CR0 if BF is omitted. *Extended mnemonic for* **cmp BF,0,RA,RB**		20-34
cmpwi	[BF,] RA, IM	Compare Word Immediate. Use CR0 if BF is omitted. *Extended mnemonic for* **cmpi BF,0,RA,IM**		20-35
crclr	bx	Condition register clear. *Extended mnemonic for* **crxor bx,bx,bx**		20-46
crmove	bx, by	Condition register move. *Extended mnemonic for* **cror bx,by,by**		20-44
crnot	bx, by	Condition register not. *Extended mnemonic for* **crnor bx,by,by**		20-43
crset	bx	Condition register set. *Extended mnemonic for* **creqv bx,bx,bx**		20-41
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**		20-147
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**		20-147
extrwi.		*Extended mnemonic for* **rlwinm. RA,RS,b+n,32–n,31**	CR[CR0]	

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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**		20-146
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**		20-146
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**		20-9
li	RT, IM	Load immediate. (RT) ← EXTS(IM) *Extended mnemonic for* **addi RT,0,value**		20-9
lis	RT, IM	Load immediate shifted. $(RT) \leftarrow (IM \parallel {}^{16}0)$ *Extended mnemonic for* **addis RT,0,value**		20-12

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mfccr0 **mfctr** **mfdac1** **mfdac2** **mfdear** **mfdbcr0** **mfdbcr1** **mfdbsr** **mfdccr** **mfdcwr** **mfdvc1** **mfdvc2** **mfesr** **mfevpr** **mfiac1** **mfiac2** **mfiac3** **mfiac4** **mficcr** **mficdbdr** **mflr** **mfpid** **mfpit** **mfpvr** **mfsgr** **mfsler** **mfsprg0** **mfsprg1** **mfsprg2** **mfsprg3** **mfsprg4** **mfsprg5** **mfsprg6** **mfsprg7** **mfsrr0** **mfsrr1** **mfsrr2** **mfsrr3** **mfsu0r** **mftcr** **mftsr** **mfxer** **mfzpr**	RT	Move from special purpose register (SPR) SPRN. *Extended mnemonic for* **mfspr RT,SPRN** See Table 21.5, “Special Purpose Registers,” on page 21-1 for listing of valid SPRN values.		20-112
mftb	RT	Move the contents of TBL into RT, (RT) ← (TBL) *Extended mnemonic for* **mftb RT,TBL**		20-114

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mftbu	RT	Move the contents of TBU into RT, (RT) ← (TBU) *Extended mnemonic for* **mftb RT,TBU**		20-114
mr	RT, RS	Move register. (RT) ← (RS) *Extended mnemonic for* **or RT,RS,RS**		20-140
mr.		*Extended mnemonic for* **or. RT,RS,RS**	CR[CR0]	
mtcr	RS	Move to Condition Register. *Extended mnemonic for* **mtcrf 0xFF,RS**		20-116

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mtccr0 **mtctr** **mtdac1** **mtdac2** **mtdbcr0** **mtdbcr1** **mtdbsr** **mtdccr** **mtdear** **mtdcwr** **mtdvc1** **mtdvc2** **mtesr** **mtevpr** **mtiac1** **mtiac2** **mtiac3** **mtiac4** **mticcr** **mticdbdr** **mtlr** **mtpid** **mtpit** **mtpvr** **mtsgr** **mtsler** **mtsprg0** **mtsprg1** **mtsprg2** **mtsprg3** **mtsprg4** **mtsprg5** **mtsprg6** **mtsprg7** **mtsrr0** **mtsrr1** **mtsrr2** **mtsrr3** **mtsu0r** **mttbl** **mttbu** **mttcr** **mttsr** **mtxer** **mtzpr**	RS	Move to SPR SPRN. *Extended mnemonic* for **mtspr SPRN,RS** See Table 21.5, “Special Purpose Registers,” on page 21-1 for listing of valid SPRN values.		20-119
nop		Preferred no-op; triggers optimizations based on no-ops. *Extended mnemonic for* **ori 0,0,0**		20-142

Table B-5. Extended Mnemonics for PPC405CR (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
not	RA, RS	Complement register. (RA) ← ¬(RS) *Extended mnemonic for* **nor RA,RS,RS**		20-139
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**		20-150
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**		20-147
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**		20-147
rotrwi.		*Extended mnemonic for* **rlwinm. RA,RS,32–n,0,31**	CR[CR0]	
slwi	RA, RS, n	Shift left immediate. (n < 32) $(RA)_{0:31-n}$ ← $(RS)_{n:31}$ $(RA)_{32-n:31}$ ← $^{n}0$ *Extended mnemonic for* **rlwinm RA,RS,n,0,31–n**		20-147
slwi.		*Extended mnemonic for* **rlwinm. RA,RS,n,0,31–n**	CR[CR0]	
srwi	RA, RS, n	Shift right immediate. (n < 32) $(RA)_{n:31}$ ← $(RS)_{0:31-n}$ $(RA)_{0:n-1}$ ← $^{n}0$ *Extended mnemonic for* **rlwinm RA,RS,32–n,n,31**		20-147
srwi.		*Extended mnemonic for* **rlwinm. RA,RS,32–n,n,31**	CR[CR0]	

Table B-5. Extended Mnemonics for PPC405CR (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**		20-176
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**		20-177
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**		20-9
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**		20-10
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**	CR[CR0]	20-11
subis	RT, RA, IM	Subtract (IM || $^{16}0$) from (RA|0). Place result in RT. *Extended mnemonic for* **addis RT,RA,–IM**		20-12

Table B-5. Extended Mnemonics for PPC405CR (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

The PPC405CR uses load and store instructions to transfer data between memory and the general purpose registers. Load and store instructions operate on byte, halfword and word data. The storage reference instructions also support loading or storing multiple registers, character strings, and byte-reversed data. Table B-6 shows the storage reference instructions available for use in the PPC405CR.

Table B-6. 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).		20-71
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.		20-72
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.		20-73
lbzx	RT, RA, RB	Load byte from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{24}0$ || MS(EA,1).		20-74
lha	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)).		20-75
lhau	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		20-76
lhaux	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)). Update the base address, (RA) ← EA.		20-77
lhax	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and sign extend, (RT) ← EXTS(MS(EA,2)).		20-78
lhbrx	RT, RA, RB	Load halfword from EA = (RA|0) + (RB), then reverse byte order and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA+1,1) || MS(EA,1).		20-79
lhz	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		20-80

Table B-6. Storage Reference Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lhzu	RT, D(RA)	Load halfword from EA = (RA|0) + EXTS(D) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		20-81
lhzux	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2). Update the base address, (RA) ← EA.		20-82
lhzx	RT, RA, RB	Load halfword from EA = (RA|0) + (RB) and pad left with zeroes, (RT) ← $^{16}0$ || MS(EA,2).		20-83
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).		20-84
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}.		20-85
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.		20-87
lwarx	RT, RA, RB	Load word from EA = (RA|0) + (RB)and place in RT, (RT) ← MS(EA,4). Set the Reservation bit.		20-89
lwbrx	RT, RA, RB	Load word 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).		20-90
lwz	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4).		20-91

Table B-6. Storage Reference Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lwzu	RT, D(RA)	Load word from EA = (RA|0) + EXTS(D) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		20-92
lwzux	RT, RA, RB	Load word from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4). Update the base address, (RA) ← EA.		20-93
lwzx	RT, RA, RB	Load word from EA = (RA|0) + (RB) and place in RT, (RT) ← MS(EA,4).		20-94
stb	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D).		20-156
stbu	RS, D(RA)	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		20-157
stbux	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		20-158
stbx	RS, RA, RB	Store byte $(RS)_{24:31}$ in memory at EA = (RA|0) + (RB).		20-159
sth	RS, D(RA)	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D).		20-160
sthbrx	RS, RA, RB	Store halfword $(RS)_{16:31}$ byte-reversed in memory at EA = (RA|0) + (RB). MS(EA, 2) ← $(RS)_{24:31}$ ‖ $(RS)_{16:23}$		20-161
sthu	RS, D(RA)	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		20-162
sthux	RS, RA, RB	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		20-163
sthx	RS, RA, RB	Store halfword $(RS)_{16:31}$ in memory at EA = (RA|0) + (RB).		20-164
stmw	RS, D(RA)	Store consecutive words from RS through GPR(31) in memory starting at EA = (RA|0) + EXTS(D).		20-165

Table B-6. Storage Reference Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
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).		20-166
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).		20-167
stw	RS, D(RA)	Store word (RS) in memory at EA = (RA|0) + EXTS(D).		20-169
stwbrx	RS, RA, RB	Store word (RS) byte-reversed in memory at EA = (RA|0) + (RB). MS(EA, 4) $\leftarrow$ $(RS)_{24:31}$ || $(RS)_{16:23}$ || $(RS)_{8:15}$ || $(RS)_{0:7}$		20-170

stwcx.	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB) only if the reservation bit is set.
		if RESERVE = 1 then
		MS(EA, 4) $\leftarrow$ (RS)
		RESERVE $\leftarrow$ 0
		(CR[CR0]) $\leftarrow$ $^{2}0$ || 1 || XER_{so}
		else
		(CR[CR0]) $\leftarrow$ $^{2}0$ || 0 || $XER_{so.}$
				20-171

stwu	RS, D(RA)	Store word (RS) in memory at EA = (RA|0) + EXTS(D). Update the base address, (RA) $\leftarrow$ EA.		20-173
stwux	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB). Update the base address, (RA) $\leftarrow$ EA.		20-174
stwx	RS, RA, RB	Store word (RS) in memory at EA = (RA|0) + (RB).		20-175

B.6 Arithmetic and Logical Instructions

Table B-7 shows the set of arithmetic and logical instructions supported by the PPC405CR. 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-7. Arithmetic and Logical Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
add	RT, RA, RB	Add (RA) to (RB). Place result in RT.		20-6
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].		20-7
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].		20-8
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.		20-9
addic	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].		20-10
addic.	RT, RA, IM	Add EXTS(IM) to (RA|0). Place result in RT. Place carry-out in XER[CA].	CR[CR0]	20-11
addis	RT, RA, IM	Add (IM || ^{16}0) to (RA|0). Place result in RT.		20-12

Table B-7. Arithmetic and Logical Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
addme	RT, RA	Add XER[CA], (RA), (-1). Place result in RT. Place carry-out in XER[CA].		20-13
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].		20-14
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.		20-15
and.			CR[CR0]	
andc	RA, RS, RB	AND (RS) with ¬(RB). Place result in RA.		20-16
andc.			CR[CR0]	
andi.	RA, RS, IM	AND (RS) with ($^{16}0$ || IM). Place result in RA.	CR[CR0]	20-17
andis.	RA, RS, IM	AND (RS) with (IM || $^{16}0$). Place result in RA.	CR[CR0]	20-18
cntlzw	RA, RS	Count leading zeros in RS. Place result in RA.		20-38
cntlzw.			CR[CR0]	
divw	RT, RA, RB	Divide (RA) by (RB), signed. Place result in RT.		20-59
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.		20-60
divwu.			CR[CR0]	
divwuo			XER[SO, OV]	
divwuo.			CR[CR0] XER[SO, OV]	
eqv	RA, RS, RB	Equivalence of (RS) with $\overline{(RB)}$. (RA) ← ¬((RS) ⊕ (RB))		20-62
eqv.			CR[CR0]	
extsb	RA, RS	Extend the sign of byte $(RS)_{24:31}$. Place the result in RA.		20-63
extsb.			CR[CR0]	

Table B-7. Arithmetic and Logical Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
extsh	RA, RS	Extend the sign of halfword $(RS)_{16:31}$. Place the result in RA.		20-64
extsh.			CR[CR0]	
mulhw	RT, RA, RB	Multiply (RA) and (RB), signed. Place hi-order result in RT. $prod_{0:63} \leftarrow (RA) \times (RB)$ (signed). $(RT) \leftarrow prod_{0:31.}$		20-125
mulhw.			CR[CR0]	
mulhwu	RT, RA, RB	Multiply (RA) and (RB), unsigned. Place hi-order result in RT. $prod_{0:63} \leftarrow (RA) \times (RB)$ (unsigned). $(RT) \leftarrow prod_{0:31.}$		20-126
mulhwu.			CR[CR0]	
mulli	RT, RA, IM	Multiply (RA) and IM, signed. Place lo-order result in RT. $prod_{0:47} \leftarrow (RA) \times IM$ (signed) $(RT) \leftarrow prod_{16:47}$		20-129
mullw	RT, RA, RB	Multiply (RA) and (RB), signed. Place lo-order result in RT. $prod_{0:63} \leftarrow (RA) \times (RB)$ (signed). $(RT) \leftarrow prod_{32:63.}$		20-130
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.		20-131
nand.			CR[CR0]	
neg	RT, RA	Negative (two's complement) of RA. $(RT) \leftarrow \neg(RA) + 1$		20-132
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.		20-139
nor.			CR[CR0]	
or	RA, RS, RB	OR (RS) with (RB). Place result in RA.		20-140
or.			CR[CR0]	
orc	RA, RS, RB	OR (RS) with $\neg$(RB). Place result in RA.		20-141
orc.			CR[CR0]	
ori	RA, RS, IM	OR (RS) with ($^{16}0 \parallel IM$). Place result in RA.		20-142
oris	RA, RS, IM	OR (RS) with ($IM \parallel {}^{16}0$). Place result in RA.		20-143

Table B-7. Arithmetic and Logical Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
subf	RT, RA, RB	Subtract (RA) from (RB). (RT) ← ¬(RA) + (RB) + 1.		20-176
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].		20-177
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].		20-178
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].		20-179
subfme	RT, RA, RB	Subtract (RA) from (–1) with carry-in. (RT) ← ¬(RA) + (–1) + XER[CA]. Place carry-out in XER[CA].		20-180
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].		20-181
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.		
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.		

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-8 summarizes the CR logical instructions.

Table B-8. 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}.		20-39
crandc	BT, BA, BB	AND bit (CR_{BA}) with ¬(CR_{BB}). Place result in CR_{BT}.		20-40
creqv	BT, BA, BB	Equivalence of bit CR_{BA} with CR_{BB}. $CR_{BT} \leftarrow \neg(CR_{BA} \oplus CR_{BB})$		20-41
crnand	BT, BA, BB	NAND bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-42
crnor	BT, BA, BB	NOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-43
cror	BT, BA, BB	OR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-44
crorc	BT, BA, BB	OR bit (CR_{BA}) with ¬ (CR_{BB}). Place result in CR_{BT}.		20-45
crxor	BT, BA, BB	XOR bit (CR_{BA}) with (CR_{BB}). Place result in CR_{BT}.		20-46
mcrf	BF, BFA	Move CR field, (CR[CRn]) ← (CR[CRm]) where m ← BFA and n ← BF.		20-107

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-9. 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)		20-19
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.	20-20
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.	20-26
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.	20-30
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-10 shows the comparison instructions supported by the PPC405CR.

Table B-10. 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.		20-34
cmpi	BF, 0, RA, IM	Compare (RA) to EXTS(IM), signed. Results in CR[CRn], where *n* = BF.		20-35
cmpl	BF, 0, RA, RB	Compare (RA) to (RB), unsigned. Results in CR[CRn], where *n* = BF.		20-36
cmpli	BF, 0, RA, IM	Compare (RA) to ($^{16}0$ || IM), unsigned. Results in CR[CRn], where *n* = BF.		20-37

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-11 shows the PPC405CR rotate and shift instructions.

Table B-11. Rotate and Shift Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
rlwimi	RA, RS, SH, MB, ME	Rotate left word immediate, then insert according to mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m) ∨ ((RA) ∧ ¬m)		20-146
rlwimi.			CR[CR0]	
rlwinm	RA, RS, SH, MB, ME	Rotate left word immediate, then AND with mask. r ← ROTL((RS), SH) m ← MASK(MB, ME) (RA) ← (r ∧ m)		20-147
rlwinm.			CR[CR0]	
rlwnm	RA, RS, RB, MB, ME	Rotate left word, then AND with mask. r ← ROTL((RS), $(RB)_{27:31}$) m ← MASK(MB, ME) (RA) ← (r ∧ m)		20-150
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.		20-152
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).		20-153
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).		20-154
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.		20-155
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-12. Cache Control Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
dcba	RA, RB	Speculatively establish the data cache block which contains the EA (RA|0) + (RB).		20-47
dcbf	RA, RB	Flush (store, then invalidate) the data cache block which contains the EA (RA|0) + (RB).		20-49
dcbi	RA, RB	Invalidate the data cache block which contains the EA (RA|0) + (RB).		20-50
dcbst	RA, RB	Store the data cache block which contains the EA (RA|0) + (RB).		20-51
dcbt	RA, RB	Load the data cache block which contains the EA (RA|0) + (RB).		20-52
dcbtst	RA,RB	Load the data cache block which contains the EA (RA|0) + (RB).		20-53
dcbz	RA, RB	Zero the data cache block which contains the EA (RA|0) + (RB).		20-54
dccci	RA, RB	Invalidate the data cache congruence class associated with the EA (RA|0) + (RB).		20-56
dcread	RT, RA, RB	Read either tag or data information from the data cache congruence class associated with the EA (RA|0) + (RB). Place the results in RT.		20-57
icbi	RA, RB	Invalidate the instruction cache block which contains the EA (RA|0) + (RB).		20-65
icbt	RA, RB	Load the instruction cache block which contains the EA (RA|0) + (RB).		20-66
iccci	RA, RB	Invalidate instruction cache.		20-67
icread	RA, RB	Read either tag or data information from the instruction cache congruence class associated with the EA (RA|0) + (RB). Place the results in ICDBDR.		20-68

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-13 shows the interrupt control instruction set.

Table B-13. Interrupt Control Instructions

Mnemonic	Operands	Function	Other Registers Changed	Page
mfmsr	RT	Move from MSR to RT, (RT) ← (MSR).		20-111
mtmsr	RS	Move to MSR from RS, (MSR) ← (RS).		20-118
rfci		Return from critical interrupt (PC) ← (SRR2). (MSR) ← (SRR3).		20-144
rfi		Return from interrupt. (PC) ← (SRR0). (MSR) ← (SRR1).		20-145
wrtee	RS	Write value of RS_{16} to the External Enable bit (MSR[EE]).		
wrteei	E	Write value of E to the External Enable bit (MSR[EE]).		

B.13 Processor Management Instructions

The processor management instructions move data between GPRs and SPRs and DCRs in the PPC405CR; these instructions also provide traps, system calls and synchronization controls.

Table B-14. 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.		20-61
isync		Synchronize execution context by flushing the prefetch queue.		20-70
mcrxr	BF	Move XER[0:3] into field CRn, where n←BF. CR[CRn] ← (XER[SO, OV, CA]). (XER[SO, OV, CA]) ← $^{3}0$.		20-108
mfcr	RT	Move from CR to RT, (RT) ← (CR).		20-109
mfdcr	RT, DCRN	Move from DCR to RT, (RT) ← (DCR(DCRN)).		20-110

Table B-14. Processor Management Instructions (continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
mfspr	RT, SPRN	Move from SPR to RT, (RT) $\leftarrow$ (SPR(SPRN)).		20-112
mtcrf	FXM, RS	Move some or all of the contents of RS into CR as specified by FXM field, mask $\leftarrow$ $^4(FXM_0)$ || $^4(FXM_1)$ || ... || $^4(FXM_6)$ || $^4(FXM_7)$. (CR)$\leftarrow$((RS) $\wedge$ mask) $\vee$ (CR) $\wedge$ $\neg$mask).		20-116
mtdcr	DCRN, RS	Move to DCR from RS, (DCR(DCRN)) $\leftarrow$ (RS).		20-117
mtspr	SPRN, RS	Move to SPR from RS, (SPR(SPRN)) $\leftarrow$ (RS).		20-119
sc		System call exception is generated. (SRR1) $\leftarrow$ (MSR) (SRR0) $\leftarrow$ (PC) PC $\leftarrow$ $EVPR_{0:15}$ || 0x0C00 (MSR[WE, PR, EE, PE, DR, IR]) $\leftarrow$ 0		20-151
sync		Synchronization. All instructions that precede **sync** complete before any instructions that follow **sync** begin. When **sync** completes, all storage accesses initiated before **sync** will have completed.		20-182
tw	TO, RA, RB	Trap exception is generated if, comparing (RA) with (RB), any condition specified by TO is true.		
twi	TO, RA, IM	Trap exception is generated if, comparing (RA) with EXTS(IM), any condition specified by TO is true.		

Appendix C. Code Optimization and Instruction Timings

The code optimization guidelines in “Code Optimization Guidelines” and the information describing instruction timings in “Instruction Timings,” on page C-3 can help compiler, system, and application programmers produce high-performance code and determine accurate execution times.

C.1 Code Optimization Guidelines

The following guidelines can help to reduce program execution times.

C.1.1 Condition Register Bits for Boolean Variables

Compilers can use Condition Register (CR) bits to store boolean variables, where 0 and 1 represent False and True values, respectively. This generally improves performance, compared to using General Purpose Registers (GPRs) to store boolean variables. Most common operations on boolean variables can be accomplished using the CR Logical instructions.

C.1.2 CR Logical Instruction for Compound Branches

For example, consider the following pseudocode:

```
if (Var28 || Var29 || Var30 || Var 31) branch to target
```

Var28–Var31 are boolean variables, maintained as bits in the CR[CR7] field ($CR_{28:31}$). The value 1 represents True; 0 represents False.

This could be coded with branches as:

```
bt      28, target
bt      29, target
bt      30, target
bt      31, target
```

Generally faster, functionally equivalent code, using CR Logical instructions, follows:

```
crcr    2, 28, 29
cror    2, 2, 30
cror    2, 2, 31
bt      2, target
```

C.1.3 Floating-Point Emulation

Two ways of handling floating-point emulation are available.

The preferred method is a call interface to subroutines in a floating-point emulation run-time library.

Alternatively, code can use the PowerPC floating point instructions. The PPC405CR, an integer processor, does not recognize these instructions and will take an illegal instruction interrupt. The interrupt handler can be written to determine the instruction opcode and execute appropriate (integer-based) library routines to provide the equivalent function.

Because this method adds interrupt context switching time to the execution time of library routines that would have been called directly by the preferred method, it is not preferred. However, this method supports code that contains PowerPC floating-point instructions.

C.1.4 Cache Usage

Code and data can be organized, based on the size and structure of the instruction and data cache arrays, to minimize cache misses.

In the cache arrays, any two addresses in which $A_{m:26}$ (the index) are the same, but which differ in $A_{0:m-1}$ (the tag), are called congruent. (This describes a two-way set-associative cache.) $A_{27:31}$ define the 32 bytes in a cache line, the smallest object that can be brought into the cache. Only two congruent lines can be in the cache simultaneously; accessing a third congruent line causes the removal from the cache of one of the two lines previously there.

Table C-1 illustrates the value of *m* and the index size for the various cache array sizes.

Table C-1. Cache Sizes, Tag Fields, and Lines

Array Size	Instruction Cache Array			Data Cache Array		
	m (Tag Field Bits)	*n* (Lines)	Index Bits	*m* (Tag Field Bits)	*n* (Lines)	Index Bits
0KB	—	—	—	—	—	—
4KB	22 (0:21)	64	21:26	21 (0:20)	64	21:26
8KB	22 (0:21)	128	20:26	20 (0:19)	128	20:26
16KB	22 (0:21)	256	19:26	19 (0:18)	256	19:26
32KB	22 (0:21)	512	18:26	18 (0:17)	512	18:26

Moving new code and data into the cache arrays occurs at the speed of external memory. Much faster execution is possible when all code and data is available in the cache. Organizing code to uniformly use $A_{m:26}$ minimizes the use of congruent addresses.

C.1.5 CR Dependencies

For CR-setting arithmetic, compare, CR-logical, and logical instructions, and the CR-setting **mcrf**, **mcrxr**, and **mtcrf** instructions, put two instructions between the CR-setting instruction and a Branch instruction that uses a bit in the CR field set by the CR-setting instruction.

C.1.6 Branch Prediction

Use the Y-bit in branch instructions to force proper branch prediction when there is a more likely prediction than the standard prediction. See “Branch Prediction” on page 3-32 for a more information about branch prediction.

C.1.7 Alignment

For speed, align all accesses on the appropriate operand-size boundary. For example, load/store word operands should be word-aligned, and so on. Hardware does not trap unaligned accesses; instead, two accesses are performed for a load or store of an unaligned operand that crosses a word boundary. Unaligned accesses that do not cross word boundaries are performed in one access.

Align branch targets that are unlikely to be hit by “fall-through” code on cache line boundaries (such as the address of functions such as **strcpy**), to minimize the number of unused instructions in cache line fills.

C.2 Instruction Timings

The following timing descriptions consider only “first order” effects of cache misses in the ICU (instruction-side) and DCU (data-side) arrays.

The timing descriptions *do not* provide complete descriptions of the performance penalty associated with cache misses; the timing descriptions do not consider bus contention between the instruction-side and the data-side, or the time associated with performing line fills or flushes. Unless specifically stated otherwise, the number of cycles apply to systems having zero-wait memory access.

C.2.1 General Rules

Instructions execute in order.

All instructions, assuming cache hits, execute in one cycle, except:

- Divide instructions execute in 35 clock cycles.
- Branches execute in one or three clock cycles, as described in “Branches.”
- MAC and multiply instructions execute in one to five cycles as described in “Multiplies.”
- Aligned load/store instructions that hit in the cache execute in one clock cycle/word. See “Alignment” for information on execution timings for unaligned load/stores.
- In isolation, a data cache control instruction takes two cycles in the processor pipeline. However, subsequent DCU accesses are stalled until a cache control instruction finishes accessing the data cache array.

Note: Note that subsequent DCU accesses do not remain stalled while transfers associated with previous data cache control instructions continue on the PLB.

C.2.2 Branches

Branch instructions are decoded in prefetch buffer 0 (PFB0) and the decode stage of the instruction pipeline. Branch targets, whether the branch is known or predicted taken, can be fetched from the PFB0 and DCD stages. Incorrectly predicted branches can be corrected from the DCD or EXE (execute) stages of the pipeline.

Branches can be known taken or known not taken, or can have address or condition dependencies. Branches having address dependencies are never predicted taken. The directions of conditional branches having no address dependencies are statically predicted.

Conditional branches may depend on the results of an instruction that is changing the CR or the CTR.

Address dependencies can occur when:

- A **bclr** instruction that is known taken, or unresolved, follows (immediately, or separated by only one instruction) a link updating instruction (**mtlr** or a branch and link).
- A **bcctr** instruction that is known taken, or unresolved, follows (immediately, or separated by only one instruction) a counter updating instruction (**mtctr** or a branch that decrements the counter).

Instruction timings for branch instructions follow:

- A branch known not taken (BKNT) executes in one clock cycle. By definition a BKNT does not have address or condition dependencies.
- A branch known taken (BKT) by definition has no condition dependencies, but can have address dependencies. A BKT without address dependencies can execute in one clock cycle if it is first decoded from the PFB0 stage, or in two clock cycles if it is first decoded in the DCD stage. A BKT having address dependencies can execute in two clock cycles if there is one instruction between the branch and the address dependency, or in three clock cycles if there are no instructions between the branch and address dependency.
- A branch predicted not taken (BPNT), which must have condition dependencies, executes in one clock cycle if the prediction is correct. If the prediction is incorrect, the branch can take two or three cycles. If there was one instruction between the branch and the instruction causing the condition dependency, the branch executes in two cycles. If there were no instructions between the branch and the instruction causing the condition dependency, the branch executes in three clock cycles.
- A branch that is correctly predicted taken (BPT), which must have condition dependencies, executes in one clock cycle, if it is first decoded from the PFB0 stage, or two clock cycles if it is first decoded in the DCD stage. If the prediction is incorrect, the branch can take two or three cycles. If there is one instruction between the branch and the instruction causing the condition dependency, the branch executes in two cycles. If there are no instructions between the branch and the instruction causing the condition dependency, the branch executes in three clock cycles.

C.2.3 Multiplies

For multiply instructions having two word operands, hardware internal to the core automatically detects smaller operand sizes (by examining sign bit extension) to reduce the number of cycles necessary to complete the multiplication.

The PPC405CR also supports multiply accumulate (MAC) instructions and multiply instructions having halfword operands.

Word and halfword multiply instructions are pipelined in the execution unit and use the same multiplication hardware. Because these instructions are pipelined in the execution stage they have latency and reissue rate cycle numbers. Under conditions to be described, a second multiply or MAC instruction can begin execution before the first multiply or MAC instruction completes. When these conditions are met, the reissue rate cycle numbers should be used; otherwise, the latency cycle numbers should be used. (A MAC or multiply instruction can follow another MAC or a multiply and still meet the conditions that support the use of the reissue rate cycle numbers.

Use reissue rate cycle numbers for multiply or MAC instructions that are followed by another multiply or MAC instruction, and do not have an operand dependency from a previous multiply or MAC instruction. However, one operand dependency is allowed for reissue rate cycle numbers. Internal forwarding logic allows the accumulate value of a first MAC instruction to be used as the accumulate value of a second MAC instruction without affecting the reissue rate.

Use latency cycle numbers for multiply or MAC instructions that are not followed by another multiply or MAC, or that have an operand dependency from a previous multiply or MAC instruction. However, accumulate-only dependencies between adjacent MAC instructions use reissue rate cyle numbers.

An operand dependency exists when a second multiply or MAC instruction depends on the result of a first multiply or MAC instruction.

Table C-2 summarizes the multiply and MAC instruction timings. In the table, the syntax "[**o**]" indicates that the instruction has an "o" form that updates XER[SO,OV], and a "non-o" form. The syntax "[**.**]" indicates that the instruction has a "record" form that updates CR[CR0], and a "non-record" form.

Table C-2. Multiply and MAC Instruction Timing

Operation	Reissue Rate Cycles	Latency Cycles
MAC		
MAC and negative MAC instructions	1	2
Halfword × Halfword		
mullhw[**.**], **mullhwu**[**.**], **mulhhw**[**.**], **mulhhwu**[**.**], **mulchw**[**.**], **mulchwu**[**.**]	1	2
mulli[**.**], **mullw**[**o**][**.**], **mulhw**[**.**], **mulhwu**[**.**]	2	3
Halfword × Word		
mulli[**.**], **mullw**[**o**][**.**], **mulhw**[**.**], **mulhwu**[**.**]	2	3
Word × Word		
mullw[**o**][**.**], **mulhw**[**.**], **mulhwu**[**.**]	4	5

C.2.4 Scalar Load Instructions

Generally, the PPC405CR executes cachable load instructions that hit in the data cache array or fill buffer, or noncachable load instructions that hit in the fill buffer (when enabled), in one cycle. However, the pipelined nature of load instructions can even cause loads that hit in the cache or line buffer to appear to take extra cycles under some conditions.

If a load is followed by an instruction that uses the load target as an operand, a load-use dependency exists. When the load target is returned, it is forwarded to the operand register of the "using" instruction. This forwarding results in an additional cycle of latency to a load immediately followed by a "using" instruction, causing the load to appear to execute in two cycles.

Because the PPC405CR can execute instructions that follow load misses if no load-use dependency exists, the load and the "using" instruction should be separated by two "non-using" instructions when possible. If only one instruction can be placed between the load and the "using" instruction, the load appears to execute in two cycles.

C.2.5 Scalar Store Instructions

Cachable stores that miss in the DCU, and noncachable stores, are queued in the data cache so that the store appears to execute in a single cycle if operand-aligned. Under certain conditions, the DCU can pipeline up to three store instructions. (See Chapter 4, "Cache Operations," for more information.) **stwcx.** instructions that do not cause alignment errors execute in two cycles.

C.2.6 Alignment in Scalar Load and Store Instructions

The PPC405CR requires an extra cycle to execute scalar loads and stores having unaligned big or little endian data (except for **lwarx** and **stwcx.**, which require word-aligned operands). If the target data is not operand aligned, and the sum of the least two significant bits of the effective address (EA)

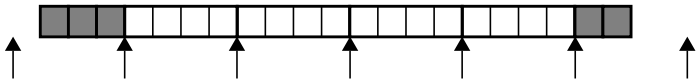
and the byte count is greater than four, the PPC405CR decomposes a load or store scalar into two load or store operations. That is, the PPC405CR never presents the DCU with a request for a transfer that crosses a word boundary. For example, a **lwz** with an EA of 0b11 causes the PPC405CR to decompose the **lwz** into two load operations. The first load operation is for a byte at the starting effective address; the second load operation is for three bytes, starting at the next word address.

C.2.7 String and Multiple Instructions

Calculating execution times for string and multiple instructions (**lmw** and **stmw**) instructions requires an understanding of data alignment, and of the behavior of the string instructions with respect to alignment.

In the following example, the string contains 21 bytes. The first three bytes do not begin on a word boundary, and the final two bytes do not end on a word boundary. The PPC405CR handles any unaligned leading bytes as a special case, then moves as many bytes as aligned words as possible, and finally handles any unaligned trailing bytes as a special case.

In the following example, arrows indicate word boundaries (the address is an exact multiple of four); shaded boxes represent unaligned bytes.



The execution time of the string instruction is the sum of the:

1. Cycles required to handle unaligned leading bytes; if any, add one clock cycle.

   In the example, there are unaligned leading bytes; this transfer adds one clock cycle.

2. Cycles required to handle the number of word-aligned transfers required. Assuming data cache hits, each word-aligned transfer requires one clock cycle.

   In the example, there are four aligned words; this transfer requires four clock cycles.

3. Cycles required to handle unaligned trailing bytes; if any, add one clock cycle.

   In the example, there are unaligned trailing bytes; this transfer adds one clock cycle.

A string instruction operating on the example 21-byte string requires six clock cycles.

C.2.8 Loads and Store Misses

Cachable stores that miss in the DCU, and noncachable stores, are queued internally in the DCU so that the store instruction appears to execute in one cycle. Under certain conditions, the DCU can pipeline up to three store instructions. (See the Chapter 4, “Cache Operations,” for more information.)

Because the PPC405CR can execute instructions that follow load misses if no load-use dependency exists, the load and the “using” instruction should be separated by “non-using” instructions whenever possible. The number of load miss penalty cycles incurred by a load that misses in the DCU or DCU fill buffer is reduced by one cycle for every non-use instruction following the load. When the number of non-use instructions following the load is equal to or greater than the number of cycles that it takes to obtain the load data, the load instruction appears to execute in a single cycle. The number of cycles that it takes to obtain load data when it misses in the data cache and fill buffer depends on whether operand forwarding is enabled or disabled and the system memory timing.

C.2.9 Instruction Cache Misses

Refer to “Instruction Processing” on page 3-29 for detailed information about the instruction queue and instruction fetching.Table C-4 illustrates instruction cache penalties for cachable and noncachable fetches that miss in the ICU array and fill buffer.

Table C-4. Instruction Cache Miss Penalties

Type of ICU Request	Miss Penalty Cycles
Sequential	3
Branch Taken from DCD	5
Branch Taken from PFB0	4

Table C-4 assumes that:

- The PPC405CR and processor local bus (PLB) run at the same frequency
- The PLB returns an address acknowledge during the first cycle in which the DCU asserts the PLB request
- The target instruction is returned in the cycle following the address acknowledge cycle

The penalty cycles shown for sequential ICU requests assume that the DCD stage and pre-fetch queue are filled with single-cycle nonbranching instructions or BKNT branch instructions. The penalty cycles for the remaining two rows are for taken branches from DCD and PFB0 respectively.

Index

C

D

E

R

S

T

The information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change IBM's product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN "AS IS" BASIS. In no event will IBM be liable for any damages arising directly or indirectly from any use of the information contained in this document.

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY
12533-6531

The IBM home page can be found at www.ibm.com

The IBM Microelectronics Division home page can be found at www.chips.ibm.com

Address technical queries about this product to ppcsupp@us.ibm.com