LSI LOGIC

MiniRISC™ CW4010 Superscalar Microprocessor Core Technical Manual

A CoreWare® Product

Order Number C14032

This document is preliminary. As such, it contains data derived from functional simulations and performance estimates. LSI Logic has not verified either the functional descriptions, or the electrical and mechanical specifications using production parts.

Document DB14-000027-00, First Edition (July 1996)
This document describes revision A of LSI Logic Corporation's CW4010 Superscalar Microprocessor Core and will remain the official reference source for all revisions/releases of this product until rescinded by an update.

To receive product literature, call us at 1-800-574-4286 (or 415-940-6877 outside the U.S. and Canada) and ask for Department JDS; or visit us at http://www.lsilogic.com.

TRADEMARK ACKNOWLEDGMENT
LSI Logic logo design and CoreWare are registered trademarks and MiniRISC and MiniSIM are trademarks of LSI Logic Corporation. Sun and SPARCstation are trademarks of Sun Microsystems, Inc. SPARC is a registered trademark of SPARC International, Inc. Products bearing the SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc. MIPS is a trademark of MIPS Technologies, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc. All other brand and product names may be trademarks of their respective companies.

Preface

This book is the primary reference and technical manual for the MiniRISC™ CW4010 Superscalar Microprocessor Core, referred to in this document as the CW4010 core, the CW4010, or the core. The book contains a complete functional description of the CW4010.

Audience

The book is intended for use by engineers and managers who are evaluating the CW4010 core, or for engineers who are designing with the core. The book assumes that this audience is familiar with the concepts of microprocessors and related support devices.

Organization

The book has the following chapters and a glossary of terms.

- Chapter 1, **Introduction**, provides an overview of the CW4010 core and describes the features of the LSI Logic CoreWare® program.
- Chapter 2, **Architectural Overview**, describes the CPU pipeline and microarchitecture, the instructions set architecture, the system coprocessor (CP0), memory management, exception processing, and cache maintenance.
- Chapter 3, **Instruction Set Summary**, describes the MIPS R-series instructions and the instruction set extensions supported in the CW4010 core.
- Chapter 4, **CW4010 Exception Processing**, describes how the CW4010 handles exception processing.
- Chapter 5, **CW4010 Memory Management**, provides detailed information about CP0 and the CW4010 memory management system.
- Chapter 6, **CW4010 Caches**, provides detailed information about the CW4010 caches and cache maintenance.

- Chapter 7, **Signals**, describes the CW4010 core I/O signals.
- Chapter 8, **Interface Operation**, describes the main timing scenarios for CW4010 transactions.
- Chapter 9, **Specifications**, refers you to an addendum that contains specifications for the CW4010 core.
- Appendix A, **Programmer's Notes**, provides information that is useful if you are writing software for the CW4010 core.

Related Publications

CW33300 Enhanced Self-Embedding Processor Core User's Manual, Order No. C14014

Conventions Used in This Manual

Terms that appear in the glossary are shown in **boldface** the first time they are mentioned in the text.

The term "word" is used to define a 32-bit quantity, either signed or unsigned. This means that in the CW4010 core a word consists of four 8-bit bytes; a doubleword has 64 bits, or eight 8-bit bytes; and a halfword has 16 bits, or two 8-bit bytes.

Hexadecimal numbers are indicated by the prefix 0x before the number, for example, 0x32CF. Binary numbers are indicated by a subscript "2" following the number, for example $000.0010.1100.1111_2$.

The following signal conventions are used throughout the manual:

- Signals that are inputs have a lower case "i" as part of the signal name, for example, SCDip. Signals that are outputs have a lower case "o", for example, SCDop. Lowercase characters are used to avoid confusion between uppercase "I" and "1", and uppercase "O" and "0."
- Active-LOW signals have a lowercase "n" at the end of the signal name, for example RESETn. Active-HIGH signals have a lowercase "p" at the end of the signal name, for example SCAop.
- The term "assert" means to drive a signal true or active. The term "deassert" means to drive a signal false or inactive.

Contents

Tables

Chapter 1
Introduction

This chapter introduces the LSI Logic CoreWare program and describes its features. It also provides an overview of the CW4010 core. This chapter contains the following sections:

1.1 CW4010 Overview

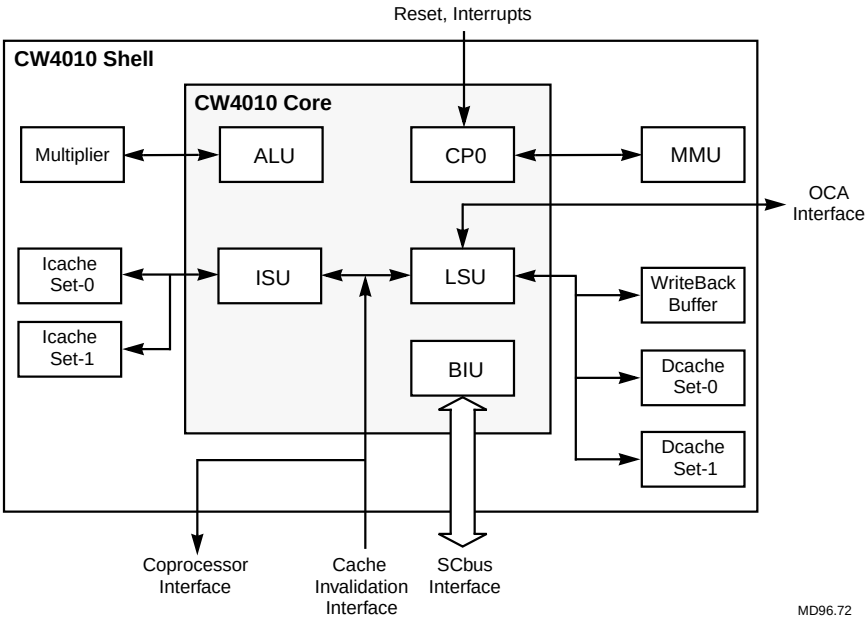
LSI Logic Corporation has developed the MiniRISC CW4010 Superscalar Core, the world’s first MIPS-II compatible superscalar core, using LSI Logic’s CoreWare system-on-a-chip methodology. The CW4010 is a member of LSI Logic’s MiniRISC family, the next generation of MIPS RISC products.

You can use the CW4010 as a microprocessor core in products that require higher performance than that of the CW4001 microprocessor core. The CW4010 is available as a CoreWare product for use in customer ASIC designs, and is also used in LSI Logic’s ASSPs (Application Specific Standard Products).

1.1.1 Core and Shell

As shown in Figure 1.1, the CW4010 is implemented at two levels: the core and the shell.

Figure 1.1 CW4010 Core Interface to External Building Blocks



The CW4010 superscalar microprocessor core is an **encrypted** synthesizable **Verilog model**. It is process independent and made up of the following units:

- An arithmetic logic unit (ALU)
- A system control coprocessor (CP0)
- A bus interface unit (BIU)
- A load store unit (LSU)
- An instruction scheduler unit (ISU)

The following microprocessor building blocks are available with the basic microprocessor core and are shown as part of the shell. The shell is an **unencrypted** Verilog model that contains:

- A direct-mapped or two-way set associative instruction cache with cache sizes selectable up to 16 Kbytes
- A direct-mapped or two-way set associative data cache

- A memory management unit (MMU) with an up to 64-entry translation lookaside buffer (TLB)
- A standard multiply/divide unit or a high-performance multiply/accumulate unit
- A WriteBack Buffer for writeback cache mode

1.1.2 Interfaces

The CW4010 has four on-chip interfaces:

- The coprocessor interface connects the core with up to three coprocessors (CP1, CP2, and CP3), as well as the internal coprocessor (CP0).
- The cache invalidation interface connects the core with optional cache coherency logic. The core uses this bus to communicate only with the on-chip caches.
- The SCbus, the bidirectional system bus, allows the CW4010 to communicate with system elements outside the core.
- The OCA (On-Chip Access Bus) allows access to on-chip modules at the CR stage without going through the SCbus.

1.1.3 Related Modules

In addition to the core, the MiniRISC product family includes a variety of other modules including:

- LSI Logic's MiniSIM™ architectural simulator
- Verilog and VHDL models
- A system verification environment
- A PROM monitor
- Third party software support
- Core bond-out chip for emulation
- Evaluation boards for concurrent software development
- LSI Logic's CoreWare, described in Section 1.3, "CoreWare Program," on page 1-5

1.2 Features

The CW4010 core has the following features:

- Full MIPS-II instruction set implementation (R4000 32-bit mode compatible)
- Instruction set extensions to support embedded applications
- Superscalar execution with up to two instructions issued per clock cycle
- 64-bit on-chip system interface
- High-performance coprocessor interface for user definable coprocessors and high performance hardware floating-point unit (FPU)
- 3.3 Volt operation
- 80-MHz worst-case commercial maximum clock rate using standard cell ASIC
- 150 Dhrystone MIPS at 80 MHz
- 160 native MIPS peak, 110 native MIPS sustained with standard compiled MIPS code at 80 MHz
- Core Power—5 mW/MHz with power management
- Configurable modular design to meet customer requirements
- Integrated cache controllers with separate instruction and data caches—sizes selectable from 2 Kbytes to 16 Kbytes each
- Optional, modifiable building blocks, such as a Multiplier with or without Accumulator and a Memory Management Unit (MMU)
- Fully testable in embedded ASIC designs
- Models available:
  - Performance and software development model
  - Verilog and VHDL models (referred to in this manual as HDL models)
  - Gate-level, timing-accurate model in various third party simulation environments
- Compatible with the full range of MIPS and third party software development tools
- Compact basic microprocessor core size—3 mm by 3 mm including BIU, cache controllers, and write buffer

1.3 CoreWare Program

The CoreWare program offers a new approach to system design. Through the CoreWare program, LSI Logic gives you the ability to combine the MiniRISC CW4010 Superscalar Microprocessor Core with other cores (such as microprocessors, floating-point processors, and peripheral building blocks) on a single chip, and to create products uniquely suited to your applications. This approach—combining high-performance building blocks, sophisticated design software, and expert support—gives you unparalleled design flexibility and allows you to create high-quality, leading-edge products for a wide range of markets.

CoreWare is LSI Logic's proprietary approach to creating custom, single-chip systems with more complex high-level logic blocks than those provided by standard gate arrays. It provides a new paradigm that allows you to use applications-optimized engineering.

CoreWare elements allow you to produce silicon that meets your specific application requirements. Complex systems-on-a-chip can be fabricated with unprecedented time-to-market and low cost compared with the gate array approach or full custom design.

The CoreWare program provides three major design components:

- CoreWare building blocks
- A design environment
- Expert technical support

1.3.1 CoreWare Building Blocks

CoreWare building blocks include elements based on LSI Logic's high-performance standard products as well as other, industry-standard products. These blocks are fully supported library elements for use in the LSI Logic hardware development environment.

The CoreWare library contains a wide range of complex cores based on accepted and emerging industry standards for networking, system logic, digital video, and other applications. These cores can be combined with your own unique logic to create a wide variety of single-chip applications such as network routers, RAID disk controllers, and PC I/O docking stations. Note that the building blocks include gate-level simulation models with timing information, so you can accurately simulate device performance and trade off various implementation options. In addition to gate-level simulation models, some building blocks also include behavioral simulation models.

1.3.2 Design Environment

The CoreWare building blocks, which include embedded MIPS and SPARC processors, bus interface controllers, and a family of floating-point processors, are fully supported library elements for use in the LSI Logic hardware development environment.

1.3.3 Expert Support

LSI Logic's in-house experts support the CoreWare program with high-level design and market experience in a wide variety of application areas. These experts provide design support from system architecture definition through chip layout and test vector generation. They help determine how many functions to integrate on a single chip, trading off functionality versus cost to find the most cost-effective solution. When the trade-offs are complete, working with LSI Logic's applications engineers, you can implement and test the design.

Chapter 2
Architectural Overview

This chapter discusses the CPU pipeline and microarchitecture, the instruction set architecture, the System Coprocessor (Coprocessor-0), Memory Management, Exception Processing, and Cache Maintenance. This chapter contains the following sections:

2.1 Architectural Overview

The CW4010 is fully compatible with the R3000 and R4000 32-bit instruction sets (MIPS-I and MIPS-II), but uses an updated hardware architecture to provide higher absolute performance than any other available MIPS core. The CW4010 also provides substantially better instructions-per-clock performance than other MIPS processors. At the same time, the hardware design remains compact in comparison with other superscalar architectures.

The CW4010 implements a 32-bit virtual address space. Individual memory locations are byte-addressed. Up to 2 Gbytes of virtual address space is available to each user-level process.

The CW4010 implements a 32-bit physical address space. Individual memory locations are byte-addressed, and, combined with the virtual address space, provide a total of 4 Gbytes of addressable physical memory.

The CW4010 can issue and complete two instructions per cycle using a combination of five independent execution units:

- Arithmetic Logic Unit (ALU)
- Load/Store/Add Unit (LSU)

  LSU executes load and store instructions. It also executes add and load immediate instructions, allowing an add instruction to be issued with another add or logical instruction.
- Branch Unit
- Multiply/Shift Unit
- Coprocessor Interface

  Coprocessor Interface can feed an instruction to an LSI or customer-defined CoProcessor Unit

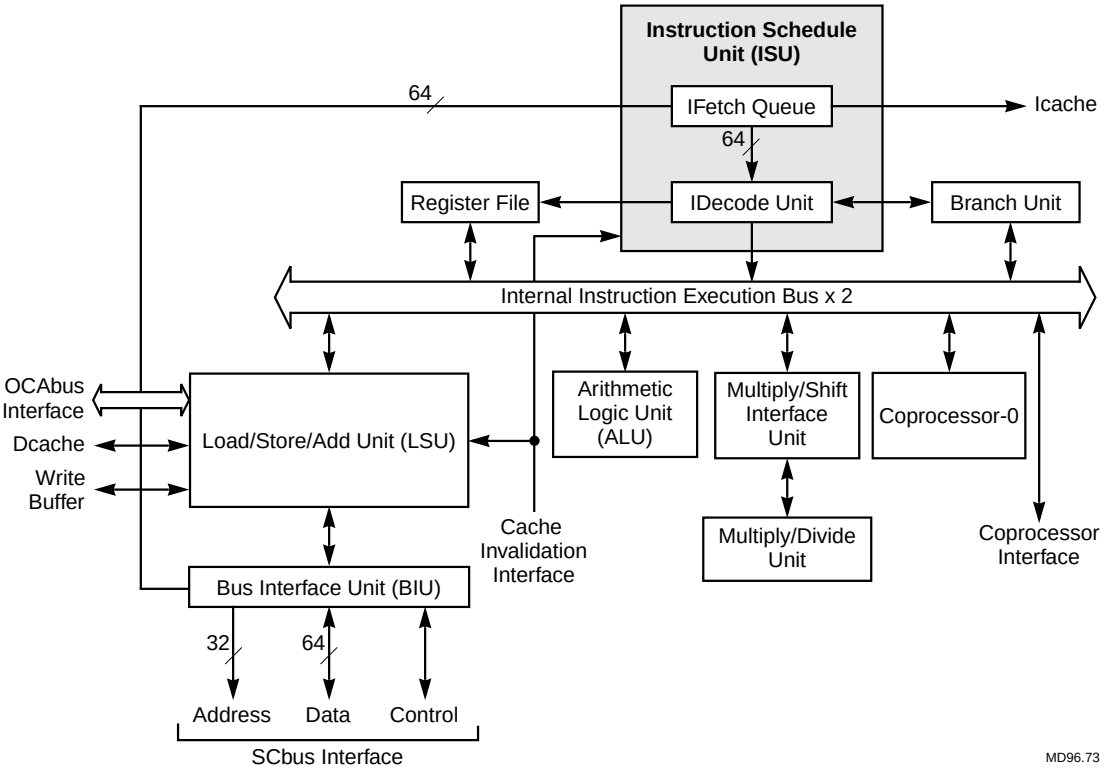
All instructions, except multiply and divide, can be completed in a single cycle.

Load instructions have a single hardware delay slot for loads that hit in the cache, but the hardware activates an interlock on register conflicts so that no NOP (no operation) is required in the delay slot. On a load miss, the CW4010 extends the hardware conflict detection so that if the load data is not required by subsequent instructions in the pipeline, the CPU is not stalled. The operation is called *load scheduling.*

The CW4010 has an instruction pre-fetch queue and branch prediction logic to boost branch performance. This means that correctly predicted branches are completed with no penalty and incorrectly predicted branches normally have a penalty of just one cycle. The CW4010 accomplishes branch prediction with a simple hardware algorithm that more than 90% accurate for most application code.

Figure 2.1 shows a simplified block diagram of the basic CPU core.

Figure 2.1
CW4010 Block Diagram



Three units handle instructions:

- The IFetch Queue optimizes the supply of instructions to the microprocessor, even across breaks in the sequential flow of execution (jumps and branches).
- The IDecode Unit decodes the instructions from the IFetch Queue, determines the actions required for the instruction execution, and manages the Register File, LSU, ALU, and Multiply/Divide Units accordingly.
- The Branch Unit is used when branch and jump instructions are recognized within the instruction stream.

The Register File contains the core's general purpose registers. (There are 32 general purpose registers located in CP0. Of these registers 31 are read/write registers and one is the zero register.) The Register File

supplies source operands to the execution units and handles the storage of results to target registers.

Three units perform logical, arithmetic, and data-movement operations:

- The Load/Store/Add Unit (LSU) manages loads and stores of data values. Data values are loaded from either the Dcache or from the SCbus Interface in the event of a Dcache miss. Stores pass to the Dcache and the SCbus Interface through the Write Buffer. The LSU is also able to perform a restricted set of arithmetic operations, including the addition of an immediate offset as required in address calculations.
- The Arithmetic Logic Unit (ALU) calculates the result of an arithmetic or logical operation.
- The Multiply/Shift Interface Unit performs multiply and divide operations. You can select a number of modular options for this unit, including an option with full multiply/accumulate capability.

The CW4010 core has four interfaces:

- The Bus Interface Unit (BIU) manages the flow of instructions and data between the core and the system by means of the SCbus Interface. This interface provides the main channel for communication between the CW4010 core and the other functional blocks in the system. Some blocks may be implemented as CoreWare library functions integrated on the same die as the Microprocessor Core; others may be implemented in separate devices connected by means of I/O pins at the board level.
- The Coprocessor Interface allows tightly coupled special-purpose processing units to be attached to the core, enhancing the microprocessor's general-purpose computational power. This approach allows high-performance application-specific hardware to be made directly accessible to the programmer at the instruction set level. For example, a coprocessor might offer accelerated bit-mapped graphics operations or real-time video decompression.
- The Cache Invalidation Interface allows supporting hardware outside the Microprocessor Core to maintain the coherency of on-board cache contents for systems that include multiple main-bus masters.
- The OCAbus Interface allows on-chip modules to be accessed at the CR (Cache Read) stage of the pipeline without going through an

SCbus transaction. This improves performance since it reduces traffic on the SCbus and therefore reduces latency.

2.2 Cache and External Interface

Instruction cache (Icache) control is performed by the ISU (Instruction Scheduling Unit). Data cache (Dcache) control is performed by the LSU (Load/Store Unit).

A Write Buffer is also implemented within the LSU, so that CPU execution need not stall if a number of stores are performed in quick succession. The Write Buffer accepts the store addresses and data values, and passes them on to main memory as rapidly as it can accept them. During this time, the CPU proceeds with execution.

The BIU provides the interface to on-chip peripherals. One or more peripherals will typically provide a path to off-chip resources, including main memory.

The on-chip system interface presented by the BIU is the SCbus. This bus has a 64-bit data bus and a 32-bit address bus. Address and data are not multiplexed.

Refills for both the Icache and Dcache exploit the 64-bit width of the data bus to achieve the highest possible performance.

In its standard form, the BIU does not include any support for dynamic bus sizing. That is, it provides no mechanism to subdivide transactions between the CPU core and the on-chip peripherals, which can directly support the requested data transfer width. However, such functionality can be accommodated through simple supporting logic interposed between the BIU and the on-chip peripherals.

Again, the standard version BIU does not contain support for WriteBack data cache management. It implements only a WriteThrough policy. However, an additional supporting unit, the WriteBack Buffer, may be attached to provide WriteBack control in those applications which require it.

Excluding features such as dynamic bus sizing and WriteBack cache management from the standard BIU keeps it small and simple. Designs that do not need these features are not compelled to implement them.

2.3 Clocking and Power Management

The CPU core is clocked by a single phase, 1x clock with a 40–60% duty cycle requirement. Applications that require a slower system clock interface may use a phase-locked loop (PLL) available as a cell in LSI Logic's ASIC libraries and logic to implement a clock multiplier circuit for the CPU.

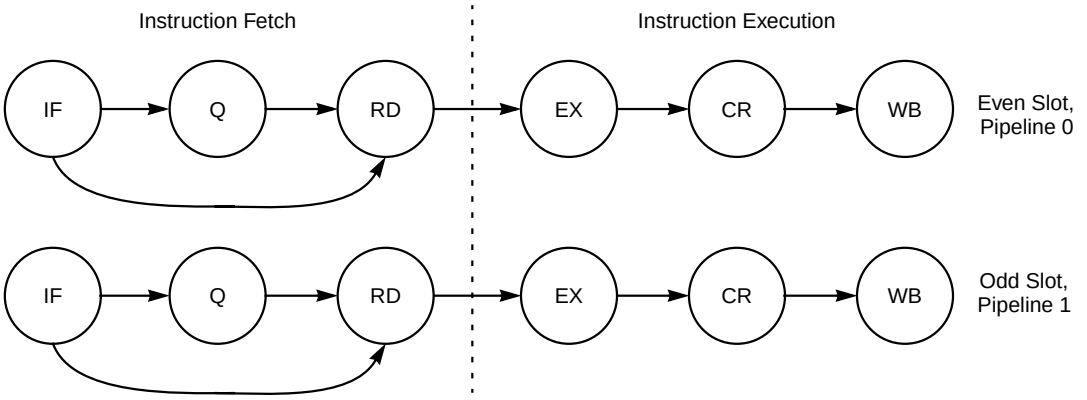
Power management is provided by the CPU by WAITI (Wait for Interrupt) instruction and by gating the clock separately for each functional unit. Units are clocked only when needed. In addition, the core and cache RAMs are completely static, so that the clock may be slowed or turned off by the user logic to save power.

2.4 Pipeline Architecture (ISA)

This section describes the CPU pipelines, instruction fetching and scheduling. It also contains an instruction set summary.

As shown in Figure 2.2, the CW4010 core has two identical concurrent six-stage pipelines that provide the core with its superscalar capabilities. One pipeline is known as the even pipeline or even slot, and the other as the odd pipeline or odd slot.

Figure 2.2
CW4010 Instruction Pipeline



1. Branch instruction encountered.
2. Q state bypassed.

MD96.74

The first three pipeline stages are used during instruction fetch and the last three stages during instruction execution. Once a stage has accepted an instruction from the previous stage it must hold the

instruction for re-execution in case the pipeline stalls. The function of each pipeline stage is summarized below.

1. IF (Instruction Fetch)—the CW4010 fetches the instruction during the first stage.
2. Q (Queuing)—instructions may enter this conditional stage if they deal with branches or register conflicts. An instruction that does not cause a branch or register conflict is fed directly to the RD stage.
3. RD (Read)—during this stage, any required operands are read from the Register File while the instruction is decoded.
4. EX (Execute)—all instructions are executed in this stage. Conditional branches are resolved in this cycle. The address calculation for load and store instructions is performed in this stage.
5. CR (Cache Read)—this stage is used to read the cache for load and store instructions. Data is returned to the register bypass logic at the end of this stage.
6. WB (WriteBack)—results are written into the Register File during this stage.

Sections 2.4.1, 2.4.2, and 2.4.3 provide more detailed information about pipeline transactions.

2.4.1 Instruction Fetch and Scheduling: IF, Q, and RD Stages

The IF, Q, and RD stages fetch two instructions per cycle and issue them to the EX (execute) stage. The CW4010 fetches instructions as doubleword aligned pairs (slot 0 and slot 1). There is a two-instruction window in the RD stage during the instruction decode operation. When only slot 0 can be scheduled because slot 1 has a dependency, the window slides down one instruction. In other words, although instructions are always fetched as doubleword pairs, they are scheduled on single-word boundaries.

The primary purpose of the Q stage is the execution of branch instructions with minimal penalty. The CW4010 generally fills the Q stage whenever the RD stage has to stall. This occurs fairly frequently on typical compiled code, because of register conflicts, cache misses, and resource conflicts. Filling the Q stage in these cases allows the IF stage to work ahead one cycle.

If a branch instruction is encountered when the Q stage is already active, it is predicted that the branch will be taken. The IF stage does not bring

in any more instructions following from the current address, but instead begins fetching those instructions starting at the branch target address. At this point, the Q stage still holds the pair of instructions immediately following the pair that contained the branch.

The branch target enters the RD stage, bypassing the Q stage, as shown in Figure 2.2 on page 2-6. The branch prediction logic in the ISU (Instruction Scheduler Unit) resolves the branch condition when the branch instruction enters the EX stage. If the branch prediction logic predicts the branch correctly, the instructions in the Q stage are cancelled. If it predicts the branch incorrectly, the ISU cancels the branch target. In this case, it takes non-branch sequential instructions from the Q stage and restarts the IF stage at the non-branch sequential stream. The process is different when the branch instruction is in the odd instruction slot.

If the branch prediction logic correctly predicts a branch in the even instruction slot when the Q stage is full, there is generally no cycle penalty associated with it. If the branch prediction logic predicts the branch incorrectly, the branch has a one cycle penalty.

If the branch instruction was in the odd instruction slot, the branch delay slot instruction always executes by itself and has no chance to fill the other execution slot. There may be some advantage to a software assembler that can attempt to place branches in even word addresses.

The branch prediction logic must be able to look at two instructions at the same time, from either the Q latches or the RD latches, depending on whether the Q stage is active. When it looks at the two instructions, if one is a branch, it passes the offset in that instruction into a dedicated adder to calculate the branch address for the IF stage of the instruction fetch. Because this is done speculatively, it also saves the non-branch value of the PC (Program Counter) for the possible restart of the sequential instructions from the Q stage.

After the ISU has allowed an instruction pair to pass into the RD stage, the instruction is decoded, and at the same time the register source addresses are passed to the register file so that the operands can be read. Register dependencies and resource dependencies are checked in this stage. If the instruction in slot 0 has no dependency on a register or resource currently tied up by a previous instruction, it is passed immediately into the EX stage where it forks to the appropriate execution

unit. The instruction in slot 1 may also be dependent on a resource or register in slot 0, so it must be checked for dependencies against both slot 0 and any previous unretired instruction. If either instruction must be held in the RD stage and the Q stage is not full, the IF stage is allowed to continue to fill the Q stage. If the Q stage is full, then the Q and IF stages are frozen (stalled).

In the RD stage, register bypass opportunities are considered and the bypass multiplexer control signals are set for potential bypass cases from a previous instruction still in the pipeline.

2.4.2 Execute Stage

During instruction execution, a pair of instructions (or a single instruction when there was a previous block) are individually passed to independent execution units. Each execution unit receives its operands from the register bypass logic and an instruction from the instruction scheduler. Each instruction spends one RUN cycle in an execution unit. For ALU and other single cycle instructions, the result is then fed to the register/bypass unit for the CR stage.

2.4.3 CR and WB Stages

For load and store instructions, the cache lookup occurs during the CR stage. For load instructions, data is returned to the register/bypass unit during the CR stage, including loads to coprocessors.

For all other instructions, CR and WB are holding stages used to hold the result of the execute stage for writeback to the register file.

2.5 Instruction Set Summary

Table 2.1 summarizes the instruction set for the CW4010. The CW4010 supports both MIPS-I and MIPS-II instructions, and also implements some additional CW4010-specific instructions. If the design includes the optional MMU, the CW4010 supports the TLB instructions. All instructions are 32 bits long. Table 2.1 includes only the MIPS-II, CW4010-specific, and TLB instructions. With the exception of RFE, MIPS-I instructions are not shown.

Table 2.1
CW4010 Instruction Set
Summary

Op	Description	Op	Description
Arithmetic Instructions: ALU Immediate			
ADDI	Add Immediate	ANDI	AND Immediate
ADDIU	Add Immediate Unsigned	ORI	OR Immediate
SLTI	Set on Less Than Immediate	XORI	Exclusive OR Immediate
SLTIU	Set on Less Than Immediate Unsigned	LUI	Load Upper Immediate
Arithmetic Instructions: Three-Operand, Register-Type			
ADD	Add	SLTU	Set on Less Than Unsigned
ADDU	Add Unsigned	AND	AND
SUB	Subtract	OR	OR
SUBU	Subtract Unsigned	XOR	Exclusive OR
SLT	Set on Less Than	NOR	NOR
Branch Likely Instructions			
BEQL[1]	Branch on Equal Likely	BGEZL[1]	Branch on Greater than or Equal to Zero Likely
BNEL[1]	Branch on Not Equal Likely	BLTZALL[1]	Branch on Less Than Zero And Link Likely
BLEZL[1]	Branch on Less than or Equal to Zero Likely	BGEZALL[1]	Branch on Greater than or Equal to Zero and Link Likely
BGTZL[1]	Branch on Greater Than Zero Likely	BCzTL[1]	Branch on Coprocessor z True Likely
BLTZL[1]	Branch on Less Than Zero Likely	BCzFL[1]	Branch on Coprocessor z False Likely
Coprocessor Instructions			
LWCz	Load Word to Coprocessor z	CFCz	Move Control from Coprocessor z
SWCz	Store Word from Coprocessor z	COPz	Coprocessor Operation
MTCz	Move to Coprocessor z	BCzT	Branch on Coprocessor z True
MFCz	Move From Coprocessor z	BCzF	Branch on Coprocessor z False
CTCz	Move Control to Coprocessor z		

(Sheet 1 of 3)

Table 2.1 (Cont.)
CW4010 Instruction Set
Summary

Op	Description	Op	Description
Jump and Branch Instructions			
J	Jump	BLEZ	Branch on Less than or Equal to Zero
JAL	Jump And Link	BGTZ	Branch on Greater than Zero
JR	Jump Register	BGEZ	Branch on Greater than or Equal to Zero
JALR	Jump And Link Register	BLTZAL	Branch on Less than Zero and Link
BEQ	Branch on Equal	BGEZAL	Branch on Greater than or Equal to Zero and Link
BNE	Branch on Note Equal		
Load/Store Instructions			
LB	Load Byte	SH	Store Halfword
LBU	Load Byte Unsigned	SW	Store Word
LH	Load Halfword	SWL	Store Word Left
LHU	Load Halfword Unsigned	SWR	Store Word Right
LW	Load Word	LL[1]	Load Linked
LWL	Load Word Left	SC[1]	Store Conditional
LWR	Load Word Right	SYNC[1]	Sync
SB	Store Byte		
Multiply/Divide Instructions			
MULT	Multiply	MTHI	Move To HI
MULTU	Multiply Unsigned	MFLO	Move From LO
DIV	Divide	MTLO	Move To LO
DIVU	Divide Unsigned	MIN	Select minimum value
MFHI	Move From HI	MAX	Select maximum value

(Sheet 2 of 3)

Table 2.1 (Cont.)
CW4010 Instruction Set
Summary

Op	Description	Op	Description
Other Computational Instructions			
ADDCIU[2]	Add Circular Immediate	SELSR[2]	Select and Shift Right
FFS[2]	Find First Set	SELSL[2]	Select and Shift Left
FFC[2]	Find First Clear	MADD[2]	Multiply/Add
FLUSHD[2, 3]	Flush Data Cache	MADDU[2]	Multiply/Add Unsigned
FLUSHI[2, 3]	Flush Instruction Cache	MSUB[2]	Multiply/Subtract
FLUSHID[2, 3]	Flush Instruction and Data Cache	MSUBU[2]	Multiply/Subtract Unsigned
Special Instructions			
SYSCALL	System Call	BREAK	Breakpoint
System Control Coprocessor (CP0) Instructions			
MTC0	Move To CP0	TLBWI[4]	Write Indexed TLB Entry
MFC0	Move From CP0	TLBWR[4]	Write Random TLB Entry
RFE	Restore From Exception (R3000 mode only)	TLBP[4]	Probe TLB for Matching Entry
ERET	Exception Return (R4000 mode only)	WAITI[2]	Wait for Interrupt
TLBR[4]	Read Indexed TLB Entry		

(Sheet 3 of 3)

1. MIPS II instruction
2. CW4010-specific instruction
3. Do not confuse these instructions with the FLUSH instruction in R6000 processors
4. Valid only with implemented MMU (Memory Management Unit) building block

In addition to the standard MIPS-II instruction set, the CW4010 implements certain instruction set extensions, shown in Table 2.2, that provide greater application code performance for typical embedded applications. Instruction set extensions are included only if they significantly improve performance, have no impact on clock cycle rate, and have minimal impact on the size and complexity of the hardware.

Table 2.2
Instruction Set
Extensions

Extension	Format and Description
Find First Set, Find First Clear	`ffs, ffc` These instructions, respectively, find the first set bit and the first clear bit in the source register, and return the bit number to the destination register. They are useful for many applications such as interrupt handlers, floating point emulation, and graphics.
Select and Rotate Left, Select and Rotate Right	`selsl, selsr` These instructions select 32 bits from the 64-bit source register pair and rotate the selected data left or right by the number of bits specified in the new CP0 Rotate register. They are useful for data alignment operation in graphics and in bit-field selection routines for data transmission and compression applications.
Add Circular Immediate	`addciu` This instruction does an immediate add, modified according to the value in the new CP0 CMask register. It is useful in addressing circular buffers. This instruction is important in DSP (Digital Signal Processing) and other applications that use circular buffers.
Multiply/ADD, Multiply/SUB Instructions	`madd, msub` These instructions are useful in many signal processing and graphics transform algorithms. Only implemented with the high-performance multiply/accumulate unit, these instructions do a 32 x 32 multiply and then either add or subtract the result to the 64-bit HI/LO register pair.
Wait for Interrupt	`waiti` This instruction halts the CPU in a power saving mode until one of the hardware interrupt lines becomes active. Upon interrupt, normal execution is resumed starting at the interrupt vector address.
Minimum rd, rs, rt.	`min rd, rs, rt` The source operands `rs` and `rt` are compared as two's complement values. The smaller value is stored in the `rd` register.
Maximum rd, rs, rt.	`max rd, rst, rt` The source operands `rs` and `rt` are compared as two's complement values. The larger value is stored in the `rd` register.

2.6 Configurability and Options

The CW4010 is implemented using Verilog HDL (Hardware Description Language) as the design source, and the LSI Logic Standard Cell Library and layout tools for physical design. You can easily modify and configure the CW4010 core to meet specific design requirements. The options available in the basic core are shown in the following sections.

Note: VHDL models are also available.

2.6.1 Cache Sizes

The instruction cache sizes available are 0 Kbytes to 16 Kbytes, direct mapped or two-way set associative. The data cache sizes available are 0 Kbytes to 16 Kbytes, direct mapped or two-way set associative.

2.6.2 Standard vs High-Performance Multiply Accumulate Unit

Each project may choose either a standard multiply unit that provides the base R3000 and R4000 multiply instructions (with similar performance), or a high-performance unit that also implements the madd and msub instructions. In the standard unit, multiples are executed by retiring two bits per clock cycle, with early termination that depends on the size of the multiplican (8 x 32 bit multiply = 4 cycles, 16 x 32 = 8 cycles, 32 x 32 = 16 cycles).

The high-performance unit is intended for applications with substantial multiply/accumulate performance needs. It includes a 32 x 32 pipelined array multiplier and a 64-bit accumulator that can retire a multiply or multiply/accumulate instruction every two clock cycles with a latency of three clock cycles per result. The highest multiply/accumulate unit can retire a multiply/accumulate instruction every single clock cycle, with a latency of two clock cycles per result.

2.6.3 64-bit vs 32-bit Memory Interface

For cost-sensitive designs or applications with low memory bandwidth, the BIU can be modified to present a 32-bit data bus instead of 64-bit. The CW4010 BIU supports sizing for a 32-bit interface.

2.6.4 Memory Management Unit

The CW4010 is designed to support the 32-bit addressing mode of the R4000 MMU. The TLB that is available in the base processor design contains up to 64 single-page entries. Each page can be individually specified to be 4 Kbytes or 16 Mbytes. For designs with no TLB requirements, the TLB can be removed to save silicon.

2.7 Supporting Models and Tools

A software model, MiniSIM, is available for software development and performance modeling. The model is a stand-alone C program. It provides a software level model and the LSI Logic PMON and C runtime library that is supplied with LSI Logic's hardware evaluation boards. You can run R3000 or R4000 compiled code on the model, using the PMON debugging environment. You can obtain performance data for benchmarks and code tuning. Currently, MiniSIM is available for Sun SPARCstation. Contact LSI Logic for versions supporting other platforms.

For use later in the development cycle, but before production of silicon, LSI Logic provides a behavioral Verilog model that is pin compatible and cycle accurate with the real design without the timing. You can use the model for behavioral modeling of your system design. A VHDL version of the model is also available.

The gate level model is available in a variety of simulation environments for timing accurate simulations. For faster synthesis of logic surrounding the CPU core, LSI Logic provides a “synthesis shell.” This includes all drive capability, input loading, and timing information. It excludes the considerable complexity of the core internals, since these are of no relevance to synthesis tools.

As the design nears completion, timing analysis may be performed using a “timing shell.” This includes the same set of information as the synthesis shell, provided in a format directly usable with supported static Timing Analysis tools. Again, the core internals are of no relevance to this process, and would only reduce its efficiency.

Structural core models are provided for final versions of the design in various supported simulation tools. Contact LSI Logic for availability of the version supporting your preferred simulator.

You can use the full range of MIPS and third party software development tools for the R3000 and R4000 on the CW4010 product. The CPU is compatible with both MIPS-I and MIPS-II compilers and assemblers.

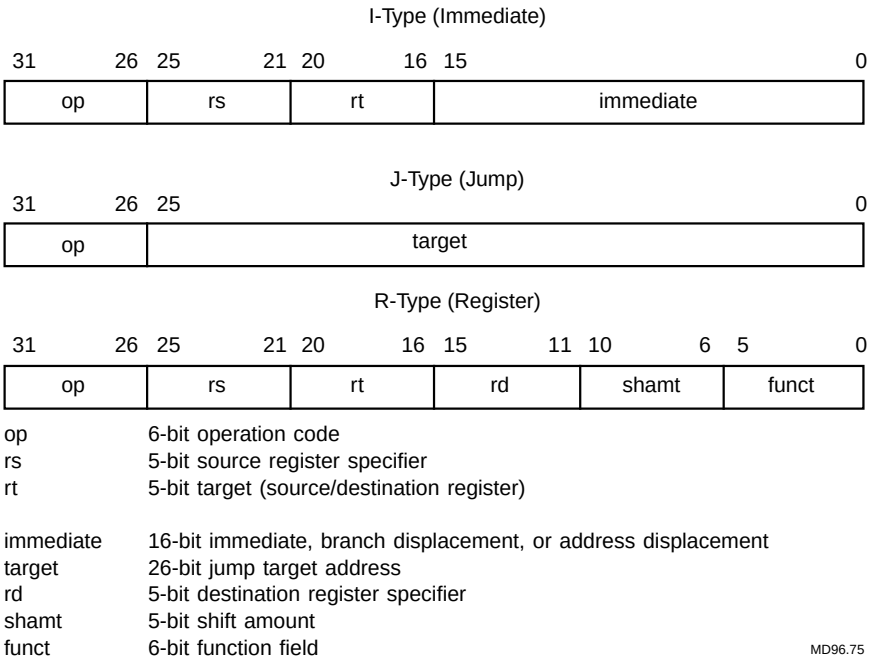
Chapter 3
Instruction Set Summary

This chapter presents an overview of the MIPS R-series instructions and the instruction set extensions supported in the CW4010. This chapter contains the following sections:

3.1 Instruction Set Formats

Every R-Series instruction consists of a single word (32 bits) aligned on a word boundary. As shown in Figure 3.1, there are three instruction formats: *I-type* (immediate), *J-type* (jump), and *R-type* (register). The restricted format approach simplifies instruction decoding. The compiler and assembler can synthesize more complicated (and less frequently used) operations and addressing modes.

Figure 3.1
Instruction Format



op	6-bit operation code
rs	5-bit source register specifier
rt	5-bit target (source/destination register)
immediate	16-bit immediate, branch displacement, or address displacement
target	26-bit jump target address
rd	5-bit destination register specifier
shamt	5-bit shift amount
funct	6-bit function field

MD96.75

3.2 Load and Store Instructions

Load and store instructions are all I-type instructions and move data between memory and general purpose registers. The only addressing mode directly supported in the base R-Series architecture is base register plus 16-bit signed immediate `offset`.

The MIPS-II extensions add the Load-Linked and Store-Conditional instructions, which support multiple processors, and the SYNC instruction, which synchronizes loads and stores. The CW4010 supports these instructions.

The load/store instruction operation code (opcode) determines the access type, which in turn indicates the size of the data item to be loaded or stored. Regardless of access type or byte-numbering order (**big-endian** or **little-endian**), the address specifies the byte that has the smallest byte address of all the bytes in the addressed field. For a big-endian machine, the smallest byte is the leftmost byte; for a little-endian machine, it is the rightmost byte.

The bytes used within the addressed word can be determined directly from the access type and the two low-order bits of the address, as shown

in Figure 3.2. Note that certain combinations of access type and low-order address bits can never occur; only the combinations shown in Figure 3.2 are allowed.

Figure 3.2
Byte Specifications for Loads/Stores

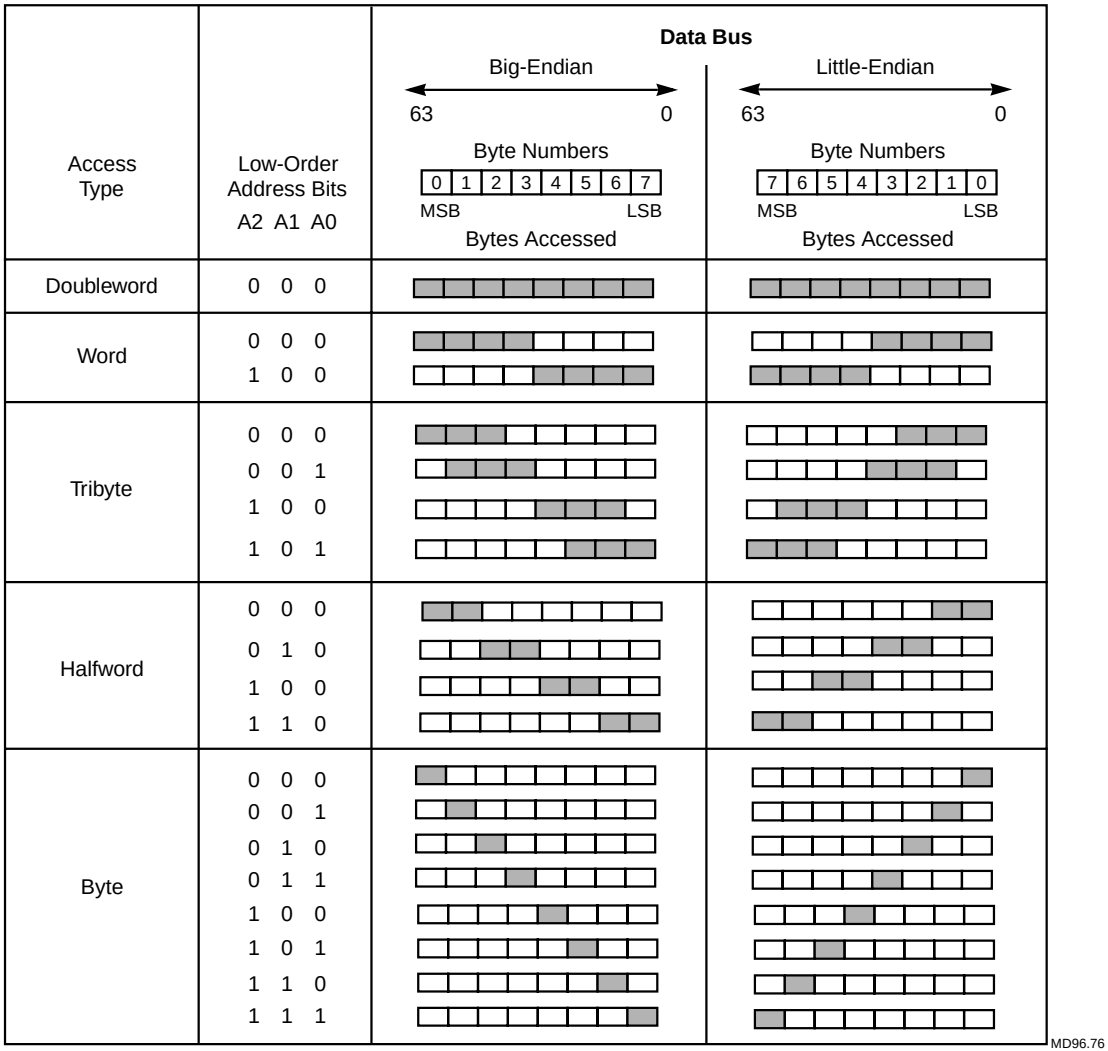


Table 3.1 and Table 3.2 describe the load and store instructions supported by the CW4010. Instruction format is shown in courier, for example, `LB rt, offset(base)`.

Table 3.1
Load and Store
Instruction Summary

Instruction	Format and Description
Load Byte	`LB rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Sign-extend the contents of addressed byte and load into `rt`.
Load Byte Unsigned	`LBU rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Zero-extend the contents of addressed byte and load into `rt`.
Load Halfword	`LH rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Sign-extend contents of addressed halfword and load into `rt`.
Load Halfword Unsigned	`LHU rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Zero-extend contents of addressed halfword and load into `rt`.
Load Word	`LW rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address, and load the addressed word into `rt`.
Load Word Left	`LWL rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Shift addressed word left so that addressed byte is leftmost byte of a word. Merge bytes from memory with contents of register `rt` and load result into register `rt`.
Load Word Right	`LWR rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Shift addressed word right so that addressed byte is rightmost byte of a word. Merge bytes from memory with contents of register `rt` and load result into register `rt`.
Store Byte	`SB rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Store least-significant byte of register `rt` at addressed location.
Store Halfword	`SH rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Store least-significant halfword of register `rt` at addressed location.
Store Word	`SW rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Store contents of register `rt` at addressed location.
Store Word Left	`SWL rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Shift contents of register `rt` left so that the leftmost byte of the word is in the position of the addressed byte. Store word containing shifted bytes into word at addressed byte.

Table 3.1
Load and Store
Instruction Summary

Instruction	Format and Description
Store Word Right	`SWR rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Shift contents of register `rt` right so that the rightmost byte of the word is in the position of the addressed byte. Store word containing shifted bytes into word at addressed byte.

Table 3.2
Load and Store
Instruction Summary
(MIPS-II ISA
Extensions)

Instruction	Format and Description
Load Linked	`LL rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address, and load the addressed word into register `rt`.
Store Conditional	`SC rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Conditionally store register `rt` at address, based on whether the load-link has been “broken.”
Sync	`SYNC` Complete all outstanding load and store instructions before allowing any new load or store instruction to start.

3.3 Computational Instructions

Computational instructions perform arithmetic, logical, and shift operations on values in registers. Computational instructions occur in both R-type (both operands are registers) and I-type (one operand is a 16-bit immediate) formats. There are five categories of computational instructions:

- Table 3.3 summarizes the ALU Immediate instructions
- Table 3.4 summarizes the 3-Operand, Register-Type instructions
- Table 3.5 summarizes the Shift instructions
- Table 3.6 summarizes the Multiply/Divide instructions
- Table 3.7 summarizes the Computational CW4010 Instruction Extensions (CW4010 ISA)

Table 3.3
ALU Immediate
Instruction Summary

Instruction	Format and Description
Add Immediate	`ADDI rt, rs, immediate` Add 16-bit, sign-extended `immediate` to register `rs` and place 32-bit result in register `rt`. Trap on two's complement overflow.
Add Immediate Unsigned	`ADDIU rt, rs, immediate` Add 16-bit, sign-extended `immediate` to register `rs` and place 32-bit result in register `rt`. Do not trap on overflow.
Set on Less Than Immediate	`SLTI rt, rs, immediate` Compare 16-bit, sign-extended `immediate` with register `rs` as signed 32-bit integers. Result = 1 if `rs` is less than immediate; otherwise result = 0. Place result in register `rt`.
Set on Less Than Immediate Unsigned	`SLTIU rt, rs, immediate` Compare 16-bit, sign-extended `immediate` with register `rs` as unsigned 32-bit integers. Result = 1 if `rs` is less than immediate; otherwise result = 0. Place result in register `rt`.
AND Immediate	`ANDI rt, rs, immediate` Zero-extend 16-bit `immediate`, AND with contents of register `rs`, and place result in register `rt`.
OR Immediate	`ORI rt, rs, immediate` Zero-extend 16-bit `immediate`, OR with contents of register `rs`, and place result in register `rt`.
Exclusive OR Immediate	`XORI rt, rs, immediate` Zero-extend 16-bit `immediate`, exclusive OR with contents of register `rs`, and place result in register `rt`.
Load Upper Immediate	`LUI rt, immediate` Shift 16-bit `immediate` left 16 bits. Set least-significant 16 bits of word to zeros. Store result in register `rt`.

Table 3.4
3-Operand, Register-Type Instruction Summary

Instruction	Format and Description
Add	`ADD rd, rs, rt` Add contents of registers `rs` and `rt` and place 32-bit result in register `rd`. Trap on two's complement overflow.
Add Unsigned	`ADDU rd, rs, rt` Add contents of registers `rs` and `rt` and place 32-bit result in register `rd`. Do not trap on overflow.
Subtract	`SUB rd, rs, rt` Subtract contents of register `rt` from `rs` and place 32-bit result in register `rd`. Trap on two's complement overflow.
Subtract Unsigned	`SUBU rd, rs, rt` Subtract contents of register `rt` from `rs` and place 32-bit result in register `rd`. Do not trap on overflow.
Set on Less Than	`SLT rd, rs, rt` Compare contents of register `rt` to register `rs` (as signed, 32-bit integers). If register `rs` is less than `rt`, `rd` = 1; otherwise, `rd` = 0.
Set on Less Than Unsigned	`SLTU rd, rs, rt` Compare contents of register `rt` to register `rs` (as unsigned, 32-bit integers). If register `rs` is less than `rt`, `rd` = 1; otherwise, `rd` = 0.
AND	`AND rd, rs, rt` Bitwise AND contents of registers `rs` and `rt` and place result in register `rd`.
OR	`OR rd, rs, rt` Bitwise OR contents of registers `rs` and `rt` and place result in register `rd`.
Exclusive OR	`XOR rd, rs, rt` Bitwise exclusive OR contents of registers `rs` and `rt` and place result in register `rd`.
NOR	`NOR rd, rs, rt` Bitwise NOR contents of registers `rs` and `rt` and place result in register `rd`.

Table 3.5
Shift Instruction
Summary

Instruction	Format and Description
Shift Left Logical	`SLL rd, rt, shamt` Shift contents of register `rt` left by `shamt` bits, inserting zeros into low-order bits. Place 32-bit result in register `rd`.
Shift Right Logical	`SRL rd, rt, shamt` Shift contents of register `rt` right by `shamt` bits, inserting zeros into high-order bits. Place 32-bit result in register `rd`.
Shift Right Arithmetic	`SRA, rd, rt, shamt` Shift contents of register `rt` right by `shamt` bits, sign-extending the high-order bits. Place 32-bit result in register `rd`.
Shift Left Logical Variable	`SLLV rd, rt, rs` Shift contents of register `rt` left. Low-order 5 bits of register `rs` specify the number of bits to shift. Insert zeros into low-order bits of `rt` and place 32-bit result in register `rd`.
Shift Right Logical Variable	`SRLV rd, rt, rs` Shift contents of register `rt` right. Low-order 5 bits of register `rs` specify the number of bits to shift. Insert zeros into high-order bits of `rt` and place 32-bit result in register `rd`.
Shift Right Arithmetic Variable	`SRAV rd, rt, rs` Shift contents of register `rt` right. Low-order 5 bits of register `rs` specify the number of bits to shift. Sign-extend the high-order bits of `rt` and place 32-bit result in register `rd`.

Table 3.6
Multiply/Divide
Instruction Summary

Instruction	Format and Description
Multiply	`MULT rs, rt` Multiply contents of registers `rs` and `rt` as two's complement values. Place 64-bit results in special registers HI and LO.
Multiply Unsigned	`MULTU rs, rt` Multiply contents of registers `rs` and `rt` as unsigned values. Place 64-bit results in special registers HI and LO.
Divide	`DIV rs, rt` Divide contents of register `rs` by the contents of `rt` as two's complement values. Place 32-bit quotient in special register LO and 32-bit remainder in HI.
Divide Unsigned	`DIVU rs, rt` Divide contents of register `rs` by the contents of `rt` as unsigned values. Place 32-bit quotient in special register LO and 32-bit remainder in HI.
Move From HI	`MFHI rd` Move contents of special register HI to register `rd`.
Move From LO	`MFLO rd` Move contents of register `rs` to special register LO.
Move To HI	`MTHI rs` Move contents of register `rs` to special register HI.
Move To LO	`MTLO rs` Move contents of register `rd` to special register LO.
Minimum	`MIN rd, rs, rt` Compare the contents of registers `rs` and `rt` as two's complement values. The smaller value is stored in register `rd`.
Maximum	`Max rd, rs, rt` Compare the contents of registers `rs` and `rt` as two's complement values. The larger value is stored in register `rd`.

Table 3.7
Computation Instruction
Extensions Summary
(CW4010 ISA)

Instruction	Format and Description
Add Circular Immediate	`ADDCIU rt, rs, immediate` The 16-bit `immediate` is sign-extended and added to the contents of general register `rs`, with the result masked by the value in CP0 register CMASK according to the formula: `rt = (rs`$_{31...cmask}$`||(rs+signextended_imed)`$_{cmask-1...0}$).
Find First Set Bit	`FFS rd, rs` Starting at the most significant bit in register `rs`, find the first bit that is set to a one, and return the bit number in register `rd`. If no bit is set, return with all bits of `rd` set to 1.
Find First Clear Bit	`FFC rd, rs` Starting at the most significant bit in register `rs`, find the first bit that is set to a zero, and return the bit number in register `rd`. If no bit is set, return with all bits of `rd` set to 1.
Select and Shift Right	`SELSR rd, rs, rt` Using register `rs` and `rt` as a 64-bit register pair and the CP0 register ROTATE as the shift count, shift the register pair `rs||rt` right the number of bits specified in ROTATE, and place the least significant 32-bit value in result register `rd`.
Select and Shift Left	`SELSL rd, rs, rt` Using register `rs` and `rt` as a 64-bit register pair and the CP0 register ROTATE as the shift count, shift the register pair `rs||rt` left the number of bits specified in ROTATE, and place the most significant 32-bit value in result register `rd`.
Multiply/Add	`MADD rs, rt` Multiply contents of registers `rs` and `rt` as two's complement values. Add 64-bit results to contents in special register pair HI/LO, and place results in HI and LO.
Multiply/Add Unsigned	`MADDU rs, rt` Multiply contents of registers `rs` and `rt` as unsigned values. Add 64-bit results to contents in special register pair HI/LO, and place results in HI and LO.
Multiply/Subtract	`MSUB rs, rt` Multiply contents of registers `rs` and `rt` as two's complement values. Subtract 64-bit results from contents in special register pair HI/LO, and place results in HI and LO.
Multiply/Subtract Unsigned	`MSUBU rs, rt` Multiply contents of registers `rs` and `rt` as unsigned values. Subtract 64-bit results from contents in special register pair HI/LO, and place results in HI and LO.

Table 3.8 shows the execution time of the multiply/divide/accumulate type instructions.

Table 3.8
Execution Time of Multiply and Divide Instructions

Operation	R3000	CW33300	R4000	CW4010 High Speed	CW4010 Basic
Multiply	12	1 + (bits/3)	10	3	1 + (bits/2)
Multiply/Add	na	na	na	3[1]	1 + (bits/2)
Divide	34	34	69	34/17[2]	35

1. For high-speed CW4010 multiply/add instructions, instructions can be pipelined for a throughput of one operation every clock cycle while the latency is three cycles. Pipelining the instructions accelerates calculations such as dot products and FIR filters that perform a series of multiplies/adds to compute a single result.
2. The divide time is shortened to 17 cycles if the divisor has less than 16 significant bits.

3.4 Jump and Branch Instructions

Jump and branch instructions change the control flow of a program. MIPS-I jump and branch instructions always occur with a one-instruction delay. The instruction immediately following the jump or branch is always executed while the target instruction is being fetched from storage. There may be additional cycle penalties, depending on circumstances and implementation, but the penalties are interlocked in hardware. The MIPS-II ISA extensions add the branch likely class of instructions that operate exactly like their non-likely counterparts, except that when the branch is *not* taken, the instruction following the branch is cancelled.

The J-type instruction format is used for both jump and jump-and-link instructions for subroutine calls. In the J-type format, the 26-bit target address is shifted left two bits and combined with the 4 high-order bits of the current program counter to form a 32-bit absolute address.

The R-type instruction format, which takes a 32-bit byte address contained in a register, is used for returns, dispatches, and cross-page jumps.

Branches have 16-bit signed offsets relative to the program counter (I-type). Jump-and-link and branch-and-link instructions save a return address in register 31.

Table 3.9 summarizes the R-Series jump instructions, Table 3.10 summarizes the branch instructions, and Table 3.11 summarizes the BranchLikely instructions.

Table 3.9
Jump Instruction Summary

Instruction	Format and Description
Jump	`J target` Shift 26-bit `target` address left two bits, combine with four high-order bits of PC, and jump to address with a one-instruction delay.
Jump and Link	`JAL target` Shift 26-bit `target` address left two bits, combine with four high-order bits of PC, and jump to address with a one-instruction delay. Place address of instruction following delay slot in register 31 (link register).
Jump Register	`JR rs` Jump to address contained in register `rs` with a one-instruction delay.
Jump and Link Register	`JALR rs, rd` Jump to address contained in register `rs` with a one-instruction delay. Place address of instruction following delay slot in `rd`.

Table 3.10
Branch Instruction
Summary

Instruction	Format and Description
Branch on Equal	`BEQ rs, rt, offset` Branch to target address[1] if register `rs` is equal to register `rt`.
Branch on Not Equal	`BNE rs, rt, offset` Branch to target address if register `rs` does not equal register `rt`.
Branch on Less than or Equal to Zero	`BLEZ rs, offset` Branch to target address if register `rs` is less than or equal to 0.
Branch on Greater Than Zero	`BGTZ rs, offset` Branch to target address if register `rs` is greater than 0.
Branch on Less Than Zero	`BLTZ rs, offset` Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero	`BGEZ rs, offset` Branch to target address if register `rs` is greater than or equal to 0.
Branch on Less Than Zero And Link	`BLTZAL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero and Link	`BGEZAL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is greater than or equal to 0.

1. All branch-instruction target addresses are computed as follows: add address of instruction in delay slot and the 16-bit `offset` (shifted left two bits and sign-extended to 32 bits). All branches occur with a delay of one instruction.

Table 3.11
BranchLikely Instruction
Summary (MIPS-II ISA
Extensions)

Instruction	Format and Description
Branch on Equal Likely	`BEQL rs, rt, offset` Branch to target address[1] if register `rs` is equal to register `rt`.
Branch on Not Equal Likely	`BNEL rs, rt, offset` Branch to target address if register `rs` does not equal register `rt`.
Branch on Less than or Equal to Zero Likely	`BLEZL rs, offset` Branch to target address if register `rs` is less than or equal to 0.
Branch on Greater Than Zero Likely	`BGTZL rs, offset` Branch to target address if register `rs` is greater than 0.
Branch on Less Than Zero Likely	`BLTZL rs, offset` Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero Likely	`BGEZL rs, offset` Branch to target address if register `rs` is greater than or equal to 0.
Branch on Less Than Zero And Link Likely	`BLTZALL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero and Link Likely	`BGEZALL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is greater than or equal to 0.

1. All branch-instruction target addresses are computed as follows: add address of instruction in delay slot and the 16-bit `offset` (shifted left two bits and sign-extended to 32 bits). All branches occur with a delay of one instruction.

3.5 Trap Instructions

Trap instructions are part of the MIPS-II instruction set and provide instructions that conditionally create an exception, based on the same conditions tested in the branch instructions. Table 3.12 provides a summary of MIPS-II ISA extensions.

Table 3.12 Trap Instruction Summary (MIPS-II ISA Extensions)

Instruction	Format and Description
Trap on Equal	`TEQ rs, rt` Trap if register `rs` is equal to register `rt`.
Trap on Equal Immediate	`TEQI rs, immediate` Trap if register `rs` is equal to the `immediate` value.
Trap on Greater than or Equal	`TGE rs, rt` Trap if register `rs` is greater than or equal to register `rt`.
Trap on Greater Than or Equal Immediate	`TGEI rs, immediate` Trap if register `rs` is greater than or equal to the `immediate` value.
Trap on Greater than or Equal Unsigned	`TGEU rs, rt` Trap if register `rs` is greater than or equal to register `rt`.
Trap on Greater Than or Equal Immediate Unsigned	`TGEIU rs, immediate` Trap if register `rs` is greater than or equal to the `immediate` value.
Trap on Less Than	`TLT rs, rt` Trap if register `rs` is less than register `rt`.
Trap on Less Than Immediate	`TLTI rs, immediate` Trap if register `rs` is less than the `immediate` value.
Trap on Less Than Unsigned	`TLTU rs, rt` Trap if register `rs` is less than register `rt`.
Trap on Less Than Immediate Unsigned	`TLTIU rs, immediate` Trap if register `rs` is less than the `immediate` value.
Trap If Not Equal	`TNE rs, rt` Trap if register `rs` is not equal to `rt`.
Trap If Not Equal Immediate	`TNEI rs, immediate` Trap if register `rs` is not equal the `immediate` value.

3.6 Special Instructions

Special instructions cause an unconditional branch to the general exception-handling vector. Special instructions are always R-type and are summarized in Table 3.13.

Table 3.13
Special Instruction Summary

Instruction	Format and Description
System Call	SYSCALL Initiates system call trap, immediately transferring control to exception handler.
Breakpoint	BREAK Initiates breakpoint trap, immediately transferring control to exception handler.

3.7 Coprocessor Instructions

The CW4010 supports external (on-chip) coprocessors and implements the coprocessor instruction set. Coprocessor branch instructions are J-type. Table 3.14 summarizes the different coprocessor instructions.

Table 3.14
Coprocessor Instruction Summary

Instruction	Format and Description
Load Word to Coprocessor	`LWCz rt, offset(base)` Extends the sign of the 16-bit `offset` and adds the `offset` to the contents of the general register base to form a 32-bit unsigned effective address. The word at the memory location specified is loaded into coprocessor register `rt` of the coprocessor unit `z`.
Store Word from Coprocessor	`SWCz rt, offset(base)` Extends the sign of the 16-bit `offset` and adds the `offset` to the contents of the general register base to form a 32-bit unsigned effective address. The contents of coprocessor register `rt` of the coprocessor unit `z` are stored at the address specified by the 32-bit unsigned effective address.
Move To Coprocessor	`MTCz rt, rd` Loads the contents of general register `rt` into the `rd` register of coprocessor unit `z`.
Move From Coprocessor	`MFCz rt, rd` Loads the contents of the `rd` register of coprocessor unit `z` into general register `rt`.
Move Control to Coprocessor	`CTCz rt, rd` Loads the contents of general register `rt` into the control register `rd` of coprocessor unit `z`.
Move Control From Coprocessor	`CFCz rt, rd` Loads the contents of the control register `rd` of coprocessor unit `z` into general register `rt`.
Coprocessor Operation	`COPz cofun` Initiates a coprocessor operation that may specify and reference the coprocessor's internal registers or change the state of the coprocessor's condition line, but does not change the state within the processor or the cache memory.
Branch on Coprocessor z True (Likely)	`BCzT offset, (BCzTL offset)` Compute a branch target address by adding address of instruction to the 16-bit `offset` (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction) if coprocessor z's condition line is true. In the case of BranchLikely, the delay slot instruction is not executed when the branch is not taken.
Branch on Coprocessor z False (Likely)	`BCzF offset, (BCzFL offset)` Compute a branch target address by adding address of instruction to the 16-bit `offset` (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction) if coprocessor z's condition line is false. In the case of BranchLikely, the delay slot instruction is not executed when the branch is not taken.

3.8 System Control Coprocessor (CP0) Instructions

Coprocessor-0 instructions perform operations on the system control coprocessor (CP0) registers to manipulate the memory management and exception-handling facilities of the processor. Table 3.15 summarizes the CP0 instructions and Table 3.16 shows the extensions.

If the TLB is removed, the TLB instructions (TLBR, TLBWI, TLBWRm TLBP) cause an RI (Reserved Instruction) exception. If the CW4010 is in R3000 compatibility mode, the ERET (Exception Returned) instruction is unavailable and this causes an RI exception. Conversely, if the CW4010 is in R4000 mode, the RFE (Restore From Exception) instruction is unavailable and this causes an RI exception.

Table 3.15
CP0 Instruction Summary

Instruction	Format and Description
Move To CP0	`MTC0 rt, rd` Loads contents of CPU register `rt` into CP0 register `rd`.
Move From CP0	`MFC0 rt, rd` Loads contents of CP0 register `rd` into CPU register `rt`.
Read Indexed TLB Entry[1]	`TLBR` Loads EntryHi and EntryLo with the TLB entry pointed to by the Index register.
Write Indexed TLB Entry[1]	`TLBWI` Loads TLB entry pointed to by the Index register with the contents of the EntryHi and EntryLo registers.
Write Random TLB Entry[1]	`TLBWR` Loads TLB entry pointed to by the Random register with the contents of the EntryHi and EntryLo registers.
Probe TLB for Matching Entry[1]	`TLBP` Loads the Index register with the address of the TLB entry whose contents match the EntryHi and EntryLo registers. If no TLB entry matches, set the high-order bit of the Index register.
Exception Return[2]	`ERET` (R4000 Mode) Loads the PC from ErrorEPC (SR2 = 1: Error Exception) or EPC (SR2 = 0: Exception) and clear ERL bit (SR2 = 1) or EXL bit (SR2 = 0) in the Status Register. SR2 is Status register bit 2.
Restore From Exception[2]	`RFE` (R3000 Mode) Restores previous interrupt mask and mode bits of the Status register into current status bits. Restore old status bits into previous status bits.

1. If there is no MMU (Memory Management Unit) installed, any of these instructions can cause a reserved instruction exception.
2. Only one of these instructions is legal at any one time. The one that is not legal causes a reserved instruction exception.

Table 3.16
CP0 Instruction
Extension Summary

Instruction	Format and Description
Wait for Interrupt	`WAITI` Stops execution of instructions and places the processor into a power save (stall) condition until a hardware interrupt, NMI, or reset is received.

3.9 Cache Maintenance Instructions

Cache Maintenance instructions are always I-type. Table 3.17 summarizes these instructions.

Table 3.17
Cache Maintenance
Instruction Summary

Instruction	Format and Description
Flush Icache	`FLUSHI` Flush Icache needs 256 stall cycles.
Flush Dcache	`FLUSHD` Flush Dcache needs 256 stall cycles.
Flush Icache & Dcache	`FLUSHID` Flush both Icache and Dcache in 256 stall cycles.
WriteBack	`WB offset(base)` Write back a Dcache line addressed by `offset+GPR[base]`.

3.10 CW4010 Instruction Set Extensions

This section defines the CW4010 instruction set extensions.

ADDCIU — Add with Circular Mask Immediate

Format

31 26	25 21	20 16	15 0
ADDCIU	rs	rt	immediate
011100	rs	rt	immediate

Syntax **`ADDCIU rt, rs, immediate`**

Description The immediate field of the instruction is sign-extended and added to the contents of general register `rs`, the result of which is masked with the expanded value in special register CMask according to the equation shown below. The CMask register is CP0 register number 24, whose valid bits are [4:0].

The carries resulting from the addition of the sign-extended `offset` are not propagated into the final result beyond bit [CMask-1].

Operation

```
T: sign_extend_immed = (immediate15)16 || immediate15..0
   GPR[rt] = GPR[rs]31..cmask || (GPR[rs] +
    sign_extend_immed)cmask-1..0
```

Exceptions None

FFC **Find First Clear Bit**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	0	rd	0	FFC
000000	rs	0	rd	00000	001011

Syntax **`FFC rd, rs`**

Description The contents of general register `rs` are examined starting with the most significant bit. The bit number of the first clear bit is returned in general register `rd`. If no bit is set, all ones are returned in `rd`.

Exceptions None

FFS **Find First Set Bit**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	0	rd	0	FFS
000000	rs	0	rd	00000	001010

Syntax **`FFS rd, rs`**

Description The contents of general register `rs` are examined starting with the most significant bit. The bit number of the first set bit is returned in general register `rd`. If no bit is set, all ones are returned in `rd`.

Exceptions None

FLUSHD **FLUSH Data Cache**

Format

31 26	25 21	20 16	15 0
CACHE	0	FLUSHD	0
101111	00000	00010	0

Syntax **FLUSHD**

Description FLUSHD flushes all Data Cache lines and causes stall cycles for 256 clocks, regardless of the cache size.

Exceptions None

FLUSHI **FLUSH Instruction Cache**

Format

31 – 26	25 – 21	20 – 16	15 – 0
CACHE	0	FLUSHI	0
101111	00000	00001	0

Syntax **FLUSHI**

Description FLUSHI flushes all Instruction Cache lines and causes stall cycles for 256 clocks, regardless of the cache size.

Exceptions None

FLUSHID **FLUSH Instruction and Data Cache**

Format

31 26	25 21	20 16	15 0
CACHE	0	FLUSHID	0
101111	00000	00011	0

Syntax **`FLUSHID`**

Description FLUSHID flushes all Data and Instruction Cache lines and causes stall cycles for 256 clocks, regardless of the cache size.

Exceptions None

MADD **Multiply Add**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	MADD
000000	rs	rt	0	00000	011100

Syntax **`MADD rs, rt`**

Description The contents of general register `rs` and the contents of general register `rt` are multiplied. Both operands are treated as 32-bit two's complement values. When the operation is completed, the doubleword result is added to special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the Cache Configuration and Control (CCC) register.

MADD executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.19 on page 3-37.

Operation

```
T:      t <- (HI || LO) + (GPR[rs] * GPR[rt])
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

MADDU **Multiply Add Unsigned**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	MADDU
000000	rs	rt	0	00000	011101

Syntax **`MADDU rs, rt`**

Description The contents of general register `rs` and the contents of general register `rt` are multiplied with both operands treated as 32-bit unsigned values. When the operation is completed, the doubleword result is added to special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the CCC register.

The instruction executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.19 on page 3-37.

Operation

```
T:      t <- (HI || LO) + ((0||GPR[rs]) * (0||GPR[rt]))
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

MAX **Maximum**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	MAX
000000	rs	rt	rd	00000	101001

Syntax **`MAX rd, rs, rt`**

Description The source operands `rs` and `rt` are compared as two's complement values. The larger value is stored in the `rd` register.

Operation

```
T:      if GPR[rs]>GPR[rt] then
              GPR[rd]<-GPR[rs]
        else
              GPR[rd]<-GPR[rt]
        endif
```

Exceptions None

MIN **Minimum**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	MIN
000000	rs	rt	rd	00000	101000

Syntax **`MIN rd, rs, rt`**

Description The source operands `rs` and `rt` are compared as two's complement values. The smaller value is stored in the `rd` register.

Operation

```
T:      if GPR[rs]<GPR[rt] then
              GPR[rd]<-GPR[rs]
        else
              GPR[rd]<-GPR[rt]
        endif
```

Exceptions None

MSUB **Multiply Subtract**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	MSUB
000000	rs	rt	0	00000	011110

Syntax **`MSUB rs, rt`**

Description The contents of general register `rs` and `rt` are multiplied and both operands are treated as 32-bit two's complement values. When the operation is complete, the doubleword result is subtracted from special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the CCC register.

The instruction executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.19 on page 3-37.

Operation

```
T:      t <- (HI || LO) - (GPR[rs] * GPR[rt])
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

MSUBU **Multiply Subtract Unsigned**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	MSUBU
000000	rs	rt	0	00000	011111

Syntax **`MSUBU rs, rt`**

Description The contents of general register `rs` and `rt` are multiplied and both operands are treated as 32-bit unsigned values. When the operation is completed, the doubleword result is subtracted from special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the CCC register.

The instruction executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.19 on page 3-37.

Operation

```
T:      t <- (HI || LO) - ((0||GPR[rs]) * (0||GPR[rt]))
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

SELSL **Select and Shift Left**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SELSL
000000	rs	rt	rd	00000	000101

Syntax **SELSL rd, rs, rt**

Description The contents of general register rs and rt are combined to form a 64-bit doubleword. The doubleword is shifted left the number of bits specified in the CP0 register ROTATE, and the upper 32 bits of the result are placed in general register rd. This ROTATE register is CP0 register number 23, with valid bits [4:0].

Operation

```
T:      s <- ROTATE4..0
        GPR[rd] <- GPR[rs]31-s..0 || GPR[rt]31..32-s
```

Exceptions None

SELSR **Select and Shift Right**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SELSR
000000	rs	rt	rd	00000	000001

Syntax **SELSR rd, rs, rt**

Description The contents of general register `rs` and `rt` are combined to form a 64-bit doubleword. The doubleword is shifted right the number of bits specified in CP0 register ROTATE, and the lower 32 bits of the result are placed in general register `rd`. This ROTATE register is CP0 register number 23. Valid bits are [4:0].

Operation

```
T:      s <- ROTATE4..0
        GPR[rd] <- GPR[rs]s-1..0 || GPR[rt]31..s
```

Exceptions None

WAITI **Wait for Interrupt**

Format

31 26	25 21	20 16	15 11	10 6	5 0
COP0		0	0	0	WAITI
010000	10000	00000	00000	00000	100000

Syntax **`WAITI`**

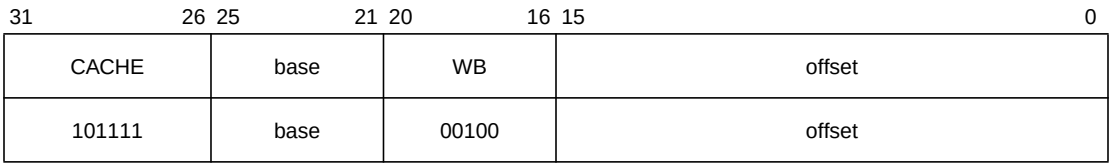
Description When this instruction is executed, the main processor clock stops and execution of instructions is halted. Execution resumes when a hardware interrupt, NMI, or reset exception is received. While it is in wait mode, the processor is in a power saving mode, using very little current because the clock is turned off to most of the circuitry.

WAITI must be followed by two or more No-Operation instructions, otherwise, the results may be undefined. Refer to Appendix A, "Programmer's Notes," for further information.

Exceptions None

WB **WriteBack Data Cache**

Format



Syntax **`WB offset(base)`**

Description Eight words of the Data cache line addressed by `offset +GPR[base]` are written back to memory if the line is dirty. Upper bits of `offset +GPR[base]` are ignored.

Exceptions None

3.11 CPU Instruction Opcode Bit Encoding

Tables 3.18 through 3.24 show the opcode bit encoding for CW4010 instructions. The following keys are referenced in the table:

***rxf1** Operation codes marked with *rxf1 cause reserved instruction exceptions in all current implementations and are reserved for future versions of the architecture.

***rxf2** Operation codes marked with *rxf2 cause reserved instruction exceptions in all current implementations and are reserved for future versions of the architecture. *rxf2 is separated from other reserved instructions for COPz. These are not detected as reserved instruction codes that cause an exception on the R3000. The R4000 detects them.

***rx40** An operation code marked with *rx40 causes a reserved instruction exception on R4000 and CW4010 processors (when in R4000 mode). It is used as a Restore From Exception (RFE) instruction on the R3000, LR33000, LR33300, and CW4010 in R3000 mode.

***rx64** Operation codes marked with *rx64 cause a reserved instruction exception. They are 64-bit instructions on R4000.

***nrx** Operation codes marked with *nrx are invalid but do not cause reserved instruction exceptions in CW4010 implementations.

x1 Operation codes marked with x1 are originally extended instructions in CW4010 implementations. They are reserved instructions that cause an exception on R4000.

x2 The operation code CACHE marked with x2 is valid only for CW4010 processors with CP0 enabled and causes a reserved instruction exception with CP0 disabled. Bits [20:16] are sub-opcodes. They are instructions for cache maintenance, and the functions are not compatible with R4000. Recommended mnemonics are FLUSHI, FLUSHD, FLUSHID, and WB `offset`(base). Undefined opcodes of CACHE instruction do not cause reserved instruction exception in CW4010 implementations.

x3 Operation codes marked with x3 are originally extended instructions in CW4010 implementations. They are used for 64-bit multiply and divide instructions on R4000. If the MUL bit or MAD bit in the CCC register is zero, they cause a reserved instruction exception. The CCC register is described in detail in Section 4.3.10, “Configuration and Cache Control (CCC) Register (16),” on page 4-20.

x4 Operation codes marked with x4 cause a reserved instruction exception if the MUL bit in the CCC register is zero.

x5 The operation code ERET marked with x5 is valid only on the R4000 and CW4010 in R4000 mode.

x6 Operation codes marked with x6 are coprocessor-3 instructions, which are not available on R4000. These are available on the R3000 and CW4010.

Table 3.18
CW4010 Opcode Bit Encoding

[31:29] \ [28:26]	Opcode 0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	COP1	COP2	COP3[x6]	BEQL	BNEL	BLEZL	BGTZL
3	*rx64	*rx64	*rx64	*rx64	ADDCIU[x1]	*rxf1	*rxf1	*rxf1
4	LB	LH	LWL	LW	LBU	LHU	LWR	*rx64
5	SB	SH	SWL	SW	*rx64	*rx64	SWR	CACHE[x2]
6	LL	LWC1	LWC2	LWC3[x6]	*rx64	*rx64	*rx64	*rx64
7	SC	SWC1	SWC2	SWC3[x6]	*rx64	*rx64	*rx64	*rx64

Table 3.19
SPECIAL Opcode Bit Encoding

[5:3] \ [2:0]	SPECIAL Function 0	1	2	3	4	5	6	7
0	SLL	SELSR[x1]	SRL	SRA	SLLV	SELSL[x1]	SRLV	SRAV
1	JR	JALR	FFS[x1]	FFC[x1]	SYSCALL	BREAK	*rxf1	SYNC
2	MFHI[x4]	MTHI[x4]	MFLO[x4]	MTLO[x4]	*rx64	*rxf1	*rx64	*rx64
3	MULT[x4]	MULTU[x4]	DIV[x4]	DIVU[x4]	MADD[x3]	MADDU[x3]	MSUB[x3]	MSUBU[x3]
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	MIN[x1]	MAX[x1]	SLT	SLTU	*rx64	*rx64	*rx64	*rx64
6	TGE	TGEU	TLT	TLTU	TEQ	*rxf1	TNE	*rxf1
7	*rx64	*rxf1	*rx64	*rx64	*rx64	*rxf1	*rx64	*rx64

Table 3.20
REGIMM Opcode rt Bit
Encoding

	[18:16]	REGIMM rt						
[20:19]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	BLTZ	BGEZ	BLTZL	BGEZL	*rxf1	*rxf1	*rxf1	*rxf1
1	TGEI	TGEIU	TLTI	TLTIU	TEQI	*rxf1	TNEI	*rxf1
2	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*rxf1	*rxf1	*rxf1	*rxf1
3	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1

Table 3.21
CACHE^{x2} Opcode rt Bit
Encoding

	[18:16]	CACHE^{x2} rt						
[20:19]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	*nrx	FLUSHI^{x2}	FLUSHD^{x2}	FLUSHID^{x2}	WB^{x2}	*nrx	*nrx	*nrx
1	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
2	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
3	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx

Table 3.22
COPz rs Opcode Bit
Encoding

	[23:21]	COPz rs						
[25:24]	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	MFCz	*rx64	CFCz	*rxf2	MTCz	*rx64	CTCz	*rxf2
1	BC	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2
2	COPz (Coprocessor defined instructions)							
3								

Table 3.23
COPz rt Opcode Bit
Encoding

[20:19] \ [18:16]	0	1	2	3	4	5	6	7
	COPz rt							
0	BCF	BCT	BCFL	BCTL	*rxf2	*rxf2	*rxf2	*rxf2
1	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2
2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2
3	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2

Table 3.24
CP0 Opcode Bit
Encoding

[5:3] \ [2:0]	0	1	2	3	4	5	6	7
	CP0 Function							
0	*nrx	TLBR	TLBWI	*nrx	*nrx	*nrx	TLBWR	*nrx
1	TLBP	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
2	RFE[rx40]	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
3	ERET[x5]	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
4	WAITI[x1]	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
5	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
6	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
7	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx

Chapter 4
CW4010 Exception Processing

This chapter describes the CW4010's system coprocessor, Coprocessor-0 (CP0), and explains how the CW4010 handles exception processing. The chapter contains the following sections:

4.1 Overview

When the CW4010 detects an exception, it suspends the normal sequence of instruction execution, exits from User mode, and enters Kernel mode where it can handle exceptions. The CW4010 reverts to Kernel mode, regardless of the mode at the time of the exception. The processor then disables interrupts and forces a software handler located at a fixed address in memory to be executed. The handler saves the context of the processor. The context must be restored when the exception has been handled. Section 5.2.1, "Operating Modes," beginning on page 5-3 provides more information on this subject.

When an exception occurs, the CP0 loads the Exception Program Counter (EPC) with a restart location where execution may resume after the exception has been serviced. The restart location in the EPC is the address of the instruction that caused the exception or, if the instruction was executing in a Branch Delay slot, the address of the branch instruction immediately preceding the delay slot. The instruction causing the exception and all the instructions following in the pipeline are aborted. They will be refetched after return from the exception.

This chapter describes the events that can initiate exception processing. Table 4.1 summarizes the events.

Table 4.1
CW4010 Exceptions

Exception	Cause
Cold Reset	Deassertion of the CW4010 cold reset signal, CRESETn.
Warm Reset	Deassertion of the CW4010 warm reset signal, WRESETn.
Non-Maskable Interrupt	Assertion of the non-maskable interrupt signal, NMIn.
Debug	Detection of a program counter breakpoint, data address breakpoint, or trace event. Not supported in standard R3000 and R4000 processors.
Address Error	Attempt to load, fetch, or store an unaligned word—that is a word that is at an address not evenly divisible by four, or a halfword that is at an address not evenly divisible by two. References to an address for which the most significant bit was set while in the CW4010 was in User mode may also cause an address error.
TLB Refill	There is no TLB entry to match a reference to a mapped address space.
TLB Entry Invalid	A virtual address reference matches a TLB entry that is marked invalid.
TLB Modified	A store operation's virtual address reference matches a TLB entry that is marked valid but is not dirty/writable.
Bus Error	Assertion of the CW4010 external bus error signal, SCBERRn.
Integer Overflow	Two's complement overflow during an add or subtract.
Trap	One of the Trap instructions results in a "true" condition.
System Call	An attempt to execute the SYSCALL instruction.
Breakpoint	An attempt to execute the BREAK instruction.
Reserved Instruction	Execution of an instruction with an undefined or reserved major operation code (bits [31:26]), or a SPECIAL instruction whose minor operation code (bits [5:0]) is undefined.
Coprocessor Unusable	Execution of a coprocessor instruction where the Cu (coprocessor usable) bit is not set for the target coprocessor.
Floating Point	Available for use by an external floating-point coprocessor.
Interrupt	Assertion of one of the CW4010's six hardware interrupt inputs, or the setting of one of the two software interrupt bits in the Cause register. Interrupts must be enabled.
External Vectored Interrupt	Assertion of the CW4010 EXViNTn input. Not supported in R3000 and R4000 processors.

4.2 R3000 Exception Compatibility Mode

Although the CW4010 processor is based on the MIPS R4000 architecture, an R3000 style exception processing capability has been added. This facility allows you to configure CP0 exception processing in such a way that existing R3000 exception handling code can be run on the CW4010 processor with little or no modification to the code.

R3000 compatibility mode is under the control of the compatibility bit (bit 24) of the Configuration and Cache Control (CCC) Register, discussed in Section 4.3.10, “Configuration and Cache Control (CCC) Register (16),” on page 4-20. The compatibility bit is reset to 0 (R4000 mode) when a cold reset exception occurs. If R3000 mode operation is desired, bit 24 should be set to 1 as part of the cold reset handler. Once it has been placed in R3000 mode, the processor should only be switched back to R4000 mode by another cold reset. When R3000 mode is enabled, the behavior of the following areas is affected:

- Status Register

  The lower six bits of the Status register are redefined to implement the Kernel/User mode and interrupt enable stack as defined by the R3000 architecture. The Status Register is discussed in detail in Section 4.3.6, “Status Register (12),” on page 4-9.

- Exception Handling Vectors

  The exception handling vectors (base and offset) are remapped to those specified by the R3000 architecture. The Exception Vectors are discussed in detail in Section 4.4.3, “Exception Vector Locations,” on page 4-31.

- Exception Return (RFE vs. ERET)

  When operating in R3000 compatibility mode, exception return is accomplished using the RFE instruction. If an attempt is made to use the ERET instruction, a Reserved Instruction exception will be recognized.

The following sections provide more detail on CW4010 exception handling. Where appropriate, the differences between standard operation R4000 and R3000 compatibility mode are noted. In all other cases, operation is identical.

4.3 Exception Handling Registers

This section describes the CP0 registers used in exception processing. Software examines these registers during exception processing to determine the cause of an exception and the state of the CPU at the time of the exception. Each of the registers is listed and described in detail in the sections that follow.

Table 4.2 CP0 Exception Processing Registers

Register Name	CP0 Register Number
Context	4
DCS (Debug Control and Status)	7
BadVAddr (Bad Virtual Address)	8
Count	9
Compare	11
Status	12
Cause	13
EPC (Exception Program Counter)	14
PRId (Processor Revision Identifier)	15
CCC (Configuration and Cache Control)	16
LLAdr (Load Linked Address)	17
BPC (Breakpoint Program Counter)	18
BDA (Breakpoint Data Address)	19
BPCM (Breakpoint PC Mask)	20
BDAM (Breakpoint Data Addr Mask)	21
ErrorPC	30

Two other CP0 registers that are part of the virtual memory management system and contain important information about exception handling are the Index register (CP0 register 0), described in Section 5.3.2.4, “Index Register (0),” on page 5-12, and the Random register (CP0 register 1), described in Section 5.3.2.5, “Random Register (1),” on page 5-12.

You can use the MTC0 (Move to Coprocessor-0) instruction to set the bits in the registers, and MTF0 (Move from Coprocessor-0) to read the contents of the registers.

4.3.1 Context Register (4)

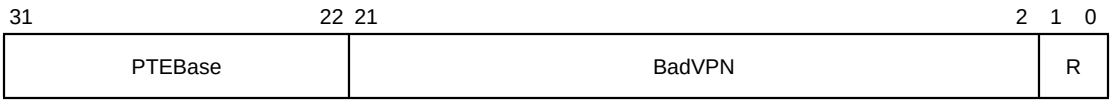
The Context register is a read/write register containing a pointer to an entry in the Page Table Entry (PTE) array. This array is an operating system data structure that stores virtual to physical address translations. When there is a TLB miss, operating system software handles the miss by loading the TLB with the missing translation from the PTE array.

The BadVPN field is not writable. It contains the VPN of the most recently translated virtual address that did not have a valid translation (TLBL or TLBS). The PTEBase field is both writable and readable, and indicates the base address of the PTE table of the current user address space.

The Context register duplicates some of the information provided in the BadVAddr register, but the information is in a form that is more useful for a software TLB exception handler.

The Context register can be used by the operating system to hold a pointer into the PTE array. The operating system sets the PTE base field register, as needed. Normally, the operating system uses the Context register to address the current page map, which resides in the kernel-mapped segment kseg2. The register is included solely for the use of the operating system. Figure 4.1 shows the format of the Context register.

Figure 4.1
Context Register



PTEBase **Page Table Entry Base** **[31:22]**
This field is the Operating System Pointer. It points to the Page Table Entry in memory.

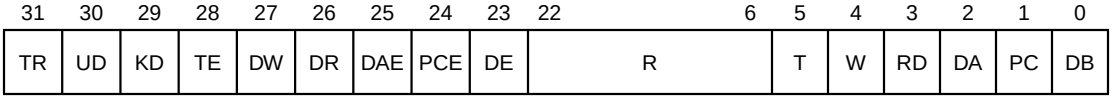
BadVPN **Bad Virtual Page Number** **[21:2]**
This field contains the most recently translated virtual address that did not have a valid translation. Bits [31:12] of this field contain the virtual address that caused the TLB miss. This format provides a table of four-byte PTEs for a page size of 4 Kbytes. For other PTE and page sizes, shifting and masking bits [31:12] produces an appropriate address.

R **Reserved** **[1:0]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

4.3.2 Debug Control and Status (DCS) Register (7)

The Debug Control and Status (DCS) register contains the enable and status bits for the CW4010 debug facility. All bits have read/write access. Figure 4.2 shows the format of the DCS register.

Figure 4.2
DCS Register



TR **Trap** **31**
This is the trap enable bit. Setting it to 1 traps debug events to the debug exception vector. Clearing it to 0 disables the trap. However, the status bits are updated with status debug event information when the bit is cleared.

UD **User Mode Debug Event** **30**
This bit is set to 1 to detect a debug event when the CW4010 is operating in User mode.

KD **Kernel Mode Debug Event** **29**
This bit is set to 1 to detect a debug event when the CW4010 is operating in Kernel mode.

TE **Trace Event** **28**
This bit is set to 1 to detect a trace event (non-sequential fetch operation).

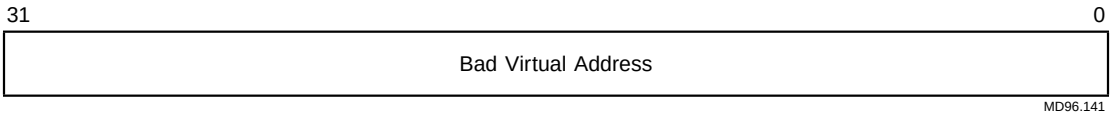
DW **Data Write** **27**
This bit is set to 1 to detect a Data Write at BDA (Breakpoint Data Address) event. The bit is used in conjunction with DAE.

DR	**Data Read** This bit is set to 1 to detect a Data Read at BDA event. The bit is used in conjunction with DAE.	**26**
DAE	**Detect BDA Event** This bit is set to 1 to detect BDA debug events.	**25**
PCE	**Program Counter Breakpoint Event** This bit is set to 1 to detect Program Counter Breakpoint events.	**24**
DE	**Debug Enable** This bit is set to 1 to enable the debug facility. Clearing the bit disables the debug facility.	**23**
R	**Reserved** These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.	**[22:6]**
T	**Trace** The setting of this bit indicates the trace event status. It is set to 1 to indicate that a trace event has been detected.	**5**
W	**Write** This bit is the write reference bit and its setting matches the DW bit setting.	**4**
RD	**Read** This bit is the read reference bit and its setting matches the DR bit setting.	**3**
DA	**DAE Debug Condition** This bit indicates the status of the DAE debug condition.	**2**
PC	**PCE Debug Condition** This bit indicates the status of the PCE debug condition.	**1**
DB	**Debug Detected** This bit is set whenever any debug condition is detected.	**0**

4.3.3 Bad Virtual Address (BadVAddr) Register (8)

The Bad Virtual Address (BadVAddr) register is a read-only register that holds the 32-bit failing virtual address for address error (AdEL, AdES) and TLB translation (TLBL, TLBS, Mod) exceptions. Figure 4.3 shows the format of the BadVAddr register.

Figure 4.3
BadVAddr Register



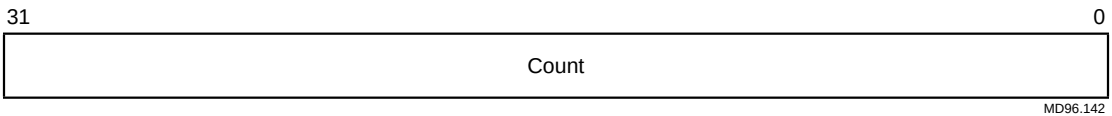
4.3.4 Count Register (9)

The Count register acts as a timer. It increments at a constant rate regardless of whether an instruction is executed, retried, or any forward progress is made. The Count register increments at half the maximum instruction issue rate.

The Count register is a read/write register—it can be written for diagnostic purposes or for system initialization to synchronize two processors operating in lock step.

Figure 4.4 shows the format of the Count register.

Figure 4.4
Count Register

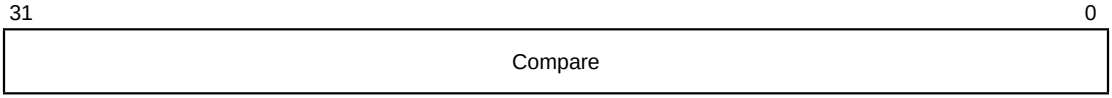


4.3.5 Compare Register (11)

The Compare register implements a timer service (see also the Count register) that maintains a stable value and does not change on its own. When the timer facility is enabled and the value of the Count register equals the value of the Compare register, interrupt bit IP7 in the Cause register is set. This causes an interrupt on the next execution cycle when the interrupt is enabled. Writing a value to the Compare register clears the timer interrupt.

For diagnostic purposes, the Compare register is a read/write register. In normal operation, the Compare register is only written. Figure 4.5 shows the format of the Compare register.

Figure 4.5
Compare Register



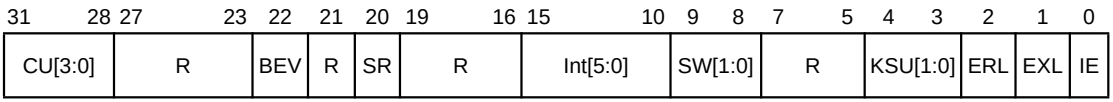
4.3.6 Status Register (12)

The Status register is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. The format of the Status register is slightly different when the CW4010 is operating in R4000 mode and R3000 mode. "R4000 Mode Operation" below, describes the format for R4000 mode operation. Section 4.3.6.2, "R3000 Mode Operation" on page 4-29 describes the format for R3000 mode operation.

4.3.6.1 R4000 Mode Operation

The format of the R4000 version of the Status register (CCC24 = 0) is shown in Figure 4.6.

Figure 4.6
Status Register
(R4000 Mode)



CU[3:0] **Coprocessor Usability Bits** **[31:28]**
The software uses this field to control accesses to the coprocessors. When the bit is set to 1 the corresponding coprocessor is usable, as shown below:

CU3 = 1 enables coprocessor 3
CU2 = 1 enables coprocessor 2
CU1 = 1 enables coprocessor 1
CU0 = 1 enables coprocessor 0

R — **Reserved** **[27:23, 21, 19:16, 7:5]**

These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

BEV — **Bootstrap Exception Vector** **22**

This bit controls the location of the TLB refill and the general exception vectors. Setting the bit to 1 implements a bootstrap operation and bootstrap vector locations are used. When the bit is cleared to 0, normal exception vectors are used.

Refer to the following subsections for further information: “Cold Reset” on page 4-12; “Warm Reset” on page 4-12.

SR — **Soft Reset** **20**

This bit indicates whether a warm reset or a non-maskable interrupt has occurred. When the bit is set to 1, it indicates a warm reset. When it is cleared to 0, it indicates a non-maskable interrupt.

Refer to the subsection “Warm Reset” on page 4-12, for further information.

INT[5:0] — **Interrupt Mask** **[15:10]**

This field is a six-bit [5:0] hardware interrupt mask. Setting a bit to 1 enables the corresponding hardware interrupt. For example, setting bit 5 enables hardware interrupt 5.

SW[1:0] — **Software Interrupt Mask** **[9:8]**

This field is a two-bit [1:0] software interrupt mask. Setting a bit to 1 enables the corresponding software interrupt.

KSU — **Kernel/User Mode** **[4:3]**

This field determines the base operating mode of the CW4010 core as follows:

[1:0] = 00, base mode is Kernel
[1:0] = 10, base mode is User
All other settings are reserved.

Refer to the following subsections for further information: “Processor Modes” on page 4-11; “Kernel Address Space Accesses” on page 4-12; “Warm Reset” on page 4-12.

ERL **Error Level** **2**

This bit determines the error level of the CW4010. When it is set to 1, the level is Error. When it is cleared to 0, the level is Normal.

Refer to the following subsections for further information: “Interrupt Enable” on page 4-11; “Processor Modes” on page 4-11; “Kernel Address Space Accesses” on page 4-12; “Cold Reset” on page 4-12.

EXL **Exception Level** **1**

This bit determines the exception level of the CW4010. When it is set to 1, the level is Exception. When it is cleared to 0, the level is Normal.

Refer to the following subsections for further information: “Interrupt Enable” on page 4-11; “Processor Modes” on page 4-11; “Kernel Address Space Accesses” on page 4-12.

IE **Interrupt Enable** **0**

Setting this bit to 1 enables interrupts. Clearing it to 0 disables interrupts.

Refer to subsection “Interrupt Enable” below for further information.

Interrupt Enable – Interrupts are enabled when the following field conditions are true:

- IE is set to 1.
- EXL is cleared to 0.
- ERL is cleared to 0.

If these conditions are met, interrupts are recognized according to the setting of the INT and SW mask bits.

Processor Modes – The setting of the KSU bit, in conjunction with the settings of the EXL and ERL bits, defines the CW4010 processor modes as follows:

- The processor is in User mode when KSU is equal to 10_b, and EXL and ERL are cleared to 0.

- The processor is in Kernel mode under any one of the following conditions:
  - KSU is equal to 00_b.
  - EXL is set to 1.
  - ERL is set to 1.

Kernel Address Space Accesses – Access to the Kernel address space is allowed only when the processor is in Kernel mode, that is under any one of the following conditions:

- KSU is equal to 00_b.
- EXL is set to 1.
- ERL is set to 1.

User Address Space Accesses – Access to the User address space is always allowed.

Cold Reset – The contents of the Status register are undefined after a cold reset, except for the following bits:

- ERL and BEV are set to 1.

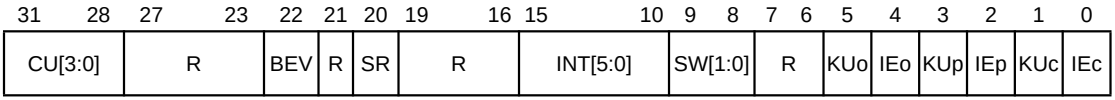
Warm Reset – The contents of the Status register are unchanged by warm reset, except for the following bits:

- ERL, BEV, and SR bits are set to 1.

4.3.6.2 R3000 Mode Operation

The format of the R3000 version of the Status register (CCC24 = 1) is shown in Figure 4.7.

Figure 4.7
Status Register
(R3000 Mode)



CU[3:0] **Coprocessor Usability Bits** **[31:28]**

The software uses this field to control accesses to the coprocessors. When the bit is set to 1 the corresponding coprocessor is usable, as shown below:

CU3 = 1 enables coprocessor 3
CU2 = 1 enables coprocessor 2
CU1 = 1 enables coprocessor 1
CU0 = 1 enables coprocessor 0

R **Reserved** **[27:23, 21, 19:16, 7:6]**

These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

BEV **Bootstrap Exception Vector** **22**

This bit controls the location of the TLB refill and the general exception vectors. Setting the bit to 1 implements a bootstrap operation and bootstrap vector locations are used. When the bit is cleared to 0, normal exception vectors are used.

Refer to the subsection “Warm Reset” on page 4-15 for further information.

SR **Soft Reset** **20**

This bit indicates whether a warm reset or a non-maskable interrupt has occurred. When the bit is set to 1, it indicates a warm reset. When it is cleared to 0, it indicates a non-maskable interrupt.

Refer to the subsection “Warm Reset” on page 4-12 for further information.

INT[5:0] **Interrupt Mask** **[15:10]**

This field is a six-bit [5:0] hardware interrupt mask. Setting a bit to 1 enables the corresponding hardware interrupt. For example, setting bit 5 to 1 enables hardware interrupt 5.

SW[1:0] **Software Interrupt Mask** **[9:8]**

This field is a two-bit [1:0] software interrupt mask. Setting a bit to 1 enables the corresponding software interrupt.

KUo **Kernel/User Mode, Old** **5**

This bit shows the old base operating mode of the CW4010 core. Setting it to 1 indicates User mode. Clearing the bit to 0 indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.

Refer to the subsection “Warm Reset” on page 4-15 for further information.

IEo **Interrupt Enable, Old** **4**

This bit shows the old interrupt enable setting. Setting it to 1 indicates that interrupts are enabled. Clearing the bit to 0 indicates that interrupts are disabled. The bit is part of a three-bit stack that indicates old, previous, and current interrupt enable settings.

Refer to the subsection “Interrupt Enable” on page 4-11 for further information.

KUp **Kernel/User Mode, Previous** **3**

This bit shows the previous base operating mode of the CW4010 core. Setting it to 1 indicates User mode. Clearing the bit to 0 indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.

Refer to the following subsections for further information: “Warm Reset” on page 4-15; “Processor Modes” on page 4-11.

IEp **Interrupt Enable, Previous** **2**

This bit shows the previous interrupt enable setting. Setting it to 1 indicates that interrupts are enabled. Clearing the bit to 0 indicates that interrupts are disabled. The bit is part of a three-bit stack that indicates old, previous, and current interrupt enable settings.

Refer to the subsection “Interrupt Enable” on page 4-11 for further information.

KUc **Kernel/User Mode, Current** **1**

This bit shows the current base operating mode of the CW4010 core. Setting it to 1 indicates User mode. Clearing the bit to 0 indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.

Refer to the following subsections for further information: “Warm Reset” on page 4-15; “Processor Modes” on page 4-11.

IEc **Interrupt Enable, Current** **0**

This bit shows the old interrupt enable setting. Setting it to 1 indicates that interrupts are enabled. Clearing the bit to 0 indicates that interrupts are disabled. The bit is part of a three-bit stack that indicates old, previous, and current interrupt enable settings.

Refer to subsection “Interrupt Enable” on page 4-11 for further information.

Interrupt Enable – Interrupts are enabled when IEc is set to 1. In this case, interrupts are recognized according to the setting of the INT and SW masks.

Processor Modes – CW4010 processor modes are defined by the setting of the KUc bit:

- The processor is in User mode when KUc is set to 1.
- The processor is in Kernel mode when KUc is cleared to 0.

Kernel Address Space Accesses – Access to the Kernel address space is allowed only when the processor is in Kernel mode.

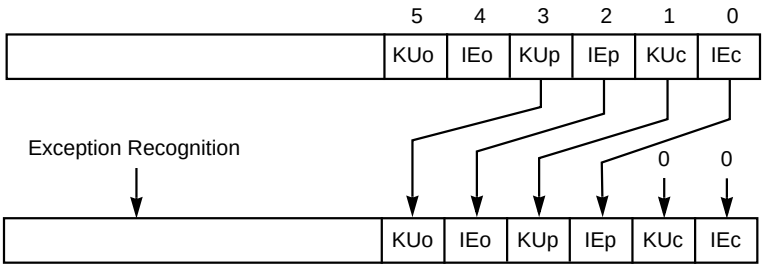
User Address Space Accesses – Access to the User address space is always allowed.

Warm Reset – The contents of the Status register are unchanged by warm reset, except for the following bits:

- The BEV and SR bits are set to 1.
- The KU and IE bits are pushed deeper into the stack and KUc and IEc are cleared to 0, for example: KUo/IEo ← KUp/IEp ←KUc/IEc ← 0/0.

Figure 4.8 shows how the CW4010 core manipulates the Status register during exception recognition.

Figure 4.8
Status Register and Exception Recognition



KUc	Current Kernel/User mode bit
IEc	Current Interrupt Enable mode bit
KUp	Previous Kernel/User mode bit
IEp	Previous Interrupt Enable mode bit
KUo	Old Kernel mode bit
IEo	Old Interrupt Enable mode bit

MD96.146

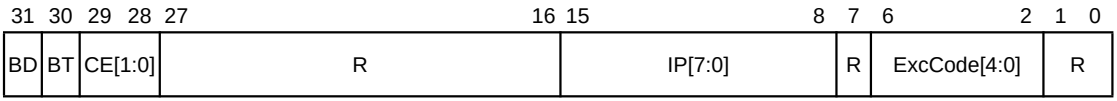
When the CW4010 recognizes an exception, it saves the current Kernel/User mode bit (KUc) and the current interrupt enable bit (IEc) in the previous Kernel/User mode bit (KUp) and previous interrupt enable bit (IEp), respectively. The previous bits are saved in the old bits, and the current bits are cleared to 0. The process is shown in the following example: KUo/IEo ← KUp/IEp ← KUc/IEc ← 0.

When the CW4010 executes a Return From Exception (RFE) instruction, the values are popped off the stack, KUc and IEc are reset to their previous values, for example: KUc/IEc ← KUp/IEp ← KUo/IEo.

4.3.7 Cause Register (13)

The Cause register is a read/write register. The contents of this register provide information about the most recent exception. The format of the register is shown in Figure 4.9. All bits in the register, with the exception of IP[1:0], are read-only bits.

Figure 4.9
Cause Register



BD **Branch Delay** **31**

This bit indicates whether or not the last exception was taken while the CW4010 was executing an instruction in the Branch Delay slot. Setting the bit to 1 indicates that the exception was taken. Clearing the bit to 0 indicates that the exception was not taken.

BT **BD Set** **30**

If the BD bit is set, setting this bit to 1 indicates that the branch was taken.

CE[1:0] **Coprocessor Error** **[29:28]**

The value in the coprocessor error field indicates the coprocessor unit referenced when a Coprocessor Unusable exception is taken:

CE1	CE0	Coprocessor Referenced
1	1	Coprocessor 3
1	0	Coprocessor 2
0	1	Coprocessor 1
0	0	Coprocessor 0

R **Reserved** **[27:16, 7, 1:0]**

These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

IP[7:0] **Interrupt Pending** **[15:8]**

This field indicates if an interrupt is pending. There is a direct correlation between the bit set and the interrupt pending. So, if IP7 (bit 15) is set, interrupt 7 is pending.

ExcCode[4:0] **Exception Code** **[6:2]**

This field defines the exception code. Table 4.3 lists the valid exception code values.

Table 4.3 Cause Register ExcCode Field

Exception Code Value	Mnemonic	Description
0	Int	Interrupt
1	Mod	TLB modification exception
2	TLBL	TLB exception (load or instruction fetch)
3	TLBS	TLB exception (store)
4	AdEL	Address error exception (load or instruction fetch)
5	AdES	Address error exception (store)
6	Bus	Bus error exception

(Sheet 1 of 2)

Table 4.3 (Cont.)
Cause Register
ExcCode Field

Exception Code Value	Mnemonic	Description
7	—	Reserved
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Arithmetic overflow exception
13	Tr	Trap exception
14	—	Reserved
15	FPE	Floating-point exception
16-31	—	Reserved

(Sheet 2 of 2)

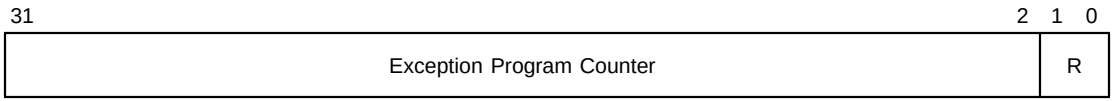
4.3.8 Exception Program Counter Register (14)

The Exception Program Counter (EPC) is a read-write register that contains the address where processing resumes after an exception has been serviced. For synchronous exceptions, the EPC register contains either:

- The virtual address of the instruction that was the direct cause of the exception
- The virtual address of the immediately preceding branch or jump instruction (when the instruction is in a Branch Delay slot, and the Branch Delay bit in the Cause register is set).

Figure 4.10 shows the format of the EPC register. Bits [31:2] make up the program counter. Bits [1:0] are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

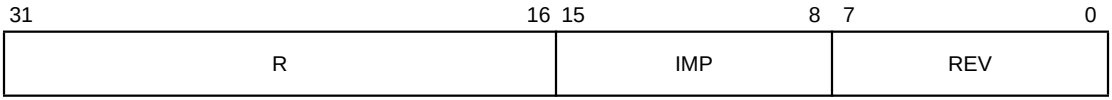
Figure 4.10
EPC Register



4.3.9 Processor Revision Identifier Register (15)

The Processor Revision Identifier (PRId) is a 32-bit, read-only register that contains information identifying the implementation and revision level of the CW4010 core, as shown in Figure 4.11.

Figure 4.11
PRId Register



R **Reserved** **[31:16]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

IMP **Implementation Number** **[15:8]**
The value in this field represents the core's implementation number. This field can be programmed at the core interface using the iMPLop[3:0] lines.

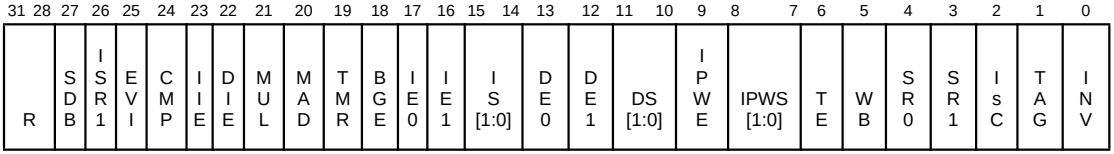
REV **Revision Number** **[7:0]**
The value of this field is interpreted as a processor unit revision number. The revision number is a value of the form y.x, where y is a major revision number in bits [7:4] and x is a minor revision number in bits [3:0]. This field can be programmed at the core interface using the REVLop[3:0] lines.

The revision number can distinguish between some chip revisions. However, LSI Logic does not guarantee that changes to this core will necessarily be reflected in the PRId register, or that changes to the revision number necessarily reflect real core changes. For this reason, these values are not listed and software should not rely on the revision number in the PRId register to characterize the core.

4.3.10 Configuration and Cache Control (CCC) Register (16)

The Configuration and Cache Control (CCC) register allows software to configure various pieces of the CW4010 design (for example, BIU, TLB, and Cache Controllers). Figure 4.12 shows the format of the CCC register.

Figure 4.12
CCC Register



MD96.150

R **Reserved** **[31:28]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

SDB **Scan Debug Mode** **27**
This bit enables the Scan Debug mode. The bit is set to 1 to enable the mode and cleared to 0 to disable the mode.

ISR1 **Icache Scratchpad RAM** **26**
This bit enables the Icache to be used as a scratchpad RAM. Setting the bit to 1 enables scratchpad RAM mode. Clearing it to 0 disables scratchpad RAM mode.

EVI **External Vectored Interrupt** **25**
This bit enables and disables external vectored interrupt. Setting the bit to 1 enables the interrupt and clearing it to 0 disables the interrupt.

CMP **R3000 Compatibility** **24**
This bit enables and disables R3000 Compatibility mode. Setting the bit to 1 enable the mode and clearing it disables the mode.

IIE **Icache Invalidate Enable** **23**
This bit enables and disables the Icache invalidate request. Setting the bit to 1 enables the request and clearing it to 0 disables the request.

DIE **Dcache Invalidate Enable** **22**
This bit enables and disables the Dcache invalidate request. Setting the bit to 1 enables the request and clearing it to 0 disables the request.

MUL **Multiplier Enable** **21**
This bit enables and disables the hardware multiplier. Setting the bit to 1 enables the multiplier and clearing it disables the multiplier.

MAD **Multiplier Accumulate Extensions** **20**
This bit allows the multiplier to support accumulate extensions. Setting the bit to 1 enables the feature and clearing the bit disables the feature. When this bit is set, MUL must also be set.

TMR **Timer** **19**
Setting this bit to 1 enables the timer facility, Count = Compare $\rightarrow$ P7.

BEG **BIU Bus Enable Grant** **18**
This bit enables and disables the BIU bus grant. Setting this bit to 1 enables the external bus master. Clearing it to 0 allows the CW4010 core to ignore the external bus master.

IE0 **Icache Set-0 Enable** **17**
This bit enables and disables Set-0 of the Icache. Setting the bit to 1 enables Set-0 and clearing it to 0 disables Set-0.

IE1 **Icache Set-1 Enable** **16**
This bit enables and disables Set-1 of the Icache. Setting the bit to 1 enables Set-1 and clearing it to 0 disables Set-1.

IS[1:0] **Icache Size** **[15:14]**
The IS[1:0] field determines the size of the Icache set. The field is set as follows:

IS1	IS0	Cache Size
0	0	1K
0	1	2K
1	0	4K
1	1	8K

DE0 **Dcache Set-0 Enable** **13**

This bit enables and disables Set-0 of the Dcache. Setting the bit to 1 enables Set-0 and clearing it to 0 disables Set-0.

DE1 **Dcache Set-1 Enable** **12**

This bit enables and disables Set-1 of the Dcache. Setting the bit to 1 enables Set-1 and clearing it to 0 disables Set-1.

DS[1:0] **Dcache Size** **[11:10]**

The DS[1:0] field determines the size of the Dcache set. The field is set as follows:

IDS1	DS0	Cache Size
0	0	1K
0	1	2K
1	0	4K
1	1	8K

IPWE **In-Page Write Enable** **9**

This bit enables and disables in-page write operations. Setting the bit to 1 enables in-page write and clearing it to 0 disables in-page write.

IPWS[1:0] **In-Page Write Size** **[8:7]**

The IPWS[1:0] field determines the size of the Icache for in-page write operations. The field is set as follows:

IPWS1	IPWS0	In-Page Write Size
0	0	1K
0	1	2K
1	0	4K
1	1	8K

TE **TLB Enable** **6**

This bit enables and disables the TLB. Setting the bit to 1 enables the TLB and clearing the bit to 0 disables the TLB.

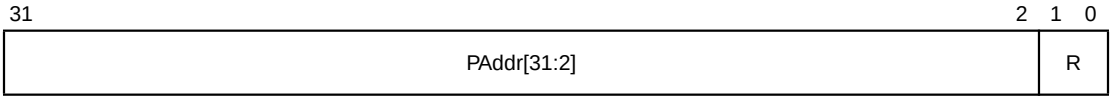
WB	**WriteBack** This bit defines operation for addresses not mapped by the TLB. Setting the bit to 1 enables a WriteBack operation and clearing it to 0 enables a WriteThrough operation.	**5**
SR0	**Scratchpad RAM Mode Set-0** This bit enables and disables scratchpad RAM mode for Set-0 of the Dcache. Setting the bit to 1 enables scratchpad mode and clearing it to 0 disables scratchpad mode.	**4**
SR1	**Scratchpad RAM Mode Set-1** This bit enables and disables scratchpad RAM mode for Set-1 of the Dcache. Setting the bit to 1 enable scratchpad mode and clearing it to 0 disables scratchpad mode.	**3**
IsC	**Isolate Cache** This bit enables isolate cache mode. This means that stores to the cache are not propagated to external memory. Setting the bit to 1 enables the mode and clearing it to 0 disables the mode.	**2**
TAG	**Tag Test Mode** This bit enables and disables tag test mode, which is used for cache maintenance. Setting the bit to 1 enables the mode and clearing it to 0 disables the mode.	**1**
INV	**Invalidate Cache Mode** This bit enables and disables cache invalidate mode, which is used for cache maintenance. Setting the bit to 1 enables the mode and clearing it to 0 disables the mode.	**0**

4.3.11 Load Linked Address (LLAddr) Register (17)

The Load Linked Address (LLAddr) register is a read/write register that contains the physical address (PAddr[31:2]) read by the most recent Load Linked instruction. This register is used for diagnostic purposes only, and serves no function during normal operation. The LLAddr register is physically located in the LSU. The CP0 must send read/write signals to the LSU when the value of the register is to be read or written.

Figure 4.13 shows the format of the LLAddr register. Bits [31:2] contain the PAddr[31:2]. Bits [1:0] are reserved and cleared to 0.

Figure 4.13
LLAddr Register

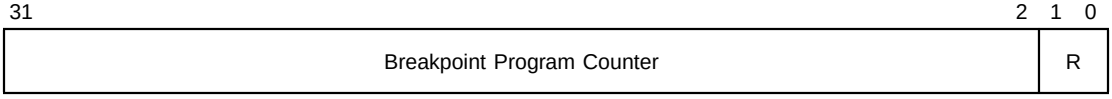


4.3.12 Breakpoint Program Counter (BPC) Register (18)

The Breakpoint Program Counter (BPC) register is a read/write register that software uses to specify a program counter breakpoint. The BPC register is used in conjunction with the Breakpoint PC Mask register, described in Section 4.3.14, “Breakpoint PC Mask (BPCM) Register (20)” on page 4-25.

Figure 4.14 shows the format of the 32-bit BPC register. Bits [31:2] make up the Breakpoint Program Counter. Bits [1:0] are reserved and cleared to 0.

Figure 4.14
BPC Register

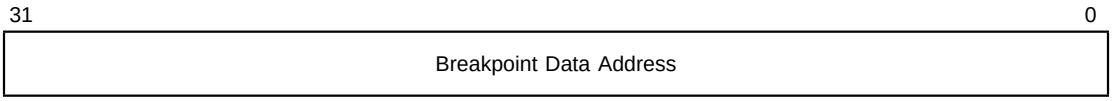


4.3.13 Breakpoint Data Address (BDA) Register (19)

The Breakpoint Data Address (BDA) register is a read/write register that software uses to specify a virtual data address breakpoint. The BDA register is used in conjunction with the Breakpoint Data Address Mask register described in Section 4.3.15, “Breakpoint Data Address Mask (BDAM) Register (21)” on page 4-25.

Figure 4.15 shows the format of the 32-bit BDA register. Bits [31:0] make up the BDA.

Figure 4.15
BDA Register

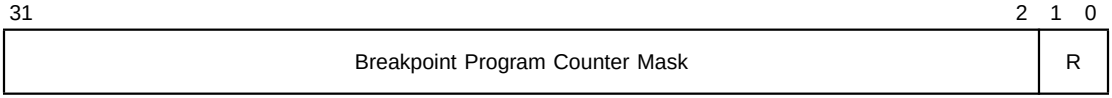


4.3.14 Breakpoint PC Mask (BPCM) Register (20)

The Breakpoint Program Counter Mask (BPCM) register is a read/write register that masks bits in the BPC register. A 1 in any bit in the BPCM register indicates that the CW4010 compares the value of the bit with the corresponding bit in the BPC register for program counter (debug) exceptions. Values of 0 in the mask indicate that the CW4010 does not check the corresponding bits in the BPC register.

Figure 4.16 shows the format of the 32-bit BPCM register. Bits [31:2] make up the mask. Bits [1:0] are reserved and cleared to 0.

Figure 4.16
BPCM Register

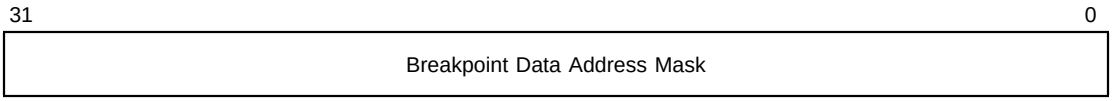


4.3.15 Breakpoint Data Address Mask (BDAM) Register (21)

The Breakpoint Data Address Mask (BDAM) register is a read/write register that masks bits in the BDA register. A 1 in any bit in the BDAM register indicates that the CW4010 compares the value of the bit with the corresponding bit in the BDA register for data address (debug) exceptions. Values of 0 in the mask indicate that the CW4010 does not check the corresponding bits in the BDA register.

Figure 4.17 shows the format of the 32-bit BDAM register. Bits [31:0] make up the BDAM.

Figure 4.17
BDAM Register



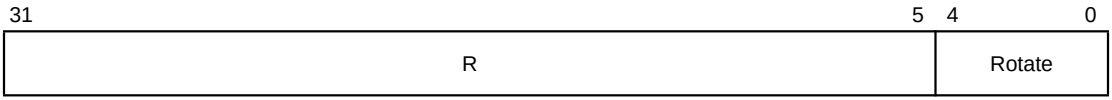
4.3.16 Rotate Register (23)

The Rotate register is used by the CW4010 instruction set extensions. Select and rotate left (SELSL), and select and rotate right (SELSR) use the lower five bits of the register [4:0] as the shift count. This is useful for data alignment operations in graphics and in bit-field selection routines for data transmission and compression applications.

Even though the Rotate register resides in the CP0, User-mode access to the register is always granted, regardless of the value contained in the Cu0 bit of the Status register.

Figure 4.18 shows the format of the Rotate register.

Figure 4.18
Rotate Register



R **Reserved** **[31:5]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

Rotate **Rotate** **[4:0]**
This field determines the shift count.

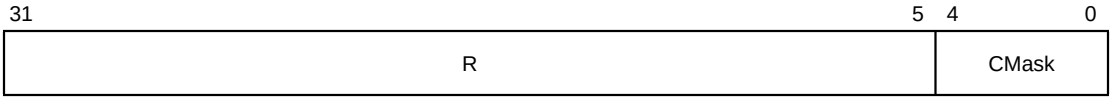
4.3.17 Circular Mask (CMask) Register (24)

The Circular Mask (CMask) register is used by the CW4010 instruction set extensions. The Load/Store word/halfword/byte with update circular instructions store a value in the destination register and update the base address register with the addition of base + offset, which is modified according to the value of bits [4:0]. This feature is important in DSP (Digital Signal Processing) and other applications that use circular buffers.

Even though the Circular Mask register resides within the CP0, User-mode access is always granted to the register, regardless of the value contained in Status[Cu0].

Figure 4.19 shows the format of the CMask register.

Figure 4.19
CMask Register



R **Reserved** **[31:5]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

CMask **Circular Mask** **[4:0]**
This field contains the circular mask.

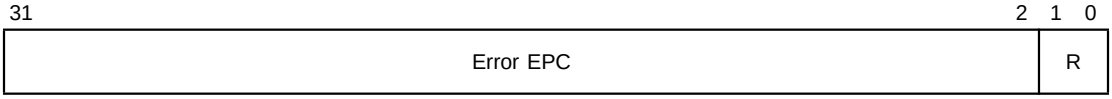
4.3.18 Error Exception Program Counter (Error EPC) Register (30)

The Error Exception Program Counter (Error EPC) register is similar to the EPC. It stores the PC (Program Counter) on cold reset, warm reset, and NMI exceptions. The read/write Error EPC register contains the virtual address at which instruction processing can resume after the interrupt has been serviced. The address may be either:

- The virtual address of the first instruction terminated by the exception
- The virtual address of the immediately preceding branch or jump instruction when the terminated instruction is in a Branch Delay slot.

Figure 4.20 shows the format of the Error EPC register. Bits [31:2] make up the Error EPC. Bits [1:0] are reserved and cleared to 0. There is no Branch Delay slot indication for the Error EPC register.

Figure 4.20
Error EPC Register



4.4 Exception Description Details

This section describes each of the CW4010 exceptions, what causes them, and how they are handled and serviced.

4.4.1 Exception Operation

To handle an exception, the processor saves the current operating state, enters Kernel mode, disables interrupts, and forces execution of a handler at a fixed address. To resume normal operation, the operating state must be restored and interrupts enabled.

When an exception occurs, the EPC register is loaded with the restart location at which execution can resume after the exception has been serviced. The EPC register contains the address of the instruction associated with the exception, or, if the instruction was executing in a Branch Delay slot, the EPC register contains the address of the branch instruction immediately preceding.

4.4.1.1 R4000 Mode Operation (Default After Cold Reset)

The CW4010 processor uses the following mechanisms for saving and restoring the operating mode and interrupt status:

- A single interrupt enable bit (IE) located in the Status register.
- A base operating mode (User, Kernel) located in the KSU field of the Status register.
- An exception level (normal, exception) located in the EXL field of the Status register.
- An error level (normal, error) located in the ERL field of the Status register.

Interrupts are enabled by setting the IE bit to 1 and both levels (EXL, ERL) to normal.

Table 4.4 shows how the current processor operating mode is defined.

Table 4.4
Current Processor Mode

Current Mode	Status KSU[1:0]	Status EXL	Status ERL
User	10	0	0
Kernel	00	0	0
Kernel	xx	1	0
Kernel	xx	0	1

Exceptions set the exception level to exception (EXL = 1). The exception handler typically resets the exception level to normal (EXL = 0) after saving the appropriate state. It sets it back to exception while restoring that state. Returning from an exception (ERET instruction) resets the exception level to normal.

4.4.1.2 R3000 Mode Operation

The R3000 mode of operation is much simpler than the R4000 mode. The current processor operating state is always defined by the KUc bit (0 –> Kernel, 1 –> User). The basic mechanism for saving and restoring the operating state of the processor is the Kernel/User (KU) and Interrupt Enable (IE) stack located in the bottom six bits of the Status register.

When responding to an exception, the current mode bits (KUc/IEc) are saved into the previous mode bits (KUp/IEp); the previous mode bits are saved into the old mode bits (KUo/IEo); and the current mode bits (KUc/IEc) are both cleared to 0.

After exception processing has been completed, the saved state is restored using the RFE instruction, which causes the previous mode bits to be copied back into the current mode bits and the old mode bits to be copied back into the previous mode bits. The old mode bits are left unchanged.

4.4.1.3 Exception Processing Diagrams

Figures 4.21 through 4.25 show the basic set of actions taken for each of the major CW4010 exception classes: Cold Reset, Warm Reset, Non-Maskable Interrupt, Common, Debug, and External Vectored Interrupt.

Figure 4.21
Cold Reset
Exception

```
Random ← TLBENTRIES - 1
Wired ← 0
CCC ← 032
DCS ← 032
ErrorPC ← PC
SR ← 04 || SR[27:23] || 1 || 0 || 0 || SR[19:3] || 1 || SR[1:0]
PC ← 0xBFC0 0000
```

Figure 4.22
Warm Reset, NMI
Exceptions

```
ErrorPC ← PC
if (CCC24 = 0) then
    SR ← SR[31:23] || 1 || 0 || 1 || SR[19:3] || 1 || SR[1:0]
else
    SR ← SR[31:23] || 1 || 0 || 1 || SR[19:6] || SR[3:0] || 02
endif
PC ← 0xBFC0 0000
```

Figure 4.23
Common
Exceptions

```
Cause ←BD || BT || CE || 012 || Cause[15:8] || 0 || ExcCode || 02
if ((CCC24 = 1) | (SR1 = 0)) then
    EPC ←PC
endif
if (CCC24 = 0) then
    SR ←SR{31:2] || 1 || SR0
else
    SR ←SR[31:6] || SR[3:0] || 02
endif
if (SR22 = 1) then
    if (CCC24 = 0) then
         PC ←0xBFC0 0200 + vector offset
    else
         PC ←0xBFC0 0100 + vector offset
    endif
else
    PC ←0x8000 0000 + vector offset
endif
```

Figure 4.24
Debug Exception

```
DCS ←DCS[31:6] || T || W || R || DA || PC || DB
Cause ←BD || BT || Cause[29:0]
if ((CCC24 = 1) | (SR1 = 0)) then
    EPC ←PC
endif
if (CCC24 = 0) then
    SR ←SR[31:2] || 1 || SR0
else
    SR ←SR[31:6] || SR[3:0] || 02
endif
if (SR22 = 1) then
    if (CCC24 = 0) then
         PC ←0xBFC0 0200 + vector offset
    else
         PC ←0xBFC0 0100 + vector offset
    endif
else
    PC ←0x8000 0000 + vector offset
endif
```

Figure 4.25
External Vectored
Interrupt Exception

```
Cause ←BD || BT || Cause[29:0]
if ((CCC24 = 1) | (SR1 = 0)) then
    EPC ←PC
endif
if (CCC24 = 0) then
    SR ←SR[31:2] || 1 || SR0
else
    SR ←SR[31:6] || SR[3:0] || 02
endif
PC ←EXVAp[31:2] || 02
```

4.4.2 Precision of Exceptions

Exceptions are logically precise. This means that the instruction that causes an exception and all those that follow it are aborted, generally before committing to any state; execution picks up where it left off before the exception; and the instruction can be re-executed after the exception has been serviced. When following instructions are killed, exceptions associated with those instructions are also killed, so that exceptions are not taken in the order detected, but in the instruction fetch order.

Interrupts generated by external devices attached to the processor have a variety of meanings, depending on the system environment into which the CW4010 core is designed. Variations in memory system design can affect the meaning of bus error exceptions and the location and means of accessing relevant parameters to service them. As far as possible, this architectural description of the exception handling system details which state information is reliable and which is unreliable.

In some cases, however, the characteristics of the pipeline staging cannot guarantee that all states in the processor and associated system will remain completely unchanged. This is because it is possibly the incomplete execution of instructions immediately following an instruction that has caused an exception. State changes that may occur include the following:

- Instructions may be read from memory and loaded into the instruction cache.
- The multiply/divide registers (HI and LO) may have been altered by a MULT/MULTU, DIV/DIVU, or MTHI/MTLO instruction.

These changes can normally be ignored because the state of the machine is sufficiently restored, allowing execution to resume after the exception has been serviced.

4.4.3 Exception Vector Locations

The Cold Reset, Warm Reset, and NMI exceptions are always vectored to location 0xBFC00000. Addresses for other exceptions are a combination of a vector offset and a base address, and they are determined by the BEV bit of the Status register. Table 4.5 shows the vector base addresses and Table 4.6 shows the vector offsets.

Table 4.5
Exception Vector
Base Addresses

BEV	R4000 Mode (CCC24 = 0)	R3000 Mode (CCC24 = 1)
0	0x80000000	0x80000000
1	0xBFC00200	0xBFC0100

Table 4.6
Exception Vector
Offset Addresses

Exception	R4000 Mode (CCC24 = 0)	R3000 Mode (CCC24 = 1)
TLB refill	0x000 (EXL = 0)	0x000 (kuseg access)
Debug	0x040	0x040
All Others	0x180	0x080

4.4.4 Priority of Exceptions

While more than one exception can occur for a single instruction, only one exception is reported. Table 4.7 shows the priority order given to the exception, with Cold Reset having the highest priority.

Table 4.7
Exception Priority
Order

Priority
Cold Reset
Warm Reset
NMI
Address Error - Instruction Fetch
TLB Refill - Instruction Fetch
TLB Invalid - Instruction Fetch
Bus Error
Integer Overflow, Trap, System Call, Breakpoint, Reserved Instruction, CoProcessor Unusable, Floating-Point Error
Address Error - Data Access
TLB Refill - Data Access
TLB Invalid - Data Access
TLB Modified - Data Write
Interrupt
External Vectored Interrupt
Debug

4.4.5 Cold Reset Exception

The primary purpose of a cold reset is to initialize the CW4010 core at power up. This section describes the cause of and response to a Cold Reset exception.

4.4.5.1 Cause

The Cold Reset exception occurs when the CRESETn signal is asserted and then deasserted. This exception is not maskable.

4.4.5.2 Handling

The CPU provides a special interrupt vector (0xBFC00000) for the Cold Reset exception. The reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to handle the exception. The processor can fetch and execute instructions while the caches and virtual memory are in an undefined state.

The contents of all registers in the CPU are undefined when the Cold Reset exception occurs except for the following:

- In the Status register, the CU[3:0] and SR bits are cleared to 0 and the ERL and BEV bits are set to 1. Other bits are undefined
- The Random register is initialized to the value of its upper bound
- The Wired register is initialized to 0

4.4.5.3 Servicing

The Cold Reset exception is serviced by initializing all processor registers, coprocessor registers, caches, and the memory system. Servicing is accomplished by performing diagnostic tests, and by bootstrapping the operating system.

4.4.6 Warm Reset Exception

The primary purpose of the Warm Reset exception is to reinitialize the processor after a fatal error. Unlike non-maskable interrupts, all cache and bus state machines are reset by this exception. Like Cold Reset, it can be used on the processor in any state. The caches, TLB, and normal exception vectors need not be properly initialized. This section describes the cause of and response to a Warm Reset exception.

4.4.6.1 Cause

The Warm Reset exception occurs when the WRESETn signal is asserted and then deasserted. This exception is not maskable.

4.4.6.2 Handling

The reset exception vector (0xBFC00000) is used for this exception. The vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to handle the exception. The SR bit of the Status register is set to distinguish between a Warm Reset exception and a Cold Reset exception.

The contents of all registers are preserved when the Warm Reset exception occurs, except for the following:

- The ErrorPC register, which contains the restart PC (Program Counter
- The BEV and SR bits of the Status register, which are set to 1
- R4000 mode, in which the ERL bit is set to 1
- R3000 mode, in which KUo/IEo ← KUp/IEp ← KUc/IEc ← 0/0

Because Warm Reset can abort cache and bus operations, cache and memory state is undefined when the Warm Reset exception occurs. Refer to Figure 4.8, on page 4-16 for further information.

4.4.6.3 Servicing

The Warm Reset exception is serviced by saving the current processor state for diagnostic purposes, and reinitializing in a manner similar to that for the Cold Reset exception.

4.4.7 Non-Maskable Interrupt (NMI) Exception

Non-Maskable interrupts cannot be disabled. They occur when a catastrophic event, such as power failure, requires immediate attention to maintain system integrity.

4.4.7.1 Cause

The Non-Maskable Interrupt exception occurs in response to the falling edge of the NMI pin. As the name implies, the NMI exception is not maskable, and occurs regardless of the settings of the EXL, ERL, and IE Status register bits.

4.4.7.2 Handling

The reset exception vector (0xBFC00000) is also used for this exception. The reset vector resides in unmapped and uncached CPU address

space, so the hardware need not initialize the TLB or the cache to handle the NMI interrupt. The SR bit of the Status register is set to differentiate the NMI exception from a Cold Reset exception.

Because an NMI could occur in the middle of another exception, it is generally not possible to continue program execution after servicing an NMI.

Unlike Cold and Warm Reset, but in common with other exceptions, NMI is taken only at instruction boundaries. The states of the caches and memory system are preserved by this exception.

The contents of all registers in the CPU are preserved when this exception occurs, except for the following:

- The ErrorPC register, which contains the restart PC
- The BEV and SR bits of the Status register, which are set to 1
- R4000 mode, in which the ERL bit is set to 1
- R3000 mode, in which KUo/IEo ← KUp/IEp ← KUc/IEc ← 0/0

4.4.7.3 Servicing

The NMI exception is serviced by saving the current processor state for diagnostic purposes and reinitializing the system in a manner similar to that for the Cold Reset exception.

4.4.8 Address Error Exception

This section describes the cause of and response to an Address Error exception.

4.4.8.1 Cause

The Address Error exception occurs when an attempt is made to:

- Load, fetch, or store a word that is not aligned on a word boundary
- Load or store a halfword that is not aligned on a halfword boundary
- Reference the Kernel address space from User mode

The Address Error exception is not maskable.

4.4.8.2 Handling

The common exception vector is used for this exception. The Cause register ExcCode is set based on the type of reference that caused the exception: AdEL for a data load or instruction fetch, AdES for a data store operation.

When the Address Error exception occurs, the BadVAddr register retains the virtual address that was not properly aligned or that referenced protected address space. The contents of the VPN field of the Context and EntryHi registers are undefined, as are the contents of the EntryLo register.

The EPC register points at the instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.8.3 Servicing

The process executing at the time should be handed a “segmentation violation” signal. This error is usually fatal to the process incurring the exception.

4.4.9 TLB Refill Exception

This section describes the cause of and response to a TLB Refill exception.

4.4.9.1 Cause

The TLB Refill exception occurs when there is no TLB entry to match a reference to a mapped address space. This exception is not maskable.

4.4.9.2 Handling

A special TLB Refill exception vector is for this exception. The Cause register ExcCode is set based on the type of reference that caused the exception: TLBL for a data load or instruction fetch and TLBS for a data store operation.

When the TLB refill exception occurs, the BadVAddr, Context, and EntryHi registers hold the virtual address that failed translation. The EntryHi register also contains the ASID from which the translation fault occurred. The Random register normally contains a valid location in

which to place the replacement TLB entry. The contents of the EntryLo register are undefined.

The EPC register points at the instruction that caused the exception unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

R4000 mode – This special exception vector is used when the exception level (at the time of TLB miss detection) is set to normal (EXL = 0). If the exception level is exception (EXL = 1), the common exception vector is used.

R3000 mode – This special exception vector is used when User or Kernel mode references to user memory space (kuseg) do not find a matching entry in the TLB. If the reference is to kernel memory space (kseg0:2), the common exception vector is used.

4.4.9.3 Servicing

To service this exception, the contents of the Context register are used as a virtual address to fetch memory locations containing the physical page frame and access control bits for a TLB entry. This information is placed in the EntryHi and EntryLo registers and written into the TLB.

It is possible that the virtual address used to obtain the physical address and access control information is on a page that is not resident in the TLB. In this case, a TLB refill exception is allowed inside the TLB Refill handler. While the first exception goes to a special exception vector offset (0x000), the second exception goes to the common exception vector offset (0x180).

The second TLB refill exception obscures the contents of the BadVAddr, Context, and EntryHi registers within the TLB Refill handler. As a result, the exact virtual address whose translation caused the first fault is not known unless the TLB Refill handler specifically saved this address. It is possible to observe only the failing PTE virtual address. The BadVAddr register now contains the original contents of the Context register within the TLB Refill handler, which is the PTE address for the original failing address.

The operating system can determine the original virtual page number that caused the fault, but not the complete address. The operating

system uses this information to fetch the PTE that contains the physical address and to access control information. It also writes the entry into the TLB and returns to the original user program.

Returning to the TLB Refill handler at this point should be avoided.

R4000 mode – When the EXL bit is set, it prevents the EPC from the first TLB refill exception from being overwritten by the second TLB refill exception. Consequently, the appropriate return address can be determined from the values of the current EPC and the BD bit of the Status register.

R3000 mode – The TLB Refill handler must save the first refill EPC and Status[BD] information in a way that allows the second refill to find it. Using this “saved” EPC register and Status[BD] information, the appropriate return address can be determined.

4.4.10 TLB Invalid Exception

This section describes the cause of and response to a TLB Invalid exception.

4.4.10.1 Cause

The TLB Invalid exception occurs when a virtual address reference matches a TLB entry that is marked invalid. This exception is not maskable.

4.4.10.2 Handling

The common exception vector is used for this exception. The Cause register ExcCode is set based on the type of reference that caused the exception: TLBL for a data load or instruction fetch, TLBS for a data store operation.

When the TLB Invalid exception occurs, the BadVAddr, Context, and EntryHi registers hold the virtual address that failed translation. The EntryHi register also contains the ASID from which the translation fault occurred. The Random register normally contains a valid location in which to place the replacement TLB entry. The contents of the EntryLo register are undefined.

The EPC register points at the instruction that caused the exception, unless this instruction is in a Branch Delay slot. If the instruction is in a

Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.10.3 Servicing

The valid bit of the TLB entry is typically cleared when:

- A virtual address does not exist
- The virtual address exists, but is not in main memory (a page fault)
- A trap is desired on any reference to the page (for example, to maintain a reference bit)

After servicing the cause of this exception, the TLB entry is located with the TLB Probe (TLBP) instruction, and replaced by an entry with the valid bit set.

4.4.11 TLB Modified Exception

This section describes the cause of and response to a TLB Modified exception.

4.4.11.1 Cause

The TLB Modified exception occurs during a store operation when the virtual address reference to memory matches a TLB entry that is marked valid but is not dirty or writable. This exception is not maskable.

4.4.11.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to the Mod.

When the TLB Modified exception occurs, the BadVAddr, Context, and EntryHi registers hold the virtual address that failed translation. The EntryHi register also contains the ASID from which the translation fault occurred. The Random register normally contains a valid location in which to place the replacement TLB entry. The contents of the EntryLo register are undefined.

The EPC register points at the instruction that caused the exception, unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set.

4.4.11.3 Servicing

The Kernel uses the failed virtual address and virtual page number to identify the corresponding access control information. The page identified may or may not permit write access. If writes are not permitted, a Write Protection Violation has occurred.

If write access is permitted, the Kernel marks the page frame as dirty/writable in the Kernel's own data structures. The TLBP instruction is used to place the index of the TLB entry that must be altered in the Index register. The EntryLo registers are loaded with physical page frame and access control bits (with the D bit set), and the EntryHi and EntryLo registers are written into the TLB.

4.4.12 Bus Error Exception

This section describes the cause of and response to a Bus Error exception.

4.4.12.1 Cause

The Bus Error exception occurs when signaled by board-level circuitry for events such as bus time-out, bus parity errors, and invalid physical memory accesses. This exception is not maskable.

In the CW4010, Bus Errors are asynchronous events with respect to CPU instruction processing (much like the NMI interrupt). This means that there is no attempt to identify the instruction that was the root source of the error.

4.4.12.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to Bus.

The EPC register points at the first instruction for which processing did not complete unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.12.3 Servicing

The physical address at which the fault occurred is not available to the exception handler. The process executing at the time of the exception must be handed a “bus error” signal, which is usually fatal.

4.4.13 Integer Overflow Exception

This section describes the cause of and response to an Integer Overflow exception.

4.4.13.1 Cause

The Integer Overflow exception occurs when an ADD, ADDI, SUB, DADD, DADDI, or DSUBI instruction results in a two's complement overflow. This exception is not maskable.

4.4.13.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to OV.

The EPC register points at the instruction that caused the exception unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.13.3 Servicing

The process executing at the time of the exception should be handed an "integer overflow" signal. This error is usually fatal to the current process.

4.4.14 Trap Exception

This section describes the cause of and response to a Trap exception.

4.4.14.1 Cause

The Trap exception occurs when a TGE, TGEU, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI, or TNEI instruction results in a TRUE condition. This exception is not maskable.

4.4.14.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to TR.

The EPC register points at the instruction that caused the exception unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.14.3 Servicing

The process executing at the time of the exception should be handed a “trap” signal. This error is usually fatal.

4.4.15 System Call Exception

This section describes the cause of and response to a System Call exception.

4.4.15.1 Cause

The System Call exception occurs when an attempt is made to execute the SYSCALL instruction. This exception is not maskable.

4.4.15.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to Sys.

The EPC register points at the SYSCALL instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in the Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set as an indicator.

4.4.15.3 Servicing

When this exception occurs, control is transferred to the applicable system routine. To resume execution, the routine must restart instruction execution after the SYSCALL instruction. This restart address can be computed using the EPC register along with the BD and BT bits in the Cause register.

- If (BD = 0) then Restart_PC = EPC + 4
- If ((BD = 1) and (BT = 0)) then Restart_PC = EPC + 8
- If ((BD = 1) and (BT = 1)) then Restart_PC = Branch Target Address

It is up to the exception handler to obtain the Branch Target Address from the prior branch when the SYSCALL instruction resides in a Branch Delay slot.

4.4.16 Breakpoint Exception

This section describes the cause of and response to a Breakpoint exception.

4.4.16.1 Cause

The Breakpoint exception occurs when an attempt is made to execute the BREAK instruction. This exception is not maskable.

4.4.16.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to BP.

The EPC register points at the BREAK instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set as an indicator.

4.4.16.3 Servicing

When the Breakpoint exception occurs, control is transferred to the applicable system routine. Additional distinctions can be made from the unused bits of the BREAK instruction (bits [25:6]), by loading the contents of the instruction at which the EPC register points. (A value of four must be added to the EPC register to locate the instruction if it resides in a Branch Delay slot).

To resume execution, the routine must start executing the instruction again after the BREAK instruction. The restart address can be computed using the EPC register along with the BD and BT bits held in the Cause register.

- If (BD = 0) then Restart_PC = EPC + 4
- If ((BD = 1) and (BT = 0)) then Restart_PC = EPC + 8
- If ((BD = 1) and (BT = 1)) then Restart_PC = Branch Target Address

When the BREAK instruction resides in a Branch Delay slot, it is up to the exception handler to obtain the Branch Target Address from the prior branch.

4.4.17 Reserved Instruction Exception

This section describes the cause of and response to a Reserved Instruction exception.

4.4.17.1 Cause

The Reserved Instruction exception occurs when an attempt is made to execute an instruction whose major opcode (bits [31:26]) are undefined, or a SPECIAL instruction whose minor opcode (bits [5:0]) are undefined. This exception also occurs on a REGIMM instruction whose minor opcode (bits [20:16]) are undefined. This exception is not maskable.

4.4.17.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to RI.

The EPC register points at the BREAK instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.17.3 Servicing

The Reserved Instruction exception can be used to trap to emulation routines for instructions not supported in the CW4010 instruction set. Once emulation has been completed, execution can be resumed using the EPC register along with the BD and BT bits in the Cause register.

- If (BD = 0) then Restart_PC = EPC + 4
- If ((BD = 1) and (BT = 0)) then Restart_PC = EPC + 8
- If ((BD = 1) and (BT = 1)) then Restart_PC = Branch Target Address

When the instruction receiving a Reserved Instruction exception resides in a Branch Delay slot, it is up to the exception handler to obtain the Branch Target Address from the prior branch.

If there is no emulation routine, the process executing at the time of the exception should be given an “illegal instruction” signal. This error is usually fatal.

4.4.18 Floating-Point Exception

This section describes the cause of and response to a floating-point exception.

4.4.18.1 Cause

The Floating-Point exception is used by the floating-point coprocessor (if installed). The contents of the Floating-Point Control Status register (inside CP1) indicate the cause of the exception.

4.4.18.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to FPE.

The EPC register points at the first instruction for which processing was not completed unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.18.3 Servicing

This exception is cleared by clearing the appropriate bit in the Floating-Point Control Status register. For an unimplemented instruction exception, the Kernel should emulate the instruction. For other exceptions, the Kernel should pass the exception to the user process that caused the exception.

4.4.19 Coprocessor Unusable Exception

This section describes the cause of and response to a Coprocessor Unusable exception.

4.4.19.1 Cause

The Coprocessor Unusable exception occurs when an attempt is made to execute a coprocessor instruction for either a corresponding coprocessor unit that has not been marked usable, or for CP0 instructions, when the unit has not been marked usable and the process is executing in User mode.

This exception is not maskable.

4.4.19.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to CPU. The contents of the CE field in the Cause register indicate the coprocessor to which an attempted reference has been made.

The EPC register points at the instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

4.4.19.3 Servicing

The coprocessor unit to which an attempted reference was made is identified by the CE field of the Cause register. The result is one of the following:

- If the process is entitled to access, the coprocessor is marked usable and the corresponding user state is restored.
- If the process is entitled to access the coprocessor, but the coprocessor does not exist or has failed, interpretation of the coprocessor instruction is possible.
- If the process is not entitled to access the coprocessor, the process executing at the time should be given some sort of “Illegal/Privileged Instruction” signal. This error is usually fatal.

4.4.20 Debug Exception

This section describes the cause of and response to a Debug exception.

4.4.20.1 Cause

The Debug exception occurs when a debug condition (read/write access at Breakpoint Data Address, read access at Breakpoint Program Counter, Trace) is detected by the CP0. The Debug Control and Status (DCS) register specifies which event was detected.

R4000 mode – In R4000 mode, the debug exception can be masked by setting the EXL bit in the Status register. When this bit is set, a debug event does not cause an exception trap even if the DCS[TE] bit is set to 1. However, the status bits of the DCS register are updated to indicate that an event was recognized.

R3000 mode – In R3000 mode, the debug exception is not maskable.

4.4.20.2 Handling

The Debug exception vector handles this exception.

4.4.20.3 Servicing

The Debug exception is a debugging aid. Typically the exception handler transfers control to a debugger, allowing you to examine the situation. The debug exception condition must be disabled to execute the failing instruction and then re-enabled.

Notes:

1. The Trace status bit (DCS5) is set whenever a branch instruction is encountered regardless of whether the branch is actually taken. However, if the debug exception trap is enabled (DCS31 = 1), an exception is recognized only if the branch is taken and the target instruction executed.
2. The Program Counter debug status bit (DCS1) is set whenever the target address of a branch falls within the specified PC address range (BPC, BPCM) regardless of whether the branch is actually taken. However, if the debug exception trap is enabled (DCS31 = 1), an exception is recognized only if the branch is taken and the target instruction executed.

4.4.21 Interrupt Exception

This section describes the cause of and response to an Interrupt exception.

4.4.21.1 Cause

The Interrupt exception occurs when one of the eight interrupt conditions is asserted. The significance of these interrupts depends on the specific system implementation. Each of the eight interrupts can be masked by clearing the corresponding bit in the Int-Mask field of the Status register. All eight interrupts can be masked at once by clearing the IE bit of the Status register.

4.4.21.2 Handling

The common exception vector is used for this exception. The ExcCode field in the Cause register is set to INT.

The IP field of the Cause register indicates the current interrupt requests. It is possible that more than one of the bits will be set at the same time, or that no bits will be set if an interrupt is asserted and then deasserted before the Cause register is read.

The EPC register points at the first instruction for which processing was not completed unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set as an indicator.

4.4.21.3 Servicing

If the interrupt is caused by one of the two software generated exceptions, the interrupt condition is cleared by setting the corresponding Cause register bit to 0.

If the interrupt is hardware generated, the interrupt condition is cleared by correcting the condition causing the interrupt pin to be asserted.

4.4.22 External Vectored Interrupt Exception

The CW4010 implements an external vectored interrupt interface, which consists of an interrupt input (EXViNTn), interrupt vector virtual address input (EXVAp[31:2]), and interrupt accepted output (EXVAEn). The signals must be asserted and deasserted on the rising edge of the system clock. This interrupt class can be enabled or disabled using the EVI bit in the CCC register (enabled when CCC24 = 1). This section describes the cause of and response to an external vectored interrupt exception.

4.4.22.1 Cause

An external vectored interrupt occurs when the EXViNTn is asserted. The significance of this interrupt depends on the specific system implementation. The interrupt can be masked by clearing the IE (R3000 = IEc) bit of the Status register.

4.4.22.2 Handling

The virtual address specified by the EXVAp[31:2] interface is used to specify the target exception handling routine. The EXVAp[31:2] address must be provided by a user-defined interrupt controller. The EXViNTn and EXVAp[31:2] inputs must be held stable and valid until the exception

is accepted. This is indicated by the assertion of the EXVAEn output for one cycle.

The EPC register points at the first instruction for which processing was not completed unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set as an indicator.

4.4.22.3 Servicing

The interrupt condition can be cleared in the user-defined interrupt controller in one of two ways: by detecting the assertion of the interrupt accepted output (EXVAEn), or by correcting the condition causing the interrupt pin (EXViNTn) to be asserted.

Chapter 5
CW4010 Memory Management

This chapter describes the System Coprocessor (Coprocessor-0) and Memory Management. It contains the following sections:

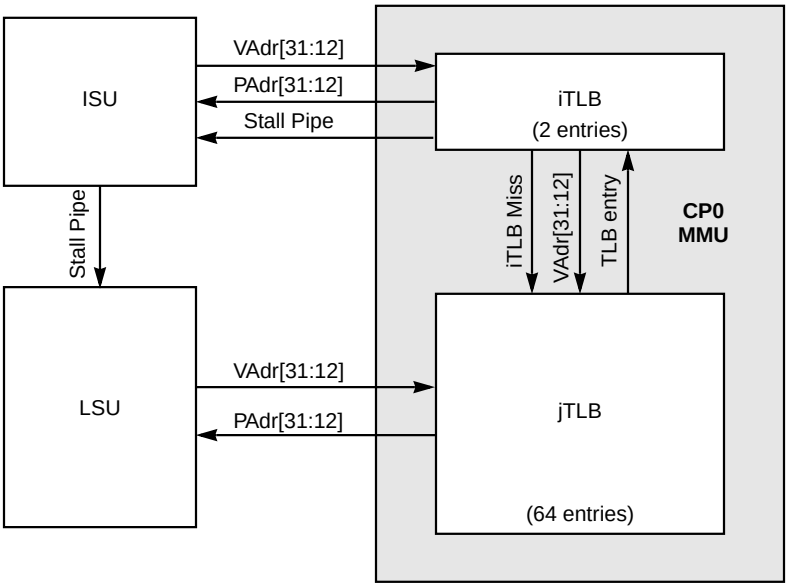
5.1 TLB Physical Organization

The physical implementation of the TLB consists of two main parts:

1. A two-entry instruction TLB (iTLB).
2. A 64-entry joint TLB (jTLB) that holds both instruction fetch and data access page translations.

The CP0 can receive virtual address translation requests from both the ISU (instruction fetch unit) and the LSU (operand data access) during the same cycle. For maximum performance, address translations must occur in parallel. The two-piece TLB structure shown in Figure 5.4 addresses this problem by creating a separate two-entry TLB to be used for instruction fetch translations. With this structure, ISU and LSU fetches can be independently processed.

Figure 5.1
TLB Block Diagram



1. VAdr = Virtual address
2. PAdr = Physical address

MD96.88

The iTLB holds the two most recently used instruction fetch page translations. If a valid translation cannot be found in the iTLB, the CP0 must stall the pipeline for two cycles and search the jTLB for a valid entry. If the CP0 finds a valid entry in the jTLB, it copies it into the less recently used iTLB entry and processing continues. If a valid entry cannot be found, a TLB exception must be posted (see Chapter 4, “CW4010 Exception Processing,” for details.)

The entries in the iTLB are purged when the EntryHi register is written (for example, during a task switch). Consequently, the iTLB does not need to keep an eight-bit ASID for each entry. This reduces storage and match circuitry. This simplification should cause little or not performance penalty, because the entries probably need to be replaced anyway.

When no TLB is present in the system, the TE field of the Configuration and Cache Control (CCC) register is cleared to 0. This is transparent to the other modules in the CW4010 core. The CP0 modifies its translation behavior in the following manner:

- Physical Address[31:12] = Virtual Address[31:12]

(For *kseg0* and *kseg1*, Physical Address [31:29] = 0); the same is true with TLB present

- The caching algorithm used for each access is based on the address segment being accessed (*kuseg*, *kseg0*, and *kseg2* = cached; *kseg1* = uncached), and the CCC register fields (IE0, IE1, DE0, DE1, and WB). Table 5.1 shows the caching algorithm criteria.

Table 5.1
Caching Algorithm Criteria

Address Segment	Icache Enabled	Ifetch Cache Algorithm	Dcache Enabled	WB	Dcache Algorithm
kuseg	0	Uncached	0	X	Uncached
kseg0	1	Cached	1	0	Cached, WriteThrough
kseg2			1	1	Cached, WriteBack
kseg1	X	Uncached	X	X	Uncached

5.2 Memory Management System

The memory model used for the CW4010 processor is based on the R3000. To extend the CPU's address space, the virtual memory translates addresses composed in a large virtual address space into the physical memory system.

The CW4010 physical address space is 4 Gbytes and uses a 32-bit address. The virtual address is also 32 bits wide, and the maximum user process size is 2 Gbytes (2^{31}).

The virtual address is extended with an Address Space Identifier (ASID) to reduce the frequency of the Translation Lookaside Buffer (TLB) flushing when switching context. The size of the ASID is 8 bits. The ASID is contained in the CP0 EntryHi register and is described in the subsection entitled "EntryHi Register (10)" on page 5-9.

5.2.1 Operating Modes

This section describes the two modes for 32-bit CW4010 operation:

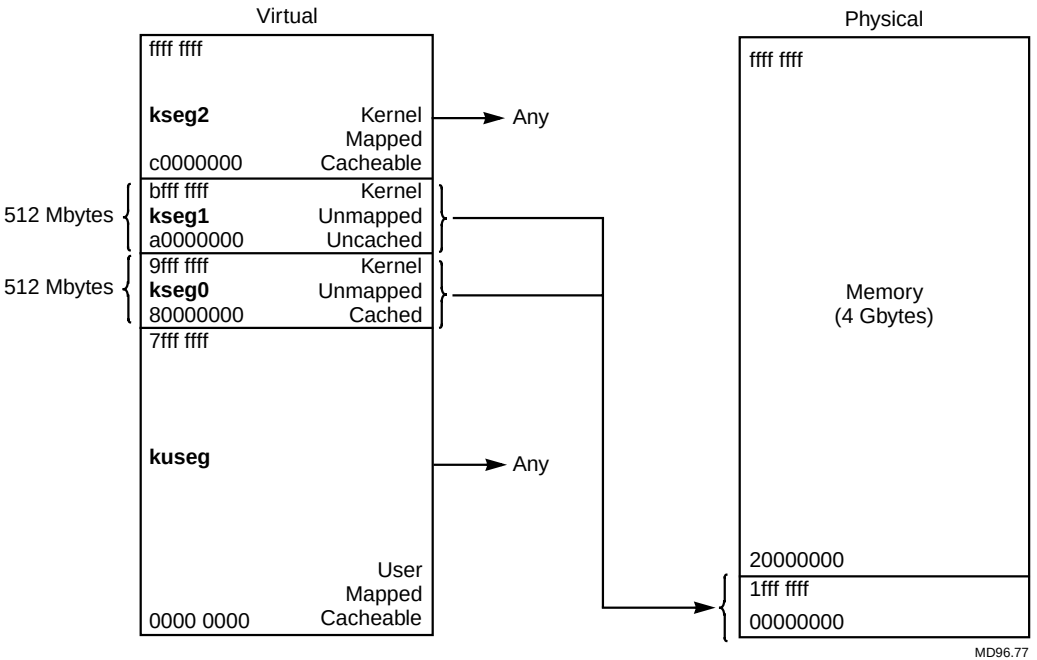
- User mode, where non-supervisory programs are executed
- Kernel mode, which is analogous to the "supervisory" mode provided by many machines

The CW4010 usually operates in User mode until an exception forces it into Kernel mode. It remains in Kernel mode until a Restore From

Exception instruction (R3000 mode), or Exception Return (R4000 mode) instruction is executed to restore the processor to the mode existing prior to the exception.

Address mapping is different for Kernel and User modes. To simplify the management of user state from within the Kernel, the user-mode address space is a subset of the Kernel-mode address space. Figure 5.2 shows the virtual-to-physical memory map for both the User mode and Kernel mode segments.

Figure 5.2
CW4010 Virtual
Memory Map



5.2.2 User Mode Virtual Addressing

In User mode, a single, uniform virtual address space (*kuseg*) of 2 Gbytes (2^{31} bytes) is available. The User segment starts at address 0x0000 0000, and all valid accesses have the most-significant bit cleared to zero. Referencing an address with the most significant bit set while in User mode causes an Address Error exception. The TLB maps all references to *kuseg* identically for either mode, and controls cache accessibility. *Kuseg* is typically used to hold user code and data, as well

as the current user process. The processor state definition of User and Kernel modes description can be found in Section 4.3.6, “Status Register (12)” on page 4-9.

5.2.3 Kernel Mode Virtual Addressing

The virtual address space is divided into regions, differentiated by the high-order bits of the address, as shown in Figure 5.2 on page 5-4 and as listed below.

kuseg Starts at virtual address 0x00000000 and is 2 Gbytes long. It allows selective caching and mapping on a per-page basis, rather than requiring an all or nothing approach. This segment overlaps Kernel memory accesses with User memory accesses as described above.

kseg0 Starts at virtual address 0x80000000 and is 512 Mbytes long. CW4010 direct maps references within *kseg0* onto the first 512 Mbytes of physical memory. These references use cache memory, but do not use the TLB for address translation. Thus, *kseg0* is typically used for kernel executable code and some kernel data.

kseg1 Starts at virtual address 0xA0000000 and is 512 Mbytes long. CW4010 direct maps references within *kseg1* onto the first 512 Mbytes of physical memory. These references do not use cache memory or the TLB for address translation. Thus, *kseg1* is typically used by operating systems for I/O registers, ROM code and disk buffers.

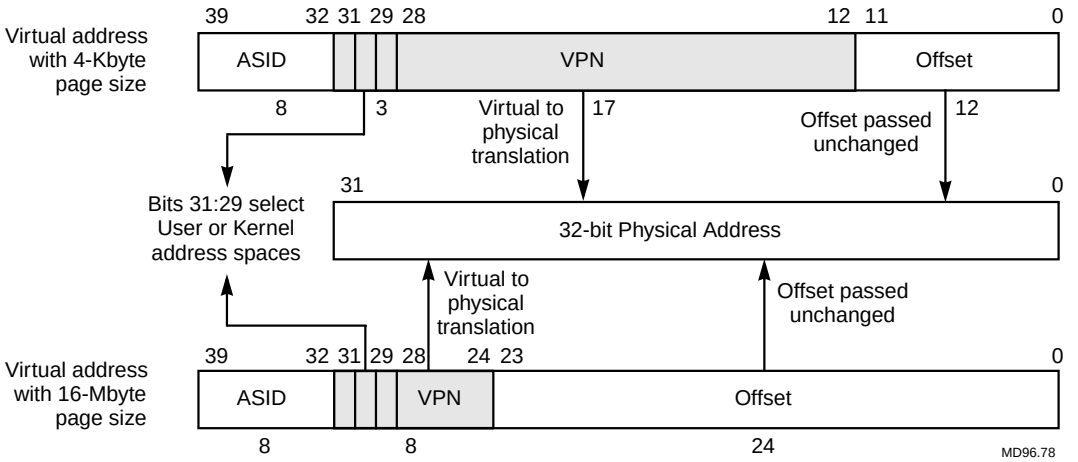
kseg2 Starts at virtual address 0xC0000000 and is 1024 Mbytes long. Like *kuseg*, it uses TLB entries to map virtual addresses to arbitrary physical ones, with or without caching. An operating system typically uses *kseg2* for stacks and per-process data that must remap on context switches. The operating system also uses *kseg2* for user page tables and some dynamically allocated data areas.

5.3 Virtual Memory and the TLB

Mapped virtual addresses are translated into physical addresses using an on-chip Translation Lookaside Buffer (TLB). The TLB is a fully-associative memory that holds 64 entries that provide mapping to 64 physical page frames. The address range mapped by a page can be either 4 Kbytes or 16 Mbytes in size. When address mapping is indicated, each TLB entry is simultaneously matched against the virtual address extended by the current ASID stored in the EntryHi register.

If there is a match (hit), the physical page number is extracted from the TLB and concatenated with the offset to form the physical address, as shown in Figure 5.3.

Figure 5.3
CW4010 Virtual
Address Format



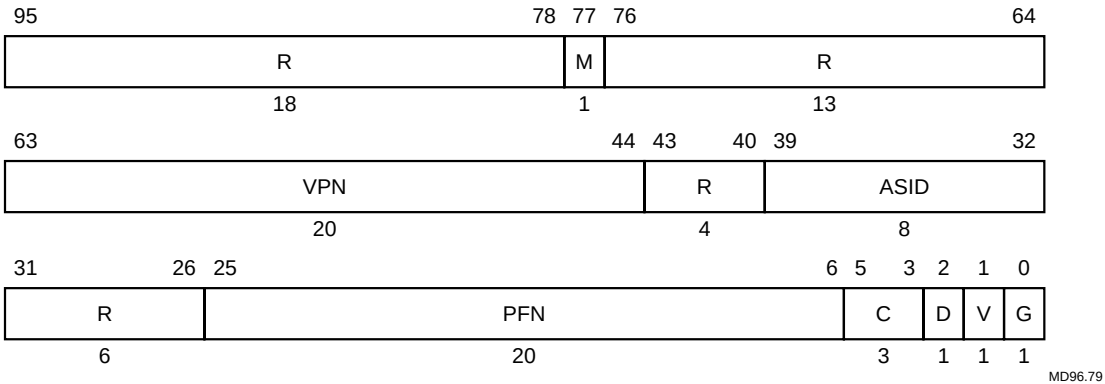
If no match occurs (a page miss), an exception is taken. Typically, software refills the TLB from a page table maintained by the system. Software can write over a selected TLB entry or use a hardware mechanism to write into a random location.

The CW4010 does not support the TLB-shutdown (TS) bit in the Status register, which indicates that more than one entry in the TLB matches the virtual address being translated. If more than one TLB entry matches the virtual address, the virtual address may be translated to an incorrect physical address. System software must ensure that this situation is never created.

5.3.1 TLB Entry Format

Figure 5.4 shows the 32-bit addressing TLB entry format the CW4010 uses. Each field of an entry has a corresponding field in the EntryHi, EntryLo, or PageMask registers described in sections 5.3.2.1, 5.3.2.2, and 5.3.2.3, beginning on page 5-9.

Figure 5.4
Format of CW4010 TLB Entry



R **Reserved** **[95:78, 76:64, 43:40, 31:26]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

M **Mask** **77**
This bit is the Page Mask bit. It is set to 1 for a 16M page and cleared to 0 for a 4K page.

VPN **Virtual Page Number** **[63:44]**
This field contains the Virtual Page Number.

ASID **Address Space ID Field** **[39:32]**
This field contains the Address Space ID.

PFN **Page Frame Number** **[25:6]**
This field contains the Page Frame Number. This is the upper bits of the physical address.

C **Cache** **[5:3]**
This field contains the Cache algorithm, which specifies whether references to the page should be cached. If the references are to be cached, you can select one of two

algorithms: WriteBack or WriteThrough. Table 5.2 shows how the Cache bits are decoded.

Table 5.2
Cache Algorithm
Bit Values

CBit Settings			Value	Algorithm
0	0	0	0	Reserved
0	0	1	1	Reserved
0	1	0	2	Uncached
0	1	1	3	Cacheable—WriteThrough
1	0	0	4	Reserved
1	0	1	5	Reserved
1	1	0	6	Reserved
1	1	1	7	Cacheable—WriteBack

D **Dirty** **2**
If this bit is set to 1, it indicates that the page marked is dirty and writable.

V **Valid** **1**
If this bit is set to 1, it indicates that the TLB entry is valid.

G **Global** **0**
If this bit is set to 1, you can ignore the contents of the ASID field during TLB lookup.

5.3.2 TLB Support Registers

The registers described in the following subsections are used in association with the CP0 TLB.

- EntryHi Register (10)
- EntryLo Register (2)
- PageMask Register (5)
- Index Register (0)
- Random Register (1)
- Wired Register (6)

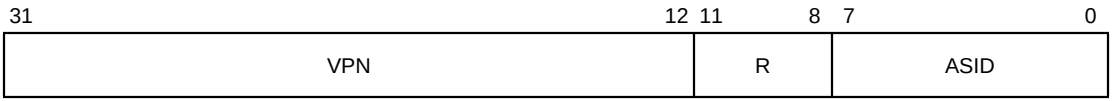
5.3.2.1 EntryHi Register (10)

The EntryHi register is a read/write register used to access the TLB. In addition, this register contains the current ASID value for the processor. The ASID value is used to match the virtual address with a TLB entry during virtual address translation. Typically, the operating system assigns a unique ASID value to each known process. In this way, mappings held in the TLB are made unique to the process whose ASID they match.

The EntryHi register holds the high-order bits of a TLB entry when performing TLB read and write operations. When either a TLB refill, TLB invalid, or TLB modified exception occurs, the EntryHi register is loaded with the Virtual Page Number (VPN) and the ASID of the virtual address that failed to have a matching TLB entry.

EntryHi is accessed by the TLBP, TLBW, TLBWI, and TLBR instructions. Figure 5.5 shows the format of this register.

Figure 5.5
EntryHi Register



VPN **Virtual Page Number** **[31:12]**
This field contains the Virtual Page Number.

R **Reserved** **[11:8]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software

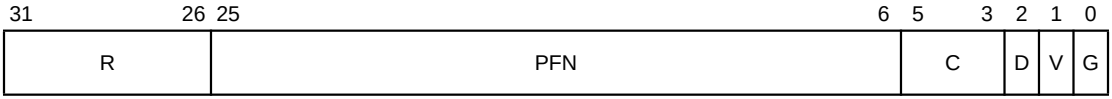
should write these bits as 0 to ensure compatibility with future versions of the software.

ASID **Address Space ID** **[7:0]**
This field contains the Address Space ID.

5.3.2.2 EntryLo Register (2)

The EntryLo register is a read/write register used to access the TLB. When performing read and write operations, the register contains a physical page frame number, cache algorithm, page dirty, translation valid, and global entry information. Figure 5.6 shows the format of this register.

Figure 5.6
EntryLo Register



R **Reserved** **[31:26]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

PFN **Physical Page Frame Number** **[25:6]**
This field contains the Physical Page Frame Number.

C **Cache** **[5:3]**
This field contains the Cache algorithm, which specifies whether references to the page should be cached. If the references are to be cached, you can select one of two algorithms: write-back or write-through. Table 5.2 shows how the Cache bits are decoded.

D **Dirty** **2**
If this bit is set to 1, it indicates that the page marked is dirty and writable.

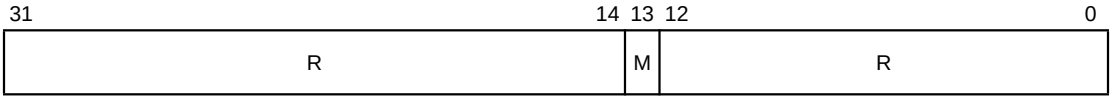
V **Valid** **1**
If this bit is set to 1, it indicates that the TLB entry is valid.

G **Global** **0**
If this bit is set to 1, you can ignore the contents of the ASID field during TLB lookup. Mapping is globally available to all ASIDs.

5.3.2.3 PageMask Register (5)

The PageMask register is a read/write register used to access the TLB. It implements a variable page size by holding a per-entry comparison mask. When virtual addresses are presented for translation, the corresponding PageMask bit in the TLB specifies whether or not virtual address bits [23:12] participate in the comparison. Figure 5.7 shows the format of the PageMask register.

Figure 5.7
PageMask Register



R **Reserved** **[31:14, 12:0]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

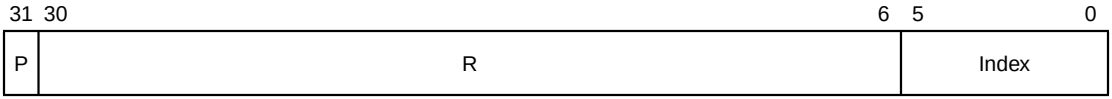
M **Mask** **13**
This field contains the PageMask. If it is set to 1, the page size is 16 Mbytes, the Physical Address [31:0] is PFN[31:24] and the Virtual Address is [23:0]. If the bit is cleared to 0, page size is 4 Kbytes, the Physical Address [31:0] is PFN[31:12] and the Virtual Address is [11:0].

5.3.2.4 Index Register (0)

The Index register is a 32-bit, read/write register containing six bits that are used to index an entry in the TLB. The high-order bit indicates the success or failure of a TLB Probe (TLBP) instruction.

The Index register also specifies the TLB entry that is affected by the TLB Read (TLBR) and TLB Write Index (TLBWI) instructions. Figure 5.8 shows the format of the Index register.

Figure 5.8
Index Register



P **Probe** **31**
If this bit is set to 1, it indicates that the last TLBP instruction failed to find a match.

R **Reserved** **[30:6]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

Index **Index** **[5:0]**
This field contains the index to the TLB entry. The TLBR and TLBWI instructions use this index.

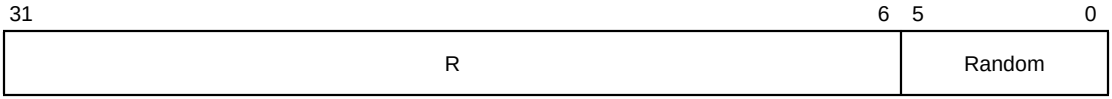
5.3.2.5 Random Register (1)

The Random register is a 32-bit read-only register that contains six bits that are used to index an entry in the TLB. The register decrements for each clock cycle. The values range between a lower bound set by the number of TLB entries reserved for exclusive use by the operating system (defined in the Wired register), and an upper bound set by the total number of TLB entries (64 maximum).

The Random register specifies the entry in the TLB affected by the TLB Write Random (TLBWR) instruction. The register does not need to be read for this purpose, but the register can be read to verify proper operation.

To simplify testing, the Random Register is set to the value of the upper bound when the system is reset. It is also set to its upper bound when the Wired register is written. The format of this register is shown in Figure 5.9.

Figure 5.9
Random Register



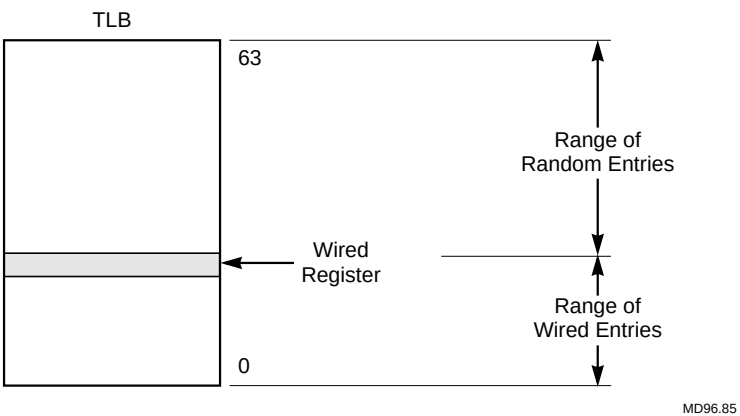
R **Reserved** **[31:6]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

Random **Random** **[5:0]**
This field contains the index to the TLB entry affected by the TLBWR instruction.

5.3.2.6 Wired Register (6)

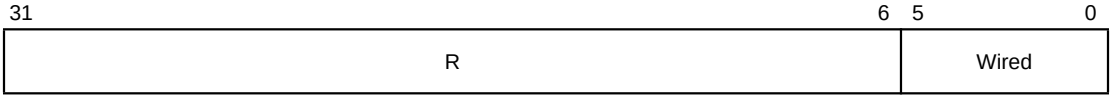
The Wired register is a read/write register that specifies the boundary between the wired (fixed, non-replaceable entries that cannot be over-written by a TLBWR operation) and random entries of the TLB. Figure 5.10 shows the location of the wired register.

Figure 5.10
Wired Register
Location



When the system is reset, the Wired register is set to zero. Writing the register also sets the Random register to the value of its upper bound. Figure 5.11 shows the format of the Wired register.

Figure 5.11
Wired Register



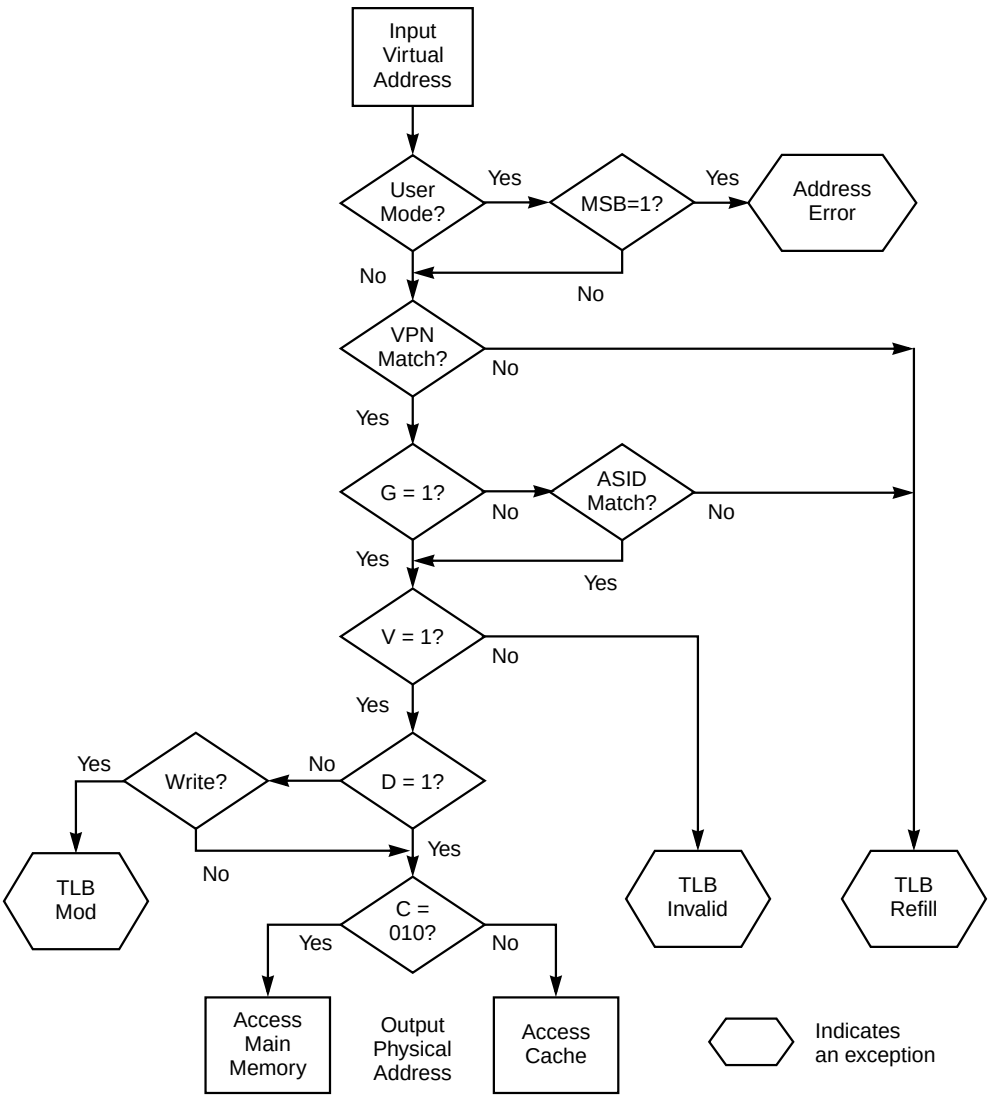
R **Reserved** **[31:6]**
These bits are not used and are read as 0. The CW4010 ignores attempts to set these bits; however, software should write these bits as 0 to ensure compatibility with future versions of the software.

Wired **Wired** **[5:0]**
This field defines the lower boundary of Random TLB entries.

5.3.3 Virtual Address Translation

During virtual-to-physical address translation, the CPU compares the ASID and, depending upon the page size, the highest 7 to 20 bits of the virtual address to the contents of the TLB. Figure 5.12 illustrates the TLB address translation process.

Figure 5.12
CW4010 TLB Address Translation Process



A virtual address matches a TLB entry when:

- VPN field of the virtual address equals the VPN field of the entry
- G bit of the TLB entry is set
- ASID held in the EntryHi register matches the ASID field in the TLB entry

Although the V bit of the TLB entry must be set for a valid translation to take place, it is not involved in the determination of a matching TLB entry.

If a TLB entry matches, the physical address and access control bits (C, D, and V) are retrieved from the entry. If no match is found, a TLB miss exception occurs. If the access control bits (D and V) indicate that the access is not valid, a TLB modification or TLB invalid exceptions occurs, respectively. If the C bits equal 010_b, the physical address that is retrieved is used to access main memory, bypassing the cache.

5.3.4 TLB Instructions

The instructions that the CW4010 provides for working with the TLB are listed in Table 5.3.

Table 5.3
TLB Instruction

Instruction	Description
Translation Lookaside Buffer Probe (TLBP)	The Index register is loaded with the address of the TLB entry whose contents match the contents of the EntryHi register. If no TLB entry matches, the highest order bit of the Index register is set. Results are undefined if a TLB reference encounters more than one matching TLB entry.
Translation Lookaside Buffer Read (TLBR)	This instruction loads the EntryHi, EntryLo, and PageMask registers with the contents of the TLB entry specified by the Index register.
Translation Lookaside Buffer Write Index (TLBWI)	This instruction loads the TLB entry specified by the Index register with the contents of the EntryHi, EntryLo, and PageMask registers.
Translation Lookaside Buffer Write Random (TLBWR)	This instruction loads the TLB entry specified by the Random register with the contents of the EntryHi, EntryLo, and PageMask registers.

Notes:

1. If the TLB is not present or not enabled in the system, the CP0 register reflects a Coprocessor Unusable exception if an attempt is made to execute any of the TLB instructions.
2. TLB instructions (TLBP, TLBR, TLBWI, and TLBWR) cannot be immediately preceded or followed by a data load instruction that requires target address translation (that is, *kuseg* and *kseg*2).
3. The instruction prior to a TLBW instruction must not generate an exception. You are recommended to use an NOP to make sure this restriction is met.
4. Three instructions are needed between MTC0 (EntryHI, EntryLo, PageMask, or Index) and subsequent TLBWI or TLBWR instructions to affect the result of the MTC0 operation.

Chapter 6
CW4010 Caches

This chapter describes the CW4010 cache and cache maintenance. It contains the following sections:

6.1 Cache Memory Organization

The CW4010 has separate caches for instructions and data. These are the Icache and Dcache, respectively. Cache maintenance features include:

- External invalidation for snooping
- Initialization
- WriteBack to Dcache
- Testing

The CW4010 Icache and Dcache are organized as follows:

1. The Icache and Dcache can be organized as direct-mapped or two-way set associative caches. A Least Recently Used (LRU) algorithm is used for two-way set associative cache replacement for the Icache. The Dcache is replaced randomly.
2. The cache controllers support configurations of 1, 2, 4 or 8 Kbyte for each set. Thus, the smallest supported configuration is a 1 Kbyte direct-mapped cache, and the largest is a 16 Kbyte two-way set associative cache, with 8 Kbyte per set.

3. The caches are indexed with a virtual address.
4. They are tagged with a physical address tag.
5. One cache line is 8 words. A single word consists of four 8-bit bytes. The cache line consists of four doublewords (32 bytes or 256 bits). Refill address ordering is wrap-around from the missing address.
6. You can select between WriteBack and WriteThrough modes. If the system has no Memory Management Unit (MMU), the WB bit in the CCC register defines the mode for all cacheable regions of memory. When the WB bit is set to 0, the mode is WriteThrough. When it is set to 1, the mode is WriteBack. If the system has an MMU system, the Translation Lookaside Buffer (TLB) entry determines the mode on a per-page basis.
7. Scratch Pad RAM mode is available, and works in a similar way to the Scratch Pad RAM in the in the LR33300. This is discussed in more detail in Section 6.4, “Dcache Scratch Pad RAM Mode,” on page 6-5.

6.2 Cache States

This section describes cache states for the Icache, WriteThrough Dcache, and WriteBack Dcache.

6.2.1 Icache and WriteThrough Dcache

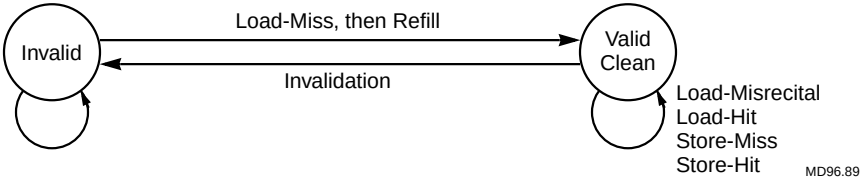
When the Icache and Dcache are operating in WriteThrough mode, only two states are used: Invalid and Valid Clear. Initialization sets all cache lines to the Invalid state. This is done using the Cache Invalidate mode described in Section 6.6.3, “Cache Maintenance by CCC Register,” starting on page 6-7, or the Cache Flush instructions described in Section 6.6.1, “Flush (All Cache Invalidation),” starting on page 6-6.

The first time a cache line is refilled because of a cache miss, its state goes from Invalid to Valid Clear. The cache remains in the Valid Clear state unless and until it is forced back to Invalid. This occurs in one of the following events:

- An external invalidate
- Another cache instruction
- Modification of the CCC register settings (see Section 4.3.10, “Configuration and Cache Control (CCC) Register (16)” on page 4-20)

The V bit of each cache line indicates the state. V = 0 is Invalid and V = 1 is Valid Clean. Figure 6.1 shows the state diagram for Icache and WriteThrough Dcache.

Figure 6.1
Cache State Diagram—Icache and WriteThrough Dcache



6.2.2 WriteBack Dcache

When the Dcache operates in WriteBack mode, three cache line states are required: Invalid, Valid Clean, and Valid Dirty. Figure 6.2 shows the state diagram for WriteBack Dcache. The V bit and WB bit of each line indicate the state, as shown in Table 6.1.

Figure 6.2
Cache State Diagram—Dcache WriteBack

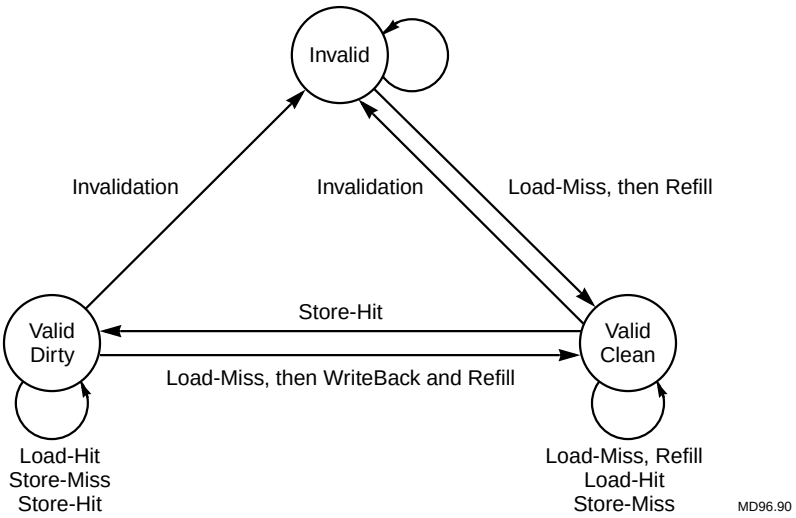


Table 6.1
Dcache WriteBack Mode

State	V Bit	WB Bit	Condition
Invalid	0	X(0)	The cache line does not contain valid information.
Valid Clean	1	0	The cache line includes valid information consistent with memory.
Valid Dirty	1	1	The cache line includes valid information, but it is not consistent with memory.

A store operation is considered to be a Dcache hit when the Tag is coincident with the physical address and the V bit is set. Of course, the physical address must be in a cached area.

When a Store-Miss occurs, the state condition of the cache line is not changed, and the store data is not written into Dcache. Instead, the store data is written to the four-word-deep write buffers, which pass it to the system's main memory.

Some lines, known as Dirty lines, contain more recent information than the main memory. Occasionally you may need to force the writing of dirty lines to main memory. You can do this using the WriteBack Cache instruction.

In WriteBack mode, data stored in the Dcache may not be passed on to the External Write Controller immediately. Because of this, the WriteBack cache instruction writes back each line of both sets in a two-way set associative configuration. The instruction does not check whether the address specified by the instruction would hit or miss at the cache line to which it pages. If the WB bit is set, the line data is written back and causes several stall cycles to read data from the Dcache. The actual number of stall cycles depends on the speed of memory access.

Cache lines can be invalidated by an external bus master. A cache line is invalidated when the Invalidate Address matches the Cache Tag ID, and the Cache Invalidation signal(s) are asserted.

6.3 Address and Cache Tag

Figure 6.3 illustrates the relationship between instruction and data address and cache memory location, for both direct map and two-way set associative cache configurations. The Word Offset field addresses a word in a line. The Line Number field addresses a line in the cache memory. The Cache Tag ID field serves as the tag for the address line.

If the system has a MMU, the cache access is indexed by the virtual address and tagged by the physical address. Because the minimum memory page size is 4 Kbyte, there is no virtual/physical address issue if the cache set size is 4 Kbyte or less. If the cache set size is 8 Kbyte and the page size is 4 Kbyte, address bit 12 of the virtual and physical address must be coincident.

Figure 6.3
Address to Cache Tag
and Line Number

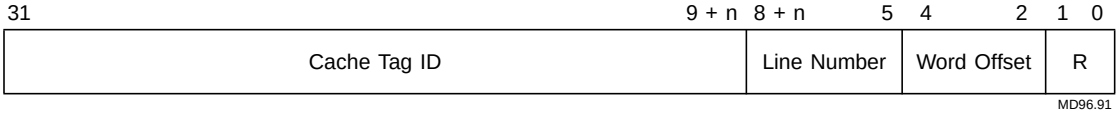


Table 6.2 shows how the value of n sets different cache sizes.

Table 6.2
Setting Cache Size

Setting Cache Size	Value of n
1 Kbyte	1
2 Kbyte	2
4 Kbyte	3
8 Kbyte	4

6.4 Dcache Scratch Pad RAM Mode

The CW4010 Dcache Set-0 or Set-1 can be configured as a Scratch Pad RAM. You can do this by setting the SR0 bit of the CCC register for Set-0 and the SR1 bit for Set-1. When SR0 is set to 1, Set-0 is enabled as a Scratch Pad RAM. When the SR1 bit is set to 1, Set-1 is enabled as a Scratch Pad RAM.

The Scratch Pad RAM must be located in one specific physical address space like a local data memory. If the CW4010 ASIC device has Dcache Tag RAM, tag must be programmed by isolating the cache before setting SR bit(s). You do this by programming the CCC register as follows: IsC = 1, TAG = 1, InV = 0; and by setting either DE0 or DE1.

If the Dcache RAM is used as only a Scratch Pad RAM in the ASIC device, the Dcache Tag RAM can be removed physically from the device. In this case, the Dcache Tag inputs of the CW4010 core must be set HIGH or LOW according to the address of the Scratch Pad RAM area. The SR bit(s) should all be set to 1.

When the Dcache Scratch Pad RAM is enabled, an access to the Scratch Pad RAM area is a local memory access without any stall cycle.

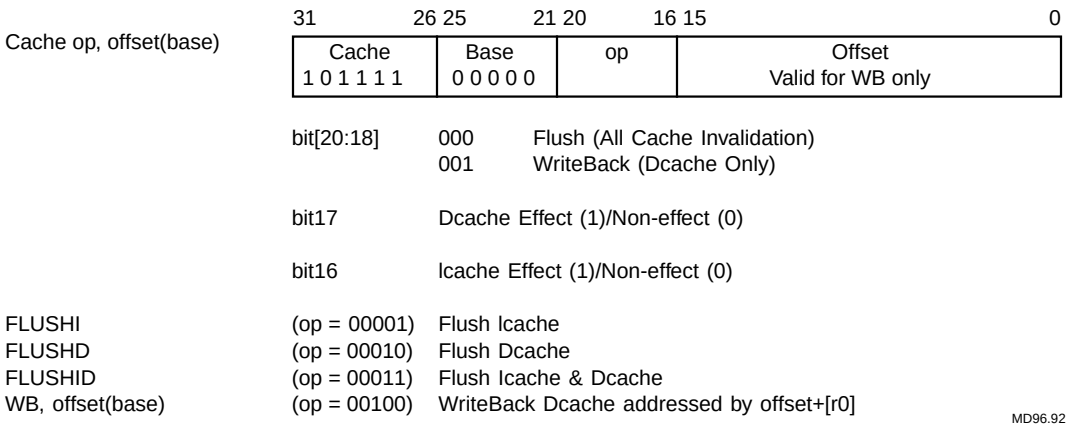
6.5 External Invalidation

Icache and Dcache lines can be invalidated by external hardware for bus snooping. The CW4010 has an invalidate strobe and invalidate address bus input. WriteBack by external hardware is not supported. Details are described in Chapter 7, “Signals.”

6.6 Cache Instructions

The CW4010 has two types of cache instructions for initialization and WriteBack. The cache instruction must be followed by three No Operation (NOP) instructions. Figure 6.4 shows the cache instruction format.

Figure 6.4
Cache Instruction Format



6.6.1 Flush (All Cache Invalidation)

One execution of a cache instruction can invalidate all lines of the Dcache or the Icache, or of both. Bit 17 of the instruction defines effect and non-effect for the Dcache, and Bit 16 defines effect and non-effect for the Icache. If both of bits are 0, this is a No Operation (NOP), and the base register and the offset have no meaning.

One cache line of one or more cache sets is invalidated during one clock cycle. Invalidation starts from the WB stage of the execution pipeline, and the pipeline stall request signal is asserted during the time that the cache lines are invalidated. If the pipeline cancel signal is asserted, the invalidation is not executed. The number of the invalidation clock cycles

is always 256, regardless of the cache size actually implemented. During this time, the CPU does not respond to interrupts.

6.6.2 WriteBack

WriteBack is effective for the Dcache only, so bits 17 and 16 are ignored. Bits [12:5] of the effective address, which is `offset+GPR[base]`, specify the Dcache line. Cache size is also a factor. For example, if the cache size is 1 Kbyte direct mapped or 2 Kbyte two-way set associative, only bits [9:5] are used. Upper bits of the effective address are ignored. Note that the Tag is not checked.

One WriteBack instruction writes back both lines of the two-way set associative cache if the WB bit is set. If WB is cleared, there is no operation. WB is executed at the WB stage and causes four stall cycles to read data from a dirty line. WB bits are cleared after the cache lines are written back.

6.6.3 Cache Maintenance by CCC Register

Certain CCC register bits support Dcache and Icache maintenance and testing. Table 6.3 lists the bits of the CCC register related to the cache.

Table 6.3
CCC Bits Related to Cache Configuration

Bit(s)	Function
IE0	Icache Set-0 Enable(1)/Disable(0)
IE1	Icache Set-1 Enable(1)/Disable(0)
IS[1:0]	Icache Size (00:1 Kbyte, 01:2 Kbytes, 10:4 Kbytes, 11:8 Kbytes)
DE0	Dcache Set-0 Enable(1)/Disable(0)
DE1	Dcache Set-1 Enable(1)/Disable(0)
DS[1:0]	Dcache Size (00:1 Kbyte, 01:2 Kbytes, 10:4 Kbytes, 11:8 Kbytes)
WB	Dcache WriteBack (1)/WriteThrough (0) if TEbit = 1
SR0	Dcache Set-0 Scratch Pad RAM Enable(1)/Disable(0)
SR1	Dcache Set-1 Scratch Pad RAM Enable(1)/Disable(0)
IsC	Dcache/Icache Isolate Cache Mode Enable(1)/Disable(0)
TAG	Dcache/Icache Tag Test Mode Enable (1)/Disable (0)
INV	Dcache/Icache Invalidate Mode Enable(1)/Disable(0)

The CW4010 has three maintenance modes that allow you to maintain and test the internal Icache and Dcache. The three modes are Data Test, Tag Test, and Invalidate. Before entering any of these modes, the

processor must be executing in kseg1 (non-cacheable address space0), interrupts must be disabled, and the caches must be isolated (IsCbit = 1). When the caches are isolated, load and store instructions access the Icache and Dcache. The system's external main memory is not affected by these load and store accesses.

To enable the cache maintenance mode, use the following procedure:

1. Set the appropriate bits in the CCC register with IsCbit = 1. You can do this by executing the MTC0 instruction, which has one delay slot. The three instructions immediately following the MTC0 instruction should not be load or store instructions.

   The IE0, IE1, DE0, and DE1 bits in the CCC register select the cache set that is to be accessed, as shown in Table 6.4. Only one cache set should be enabled when performing a load operation. Multiple caches may be enabled when performing a store operation.

Table 6.4
TAG and INV Encoding

Bit Set	Bit Number	Cache Set Accessed
IE0	17	Icache Set-0
IE1	16	Icache Set-1
DE0	13	Dcache Set-0
DE1	12	Dcache Set-1

   The TAG and INV bits in the CCC register select the Cache Maintenance function. Table 6.5 shows the encoding for the two bits.

Table 6.5
TAG and INV Encoding

TAG Bit 1	INV Bit 0	Cache Maintenance Mode
0	0	Data Test
1	0	Tag Test
x	1	Invalidate

2. Clear the IE bit in the Status register to disable all interrupts. This operation is usually done automatically because Cache Maintenance operations are done in an exception handler (most commonly the reset handler).

   – Data Test Mode

     In this mode, all loads and stores access the data RAMs selected by IE0, IE1, DE0, and DE1 bits. Effective lower address

bits specify the cache address. The precise bit field depends on the cache size and configuration actually implemented.

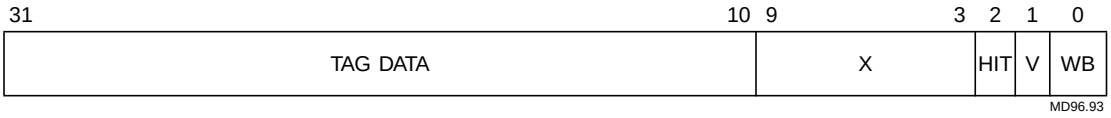
– Tag Test Mode

When TAG bit is set to 1, the CW4010 is in Tag Test Mode. Load and store operations access the Tag RAMs. The tag bits available for testing in the Tag Test Mode are the Tag Data, Hit, WriteBack (Dcache only), and Valid bits. Note that the WriteBack bit is present only in Dcache. The Hit bit is ignored during a store operation. For a load operation, the Hit bit is set if a match occurs.

The Cache Tag ID bits are written from or compared to the most significant bits of the effective address (`offset + GPR[base]`).

A load operation from the Tag RAM returns the information shown in Figure 6.5. Bits [31:10] are the Tag Data; bit 2 is the Hit bit; bit 1 is the Validate bit which reflects the setting of the INV bit in the CCC register; bit 0 is the WriteBack bit which reflects the setting of the WB bit in the CCC register. You can ignore bits [9:3].

Figure 6.5
Tag Test Mode Loaded Data Format



– Invalidate Mode

When the INV bit in the CCC register is set to 1, the CW4010 is in Invalidate mode. Because the caches contain random data on both warm and cold starts, software must invalidate all lines in the ICache and Dcache. Executing store word instructions invalidates the addressed cache line in the enabled cache(s). After reset, 0 must be written into all Tags for both sets of Dcache and Icache. Cache Flush instructions can perform the same function.

Chapter 7
Signals

This chapter describes the CW4010 core I/O signals. You will find this chapter useful if you are interfacing the CW4010 with other core logic or external logic. This chapter contains the following sections:

7.1 Signal Conventions

The following signal conventions are used in this chapter:

- Signals that are inputs have a lowercase i as part of the signal name, for example SCD**i**p. Signals that are outputs have a lower case o, for example, SCD**o**p. Lowercase characters are used to avoid confusion between uppercase alpha character I and the number 1, and uppercase alpha character O and the number 0.
- Active-LOW signals have a lowercase n at the end of the signal name, for example RESET**n**. Active-HIGH signals have a lowercase p at the end of the signal name, for example SCAo**p**.
- The term assert means to drive a signal true or active. The term deassert means to drive a signal false or inactive.
- You can use the CW4010 core in a variety of designs and with a variety of peripheral logic. For this reason, it is not always possible to identify the agent that asserts and deasserts the I/O signals. The signal descriptions in this manual indicate the states to which the core’s I/O signals must be driven. You may then select the design components needed to meet the signal requirements of the core.
- All interface signals are input to or output from the CW4010 core.

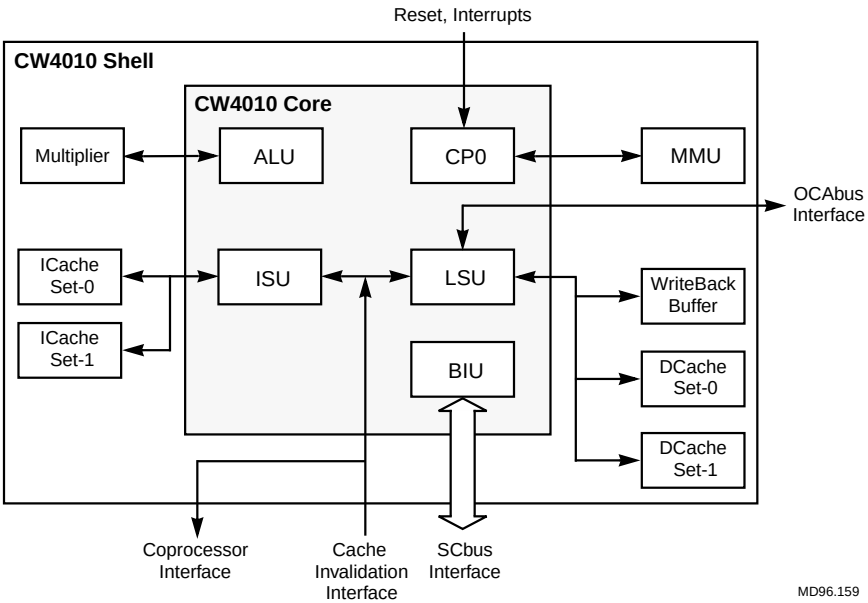
7.2 Signal Synchronization

All input signals must be synchronized to the rising edge of the system clock outside the CW4010. Asynchronous signals, such as resets or interrupts, must be synchronized by at least two sequential flipflops. All output signals are synchronized to the rising edge of the system clock inside the CW4010.

7.3 CW4010 Modularity

The CW4010 has two module levels, CW4010 Core and CW4010 Shell, as shown in Figure 7.1.

Figure 7.1
CW4010 Module



The CW4010 Core module is a process-independent encrypted synthesizable HDL (High-level Design Language) model. It includes the following internal base units:

- CP0 (System Coprocessor, Coprocessor-0)
- ALU (Arithmetic and Logic Unit)
- ISU (Instruction Scheduler Unit)
- LSU (Load Store Unit)
- BIU (Bus Interface Unit)

The CW4010 Shell is a decrypted HDL model that includes some process dependent models and provides the following options:

- Icache configuration (direct map or two-way set associative) and Icache size (0, 1, 2, 4, and 8 Kbyte for each set)
- Dcache configuration (direct map or two-way set associative) and Dcache size (0, 1, 2, 4, and 8 Kbyte for each set)
- Optional WriteBack Buffer, which can be removed for WriteThrough Dcache configuration
- Optional TLB for MMU (TLB or dummy TLB)
- High/Low performance Multiplier or Non-Multiplier configuration with optional Multiply/Addition function

The CW4010 Shell HDL model is open. You can change modules individually in the shell to design different configurations. Internal models of the CW4010 Core are not changed by these configuration changes. The optional modules are provided by LSI Logic Corporation.

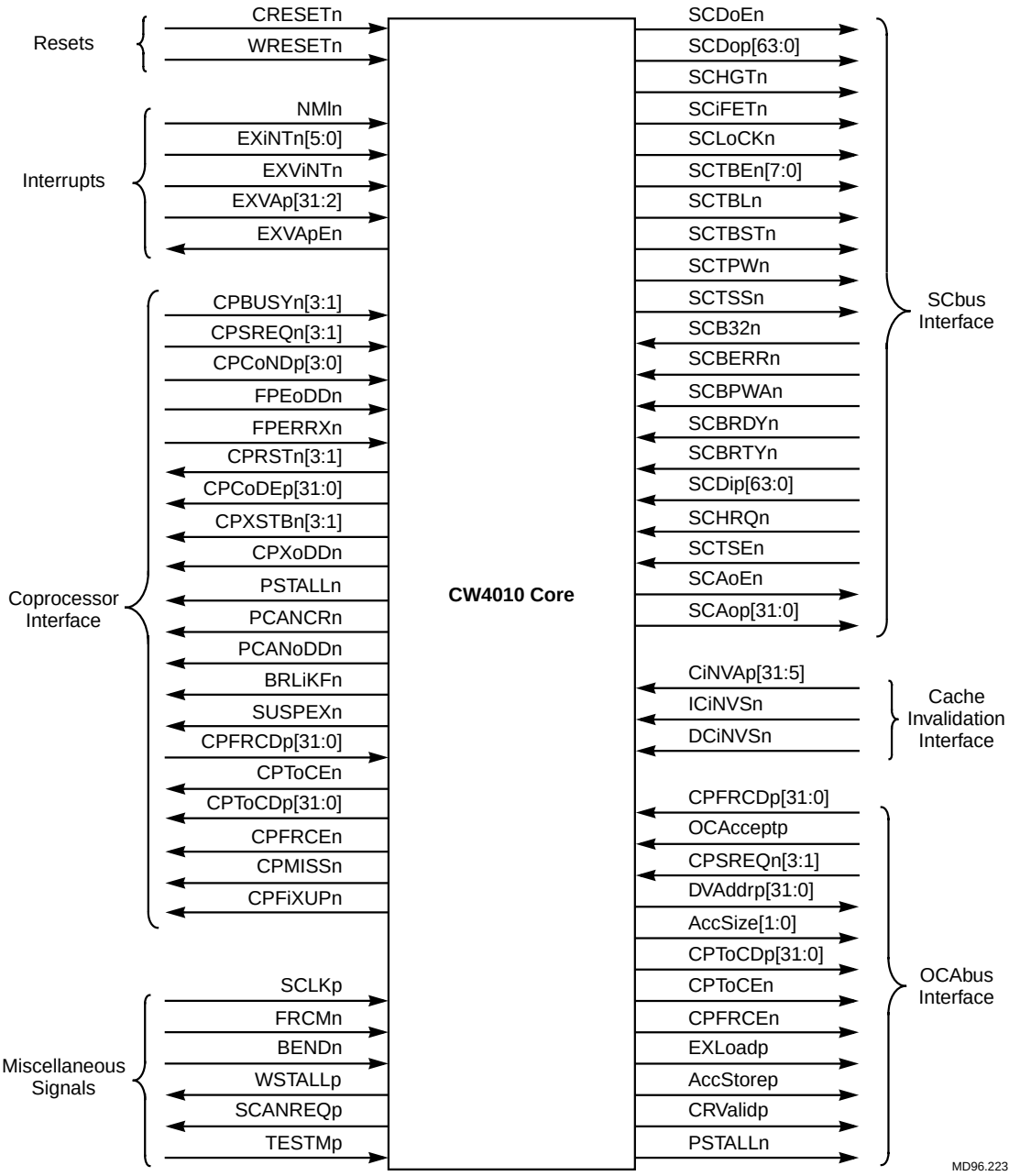
7.4 CW4010 Shell Interface Signal Definitions

The CW4010 Shell interface signals are divided into seven categories, as follows.

1. Reset signals, which interface to the CP0 unit (see Section 7.4.1, “Reset Signals,” on page 7-5)
2. Interrupt signals, which interface to the CP0 (see Section 7.4.2, “Interrupt Signals,” on page 7-5
3. SCbus Interface signals, which interface to the BIU (see Section 7.4.3, “SCbus Interface Signals,” on page 7-6)
4. Cache Invalidation Interface signals, which interface to the ISU and LSU (see Section 7.4.4, “Cache Invalidation Interface Signals,” on page 7-10)
5. Coprocessor Interface signals, which interface to the ISU and LSU (see Section 7.4.5, “Coprocessor Interface Signals,” on page 7-11)
6. OCAbus Interface signals, which interface to the LSU (see Section 7.4.6, “OCAbus Interface Signals,” on page 7-16)
7. Miscellaneous signals, such as system clock input and endian input (see Section 7.4.7, “Miscellaneous Signals,” on page 7-20)

Figure 7.2 shows the CW4010 core’s interface signals arranged in functional groups.

Figure 7.2
CW4010 Interface
Signals



7.4.1 Reset Signals

This section describes the Reset signals which interface to the CP0.

CRESETn **Cold System Reset** **Input**

The CW4010 is reset asynchronously when CRESETn is asserted. CRESETn must be deasserted synchronously on the rising edge of SCLKp. All internal states are initialized by asserting CRESETn. Of all exception inputs, CRESETn has the highest priority. When it is deasserted, the CP0 generates a cold reset exception (0xBFC00000).

WRESETn **Warm System Reset** **Input**

WRESETn must be asserted and deasserted synchronously on the rising edge of the SCLKp. While it is asserted, internal states are initialized. When it is deasserted, the CP0 generates a warm reset exception (0xBFC00000).

7.4.2 Interrupt Signals

This section describes the Interrupt signals which interface to the CP0.

EXVAp[31:2] **External Vectored Interrupt Address Input** **Input**

This is the interrupt vector address. It is accepted by the CW4010 when EXVApEn is asserted and is written directly into the program counter. EXVAp[31:2] must remain stable until EXVApEn is asserted. A virtual address for the exception handler can be provided directly.

EXVApEn **EXVAp Enable** **Output**

This is the enable signal for the interrupt vector address, EXVAp[31:2]. The CW4010 asserts EXVApEn to enable the address.

EXiNTn[5:0] **External Interrupts** **Input**

External logic asserts the EXiNTn[5:0] signals to cause the CP0 in the CW4010 to generate an interrupt exception. Assertion of these inputs is indicated in the IP[7:0] field of the Cause register. Consequently, the interrupting logic should continue to assert the external interrupt input until the exception routine has serviced the interrupt.

The interrupt inputs can be individually disabled or masked by setting the appropriate bit in the Status register.

External interrupts are not recognized if the interrupt enable bit in the Status Register is cleared. However, the inputs conditions are shown in the IP bits of the Status register.

EXViNTn **External Vectored Interrupt Input** **Input**
EXViNTn is an external interrupt input that is driven by an external interrupt controller. This feature was not present in previous LSI Logic cores. Refer to Section 4.4.22, “External Vectored Interrupt Exception” on page 4-48 for further information.

NMin **Non Maskable Interrupt** **Input**
NMin is a non-maskable interrupt. When the CW4010 detects that NMin is asserted, the CP0 generates a non-maskable interrupt exception (0xBFC00000).

7.4.3 SCbus Interface Signals

This section describes the SCbus interface signals, which interface to the BIU.

SCAoEn **Address Output Enable** **Output**
When asserted, SCAoEn indicates that the address output bus SCAop[31:0] lines are valid. The CW4010 asserts the signal when the BIU is performing an SCbus transaction, and the signal remains active throughout the operation. SCAoEn also enables SCTBSTn, SCTBEn, and SCTPWn.

SCAop[31:0] **Address Output Bus** **Output**
SCAop[31:0] is the address output bus for instruction fetch and data read/write operations. The SCAop[31:0] bus is valid only when the address output enable signal SCAoEn is asserted. It remains valid throughout the operation until SCBRDYn, SCBRTYn, or SCBERRn is asserted.

SCB32n **32-bit Bus Width Sizing** **Input**
SCB32n indicates that the external bus slave on the SCbus needs 32-bit bus sizing. The CW4010 samples this signal on the rising edge of the clock that synchronizes SCBRDYn. If the signal is asserted for a 64-bit transaction, which is a doubleword or a part of a burst transaction, the BIU generates a subsequent 32-bit word transaction and packs data to 64 bits for a read

transaction or unpacks data to 32 bits for a write transaction.

SCBERRn **Bus Error** **Input**

SCBERRn is asserted to terminate the current transaction when a bus error occurs. If SCBRDYn or SCBRTYn is asserted at the same time as SCBERRn, SCBERRn has higher priority. SCBERRn is reported to the CP0 and the CP0 generates an exception.

SCBPWAn **Bus In-Page Write Accept** **Input**

SCBPWAn indicates that the external bus slave on the SCbus accepts In-Page write transactions. External logic asserts the signal and the CW4010 samples it on the rising edge of the clock that synchronizes SCBRDYn. If the SCTPWn signal is not asserted, asserting or deasserting SCBPWAn has no significance.

SCBRDYn **Bus Ready** **Input**

The system asserts SCBRDYn when the current transaction is terminated. When asserted, it indicates that the SCbus is available. The signal remains active (LOW) until the next transaction starts. The system then deasserts the signal to indicate that the SCbus is not available.

SCBRTYn **Bus Retry** **Input**

SCBRTYn is asserted when the current transaction is terminated unsuccessfully and must be retried later. The control state goes back once to the idle state, then all bus requests are arbitrated again. If there are no other higher priority requests and SCTSEn is asserted, there is one idle state between the first transaction and a retry transaction. If SCBRDYn and SCBRTYn are asserted at the same time, SCBRTYn has the higher priority.

SCDip[63:0] **Data Input Bus** **Input**

SCDip[63:0] are data bus input signals for instruction fetch and data read transactions. The CW4010 samples SCDip[63:0] on the rising edge of the clock when SCBRDYn is asserted. Byte ordering is little endian. If you are designing a big endian system, the higher order bits, SCDip[31:0], must be swapped with the lower order bits, SCDiP[63:32], outside the CW4010 shell.

SCDoEn **Data Output Enable** **Output**

SCDoEn indicates that the data output signals SCDop[63:0] are valid. the CW4010 asserts the signal throughout the write transaction to indicate that the current transaction is a write transaction and to enable data output.

SCDop[63:0] **Data Output Bus** **Output**

SCDop[63:0] are the data output bus signals for data write operations and for data WriteBack to the Dcache. The signals are valid throughout the write transaction. Byte ordering is little endian. If you are designing a big endian system, the higher order bits, SCDop[31:0], must be swapped with the lower order bits, SCDoP[63:32], outside the CW4010 Shell.

SCHGTn **Bus Hold Grant** **Output**

The BIU enters the hold state and asserts the grant signal SCHGTn to indicate that it is releasing SCbus ownership in response to a bus hold request, SCHRQn.

SCHRQn **Bus Hold Request** **Input**

SCHRQn indicates that an external bus master is requesting ownership of the SCbus. The bus hold request has the highest priority during bus arbitration. A bus hold request cannot break continuous transactions of In-Page writes and burst read/write transactions if those transactions are supported by an asserted SCTSEn signal, but must wait until SCTSEn is deasserted.

SCiFETn **Instruction Fetch** **Output**

SCiFETn indicates that the BIU is fetching an instruction for monitoring purposes. The CW4010 drives it LOW at this time and outputs it to external logic.

SCLoCKn **Bus Lock** **Output**

When the CW4010 asserts SCLoCKn, it indicates that the processor wishes to lock the SCbus to restrict bus ownership. The CW4010 asserts the signal when a read transaction is started by executing a LoadLink instruction in an uncached area or a WriteThrough cached area. It deasserts the signal just before a write transaction is started by executing a StoreConditional instruction. During the read and write transactions, it asserts the signal continuously, preventing bus ownership from

changing during one of these transactions. If a StoreConditional transaction hits the Dcache in a WriteBack cached area while SCLoCKn is asserted, an incorrect condition exists, and the CW4010 deasserts SCLoCKn without completing any bus transactions.

SCTBEn[7:0] **Byte Enables** **Output**
SCTBEn[7:0] indicates which byte positions are valid for a transaction. The CW4010 asserts only one of the signals for a byte read or a byte write transaction. It asserts all the signals for a doubleword or a burst transaction. The SCTBEn[7:0] signals are valid when the CW4010 asserts SCAoEn.

SCTBLn **Burst Last Doubleword** **Output**
The CW4010 deasserts SCTBLn while the first, second, and third doubleword of a burst transaction is being read or written. Otherwise, it asserts the signal, which is valid on the rising edge of the system clock.

SCTBSTn **Burst Transaction** **Output**
When the CW4010 asserts SCTBSTn, it indicates that a transaction is taking place during which four doublewords will be moved, and that currently the first doubleword is being moved. It deasserts the signal after the first word has been transferred and during singleword transactions.

SCTPWn **Next Transaction is In-Page Write** **Output**
When the CW4010 asserts SCTPWn, it indicates that the next transaction is in the same DRAM page, as defined in the Configuration register. When the CW4010 asserts this signal, a maximum of four write transactions take place one after the other, even if there is an instruction fetch request or data read request. If four write transactions are performed continuously, the CW4010 asserts SCTPWn from the first through the third transaction and deasserts it for the last (fourth) transaction. The CW4010 asserts SCTPWn from the beginning of one In-Page write transaction to the end of that transaction. The write buffer in the LSU checks to see if the subsequent write request is in the same page.

SCTSEn **Transaction Start Enable** **Input**
SCTSEn enables or disables a new SCbus transaction. Transaction requests are arbitrated only when SCTSEn is

asserted. The signal must be deasserted and then asserted when SCBRDYn is asserted to allow an idle cycle between two transactions. During the time SCTSEn is deasserted, the BIU repeats the idle state.

SCTSSn **Transaction Start Strobe** **Output**
When the CW4010 asserts SCTSSn, it indicates that a transaction has started. The CW4010 asserts the signal for one clock cycle at the beginning of a transaction. If the transaction lasts through one cycle and the next transaction begins immediately, it asserts SCTSSn continuously.

7.4.4 Cache Invalidation Interface Signals

This section describes the cache invalidation interface signals, which interface to the ISU, LSU, and CP0.

CiNVAp[31:5] **Cache Invalidation Address Bus** **Input**
The CiNVAp[31:5] input bus is the address input bus for Dcache and Icache invalidation. When an external bus master writes data into the main memory, the address must be checked in the Dcache and the Icache. If the address is cached, the line must be invalidated. The CW4010 samples the bus when DCiNVSn or ICiNVSn is asserted.

DCiNVSn **Dcache Invalidation Strobe** **Input**
When asserted, DCiNVSn indicates the Cache Invalidation Address Bus is valid and that there is need for a snooping sequence. If the cache tag is not coincident with higher address bits, the line is not invalidated.

ICiNVSn **Icache Invalidation Strobe** **Input**
When asserted, ICiNVSn indicates the Cache Invalidation Address Bus is valid and that there is need for a snooping sequence. If the cache tag is not coincident with higher address bits, the line is not invalidated.

7.4.5 Coprocessor Interface Signals

This section describes the coprocessor interface signals, which interface with the ISU and LSU.

BRLiKFn **BranchLikely of Even Slot is False** **Output**

The CW4010 asserts BRLiKFn when the BranchLikely instruction is in an even slot and it is false. If, at this time, a coprocessor has a valid instruction in the EX stage, the instruction must be cancelled. It is not necessary to check whether the instruction in the EX stage is in an even or odd slot, since the CW4010 asserts BRLiKFn only when the BranchLikely instruction is in the even slot. If the BranchLikely instruction in the even slot is not taken, the instruction in the odd slot must be nullified although it has already been started.

This signal is valid at the EX stage of the pipeline.

CPBUSYn[3:1]

Coprocessor Busy **Input**

These inputs are asserted when the external coprocessors are busy and cannot accept any coprocessor operations. CPBUSYn3 is associated with the CP3, CPBUSYn2 with CP2, and CPBUSYn1 with CP1. The ISU does not assert the execution strobe, CPXSTBn[3:1], when the related CPBUSYn signal is asserted, and the CW4010 stalls until the busy signal is deasserted. Each coprocessor is independent and asserts its busy signal from the EX stage. The CW4010 examines the CPBUSYn signal at the RD stage of the pipeline, on the rising edge of the clock.

This signal is sampled at the RD stage of the pipeline.

CPCoDEp[31:0]

CP Instruction Code Bus **Output**

This bus outputs the entire instruction bit field at the RD stage. It is valid when the CW4010 asserts one of the CPXSTBn[3:1] lines. Although the CW4010 can execute two instructions per cycle, only one coprocessor instruction can be issued in one cycle. CPCoDEp[31:0] are the selected outputs of the even and odd instruction slot. External logic must sample the bus on the rising edge of the system clock when the CW4010 asserts strobe CPXSTBn.

It is not necessary to decode all bits of an instruction because the execution strobe signal is a partial decoding signal.

CPCoDEp[31:0] are valid at the RD stage of the pipeline.

CPCoNDp[3:0]

Coprocessor Condition **Input**

These inputs are used for the coprocessor condition branch instruction. The CW4010 samples the inputs in the ISU at the EX stage of a conditional branch instruction. The four inputs are associated with the four possible coprocessors (CP0-CP3). CPCoNDp0 is for CP0. However, CP0 does not need it, and CPCoNDp0 is therefore used as a general purpose condition input.

These signals are valid at the EX stage of the pipeline.

CPFiXUPn **Data Fixup Cycle Strobe for LWCz Cache Miss** **Output**

The CW4010 asserts CPFiXUPn when correct data is output on CPToCDp[31:0] during a fixup cycle. It asserts the signal during stall cycles because LWCz cache misses cause the pipeline to stall until the data is read.

CPFRCDp[31:0]

Data from Coprocessor **Input**

This bus inputs data from a coprocessor register to a CW4010 CPU general purpose register or to memory. Data on the bus is valid when CW4010 asserts the data enable signal, CPFRCEn. If there are several external coprocessors, the data bus must be multiplexed outside the CW4010 shell.

This signal is sampled at the CR stage of the pipeline.

CPFRCEn **Data from Coprocessor Enable** **Output**

The CW4010 asserts this signal to enable the data input bus CPFRCDp[31:0]. Coprocessors can generate the same information from the instruction code, CPCoDEp, by tracking the pipeline stage. External logic must decode the coprocessor number from CPCoDEp[31:0]. If the pipeline enters a stall condition when there is a coprocessor data movement instruction in the CR stage, the CW4010 asserts CPToCEn continuously until the stall condition is resolved.

This signal is valid at the CR stage of the pipeline.

CPMiSSn **Data Cache Miss Strobe for LWCz** **Output**

The CW4010 asserts CPMiSSn at the CR stage of an LWCz instruction when a Dcache miss occurs. Data at the CR stage is not correct and the correct data is put on CPToCDp[31:0] during a later fixup cycle. The CW4010 asserts PSTALLn from the WB stage of the LWCz instruction.

This signal is valid at the CR stage of the pipeline.

CPRSTn[3:1] **Coprocessor Reset** **Output**

These outputs indicate the condition of CU bit [3:1] in CP0's status register. If the CU bit is 0, the CW4010 asserts the corresponding CPRSTn[3:1] output. The CW4010 asserts the CPRSTn[3:1] signals when a cold reset is asserted. At this time, the CU bits are cleared. The CU bits are not cleared when a warm reset is asserted. The CPRSTn[3:1] outputs allow the system designer to use software resets for external coprocessors.

CPSREQn[3:1]

Coprocessor Stall Request **Input**

The external coprocessors assert these signals when they need to request a pipeline stall. Coprocessors can assert CPSREQn[3:1] while a previous coprocessor instruction is being executed, after decoding a coprocessor instruction, and after the RD stage. The CW4010 asserts PSTALLn immediately when in response to one of these signals.

CPToCDp[31:0]

Data to Coprocessor **Output**

This bus outputs data to a coprocessor register from a CW4010 CPU general purpose register or from memory. Data on the bus is valid when the CW4010 asserts the data enable signal, CPToCEn.

These signals are valid at the CR stage of the pipeline.

CPToCEn **Data to Coprocessor Enable** **Output**

The CW4010 asserts this signal to indicate when the data output bus, CPToCDp[31:0], is valid. Coprocessors can generate the same information from the instruction code, CPCoDEp, by tracking the pipeline stage. The coprocessor number must be decoded from

CPCoDEp[31:0]. If the pipeline enters a stall condition when there is a coprocessor data movement instruction in the CR stage, the CW4010 asserts CPToCEn continuously until the stall condition is resolved.

This signal is valid at the CR stage of the pipeline.

CPXoDDn **Coprocessor Instruction at Odd Slot** **Output**

When the CW4010 asserts an execution strobe, it also asserts CPXoDDn to indicate that the coprocessor instruction is in the odd slot. This information must be kept in the coprocessor pipeline until the CR stage. It is used to determine whether or not the instruction should be cancelled when the cancellation signal is asserted.

This signal is valid at the RD stage of the pipeline.

CPXSTBn[3:1]

Coprocessor Instruction Execution Strobe **Output**

These strobe signals indicate the start of a coprocessor operation that involves data movement. The CW4010 asserts only one of the signals during a clock cycle. CPXSTBn[3:1] are partial decoding signals for an instruction. For example, when the CW4010 asserts CPXSTBn1, CP1 turns on, and so forth. The ISU also uses the signals to check for resource conflicts, including coprocessor busy signals.

CPXSTBn[3:1] are valid at the RD stage of the pipeline.

FPEoDDn **FPU Error Exception in Odd Slot** **Input**

FPEoDDn indicates whether the instruction that caused an FPU exception (FPERRXn assertion) is in an even slot (FPEoDDn = HIGH) or odd slot (FPEoDDn = LOW) when it started at the RD stage. The CW4010 ignores the FPEoDDn signal when FPERRXn is deasserted.

When the instruction is started at an RD stage, the CPXoDDn signal informs the coprocessor that the instruction is in an even or odd slot. To handle a pipeline cancel correctly, the coprocessor must keep the instruction in its pipeline registers. To execute an FPU exception precisely, the coprocessor that asserts FPERRXn at the EX stage must drive FPEoDDn correctly according to the even/odd status of the EX pipeline stage.

If the FPERRXn exception does not need to be treated precisely, the FPEoDDn input must be strapped HIGH to cancel the pipeline, in the same way as an interrupt exception.

FPERRXn **Floating Point Unit Error Exception** **Input**

FPERRXn is an exception input, used specifically with an FPU coprocessor. The CW4010 samples the signal at any time in the EX stage and issues a pipeline cancel signal at the CR stage, in the same way as EXiNTn. In the Cause Register, exception code 15 is shown for the exception if it is the highest priority. FPERRXn can be used as a user-defined coprocessor exception input.

FPERRXn must be treated **precisely**. The FPU asserts FPERRXn at the EX stage of the instruction with the FPEoDDn signal assertion/deassertion. The CW4010 asserts the pipeline cancel signal at the CR stage with the correct even/odd cancel signal.

PCANCRn **Pipeline Cancel at CR stage** **Output**

When one or more exceptions occurs, the pipeline is cancelled at the CR stage and the CW4010 asserts PCANCRn. Coprocessor pipelines must be cancelled to prevent a second execution of the coprocessor instruction under either one of the following conditions: when the coprocessor returns from an exception handler; or when the coprocessor has finished executing an LWCz instruction that caused a TLB miss. The WB stage is not cancelled when PCANCRn is asserted.

This signal is valid at the CR stage of the pipeline.

PCANoDDn **Pipeline Cancel is for Odd Slot** **Output**

PCANoDDn is valid only when PCANCRn is asserted. The signal informs coprocessors whether the cancellation is for an odd or even slot. When the CW4010 asserts the signal, cancellation applies to the odd slot. When it is deasserts the signal, cancellation applies to both even and odd slots.

The coprocessor must track which slot it is executing in based on the CPXoDDn signal. When the CW4010 asserts both PCANCRn and PCANoDDn and the coprocessor instruction is in the odd slot, the instruction must be cancelled. When the CW4010 asserts

PCANCRn and deasserts PCANoDDn, the coprocessor instruction must be cancelled regardless of which slot it is operating in.

This signal is valid at the CR stage of the pipeline.

PSTALLn **Pipeline Stall Broadcasting Signal** **Output**
The CW4010 asserts this signal to indicate that the entire CW4010 pipeline is stalled. Coprocessor pipelines must be stalled when they are executing instructions. The CW4010 asserts PSTALLn for all pipeline stalls and for an LWCz instruction Dcache miss.

SUSPEXn **Suspend EX stage** **Output**
The CW4010 Instruction Scheduler Unit (ISU) asserts SUSPEXn to request coprocessors to suspend the instruction in the EX stage. The instruction in the EX stage must be held until the ISU deasserts SUSPEXn. Instructions in the CR and WB stages must be completed.

This signal is valid at the EX stage of the pipeline.

7.4.6 OCAbus Interface Signals

The CW4010 core has an On-Chip Access (OCA) interface that allows on-chip modules to be accessed at the CR stage of the pipeline without going through an SCbus transaction. This improves performance since it reduces traffic on the SCbus and therefore, reduces latency.

If the module that is the target of the transaction can respond in one clock cycle, there is no penalty for a read or write cycle. A read access on the SCbus has at least a four-clock penalty, and a write access is done through a four-deep write buffer. The on-chip modules can be accessed from the SCbus.

The CW4010 is the only bus master for the OCAbus. Instructions cannot be fetched through the OCAbus.

This section describes the OCAbus interface signals.

AccSize[1:0] **OCAbus Transaction Size** **Output**

These signals indicate the transaction size of an OCAbus transaction. The signals are valid when the CW4010 asserts either EXLoadp or AccStorep.

AccSize[1:0]	Transaction Size
00	One byte
01	Halfword
10	Tribyte
11	One word

These signals are valid at the EX stage of the pipeline.

AccStorep **OCAbus EX Stage Store Operation** **Output**

The CW4010 asserts this signal when a store instruction is being executed in the EX stage of the CW4010 pipeline. The CW4010 asserts the signal when DVAddrp[31:0] and AccSizep[1:0] are valid. DVAddrp[31:0] is decoded when the CW4010 asserts AccStorep. If the resulting address is for a device on the OCAbus, OCAcceptp is asserted.

This signal is valid at the EX stage of the pipeline.

CPFRCDp[31:0]

Data from Coprocessor **Input**

This bus inputs data from a coprocessor register to a CW4010 CPU general purpose register or to memory. It is valid when the CW4010 asserts the data enable signal, CPFRCEn. If there are several coprocessors, the data bus must be multiplexed.

These signals are valid at the CR stage of the pipeline.

CPFRCEn **Data from Coprocessor Enable** **Output**

The CW4010 asserts this signal to indicate when data on the input bus, CPFRCDp[31:0], is valid. Coprocessors can generate the same information from the instruction code, CPCoDEp[31:0], by tracking the pipeline stage, and decode the coprocessor number from CPCoDEp[31:0]. If the pipeline enters a stall condition when there is a coprocessor data movement instruction in the CR stage, the CW4010 asserts CPToCEn continuously until the stall condition is resolved.

This signal is valid at the CR stage of the pipeline.

CPSREQn[3:1]

Coprocessor Stall Request **Input**

The coprocessors assert these signals inputs when they coprocessors need to request a pipeline stall. Coprocessors can assert CPSREQn[3:1] while a previous coprocessor instruction is being executed, after decoding a coprocessor instruction, and after the RD stage. The CW4010 asserts PSTALLn immediately when one of these signals is asserted.

This signals are issued at the CR stage of the pipeline.

CPToCDp[31:0]

Data to Coprocessor **Output**

This bus outputs data to a coprocessor register from a CW4010 CPU general purpose register or from memory. The bus is valid when the CW4010 asserts the data enable signal, CPToCEn.

These signals are valid at the CR stage of the pipeline.

CPToCEn **Data to Coprocessor Enable** **Output**

The CW4010 asserts this signal to indicate when the data output bus, CPToCDp[31:0], is valid. Coprocessors can generate the same information from the instruction code, CPCoDEp[31:0], by tracking the pipeline stage, decode the coprocessor number from CPCoDEp[31:0]. If the pipeline enters a stall condition when there is a coprocessor data movement instruction in the CR stage, The CW4010 asserts CPToCEn continuously until the stall condition is resolved.

This signal is valid at the CR stage of the pipeline.

CRValidp **OCAbus CR Stage Valid** **Output**

The CW4010 asserts this signal when the CR stage of a load or store instruction is valid after it has asserted EXLoadp or AccStorep. If the load or store instruction is cancelled, the CW4010 deasserts CRValidp and the load/store operation must be cancelled.

This signal is valid at the CR stage of the pipeline.

DVAddrp[31:0]

OCAbus Virtual Address **Output**

This is the output bus for the OCAbus virtual address. It is used for a load or store instruction being executed at the EX stage for the OCAbus. The signal is valid when the CW4010 asserts either EXLoadp or AccStorep.

These signals are valid at the EX stage of the pipeline.

EXLoadp **OCAbus EX State Load Operation** **Output**

The CW4010 asserts this signal when a load instruction is being executed in the EX stage of the CW4010 pipeline. It asserts the signal when DVAddrp[31:0] and AccSize[1:0] are valid. DVAddrp[31:0] is decoded when the CW4010 asserts EXLoadp. If the resulting address is for a device on the OCAbus, OCAcceptp is asserted.

This signal is valid at the EX stage of the pipeline.

OCAcceptp **OCAbus Transaction Accepted** **Input**

When the OCA module can accept an OCA transaction, it asserts this signal. The signal is an output from the DVAddrp address decoder and it is asserted at the CR stage. When it is asserted for a read operation, the LSU selects CPFRCDp[31:0] as the data input. When it is asserted for a write operation, the data CPToCDp[31:0] is written into an OCA device at the CR stage. The data is therefore not written into the Dcache and an SCbus transaction is not requested.

This signal is valid at the CR stage of the pipeline.

PSTALLn **Pipeline Stall Broadcasting Signal** **Output**

The CW4010 asserts this signal to indicate that all stages of the CW4010 pipelines are stalled. Pipelines must be stalled when they are executing instructions.

This signal is valid at any stage of the pipeline.

7.4.7 Miscellaneous Signals

This section describes the miscellaneous signals used by the CW4010.

BENDn **Big Endian** **Input**

BENDn is a static input and must be tied LOW for big endian addressing and HIGH for little endian addressing. BENDn affects the byte positions for sizing and Load/Store data alignment.

For big endian mode, the upper 32 bits of SCDip must be swapped with the lower 32 bits, and the upper 32 bits of SCDop must be swapped with the lower 32 bits.

FRCMn **Force Cache Miss** **Input**

Asserting FRCMn forces a cache miss for both the Icache and Dcache. The CW4010 treats the transaction as an access to an uncached area. FRCMn is useful for debugging the system.

This is a static input. It is tied LOW for software debugging.

SCANREQp **Scan Debug Event** **Output**

The CW4010 asserts this signal to indicate that the CP0 has detected a scan debug event. Clock circuitry external to the core uses this information to determine when to drop out of a normal operating mode into scan debug mode.

SCLKp **System Clock** **Input**

SCLKp is the processor system clock input. It provides basic timing for the CW4010 core and determines the instruction cycle times. Internal core logic operates synchronously with the rising edge of SCLKp. Since the core processor operates at 80 MHz, you must supply an 80 MHz clock.

This clock input is used for all core modules.

TESTMp **Test Mode Enable** **Input**

TESTMp is used for testing. It is a static input and must be tied LOW during normal operation.

WSTALLp **Wait Interrupt Stall** **Output**

WSTALLp indicates that internal pipeline stages have entered a stall condition by executing a WAITI (Wait Interrupt) instruction. The CW4010 asserts WSTALLp when the instruction is at the WB stage of the CW4010 pipeline, and the signal remains active until the CW4010 receives an external exception (enabled external interrupt, NMIn, cold reset, or warm reset).

Chapter 8
Interface Operation

This chapter examines various CW4010 functional timing scenarios. It does not deal with all timing cases, however, it covers the main timing related to CW4010 transactions. For details of the operation of each of the signals discussed, refer to Chapter 7, “Signals.”

This chapter has the following sections:

In the timing diagrams shown in this chapter, all inputs and all outputs must be synchronized to the rising edge of the system clock. All inputs require setup and hold time and all outputs have valid delay times from the clock edge to the appearance of a valid level.

8.1 Reset and Exception Signals

The CW4010 has the following reset and exception inputs that connect to Coprocessor-0:

- Cold Reset
- Warm Reset
- Non-Maskable Interrupt
- Bus Error
- Floating Point Unit Exception
- Interrupts

The above inputs must be synchronized to the rising edge of the system clock.

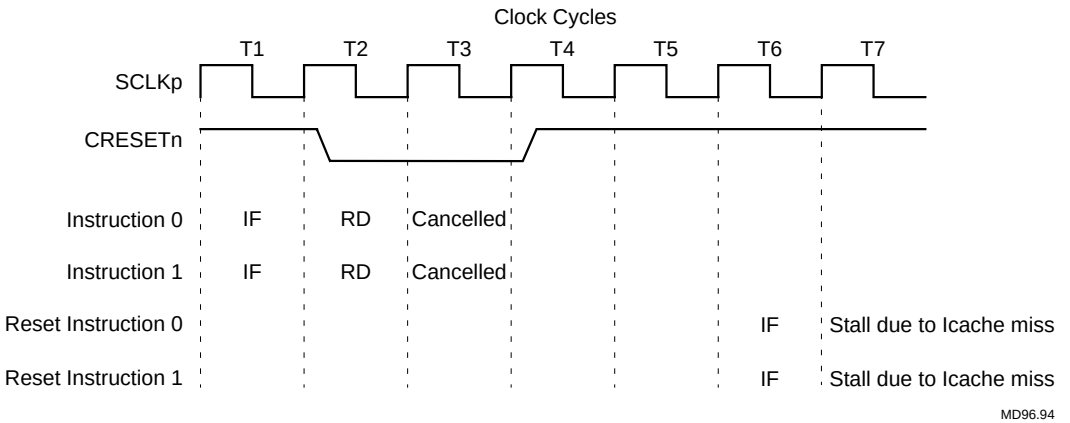
8.1.1 Cold Reset (CRESETn)

The primary purpose of a cold reset is to initialize the CW4010 core at power up.

When asserted, CRESETn initializes the internal states and control registers in the CW4010. CRESETn does not initialize general purpose registers, Icache, Dcache, or the MMU TLB.

CRESETn can be asserted asynchronously, but it must be active for at least two system clock cycles and be deasserted on the rising edge of the system clock. The CW4010 considers CRESETn a non-maskable exception and is in idle mode during the period that CRESETn is asserted. Figure 8.1 shows the timing for a Cold Reset and the start of an instruction fetch after CRESETn is deasserted.

Figure 8.1
Cold Reset and Pipeline



8.1.2 Handling Cold Resets

The CPU provides a special interrupt vector (0xBFC00000) for the CRESETn exception. The reset vector resides in unmapped and uncached CPU virtual address space, so the hardware does not need to initialize the TLB or the cache to handle the exception. The processor can fetch and execute instructions while the caches and virtual memory are in an undefined state. For further information on this subject refer to Section 4.4.5, “Cold Reset Exception” on page 4-32.

The contents of all registers in the CPU are undefined when the CRESETn exception occurs except for the following:

- In the Status register, the CU[3:0] and SR bits are cleared to zero, and the ERL and BEV bits are set to one. The other bits in the register are undefined.
- The Random register is initialized to the value of its upper bound.
- The Wired register is initialized to zero.

8.1.2.1 Servicing Cold Resets

To service the CRESETn exception, you should initialize all processor registers, coprocessor registers, caches, and the memory system. You can do this by performing diagnostic tests and by bootstrapping the operating system.

8.1.3 Warm Reset (WRESETn)

The primary purpose of the WRESETn exception is to reinitialize the processor after a fatal error.

When asserted, WRESETn initializes the CW4010 internal states and control registers. WRESETn does not initialize general purpose registers, Icache, Dcache, or the MMU TLB.

WRESETn must be asserted and deasserted on the rising edge of the system clock. It must remain active for at least two system clock cycles. WRESETn is a non-maskable exception and the CW4010 is in idle mode during the period it is asserted. The start of the instruction fetch after WRESETn is deasserted is the same as that of CRESETn, as shown in Figure 8.1.

8.1.3.1 Handling Warm Resets

The reset exception vector (0xBFC00000) is used for the WRESETn exception. The reset vector resides in unmapped and uncached CPU virtual address space, so the hardware does not need to initialize the TLB or the cache to handle the exception. The SR bit of the Status register is set to distinguish the WRESETn exception from the CRESETn exception.

Unlike a non-maskable interrupt, WRESETn resets bus state machines. Like CRESETn, it can be used on the processor in any state.

The contents of all registers are preserved when WRESETn occurs, except for the following:

- ErrorPC register, which contains the restart PC.
- ERL and BEV bits of the Status register, which are set to 1.
- SR bit of the Status register, which is set to 1.

Because WRESETn can abort cache and bus operations, cache and memory contents are undefined after the WRESETn exception occurs. For further information on this subject refer to Section 4.4.6, “Warm Reset Exception” on page 4-33.

8.1.3.2 Servicing Warm Resets

To service the WRESETn, you should save the current processor state to use for diagnostic purposes, and also to reinitialize all processor registers, the coprocessor and the memory system.

8.1.4 Non-Maskable Interrupt (NMin)

The Non-Maskable Interrupt input NMin must be asserted and deasserted on the rising edge of the system clock. When NMin is sampled and found to be active on the rising edge of the clock, the CP0 provides an non-maskable exception vector (0xBFC00000). Figure 8.2 shows the timing diagram for the fastest detected case. Figure 8.3 shows the case in which NMin is not serviced immediately because of a pipeline stall. The CW4010 detects the falling edge of NMin and latches the signal until it is ready to service it.

Figure 8.2
NMin and Pipeline (NMin is Detected Immediately)

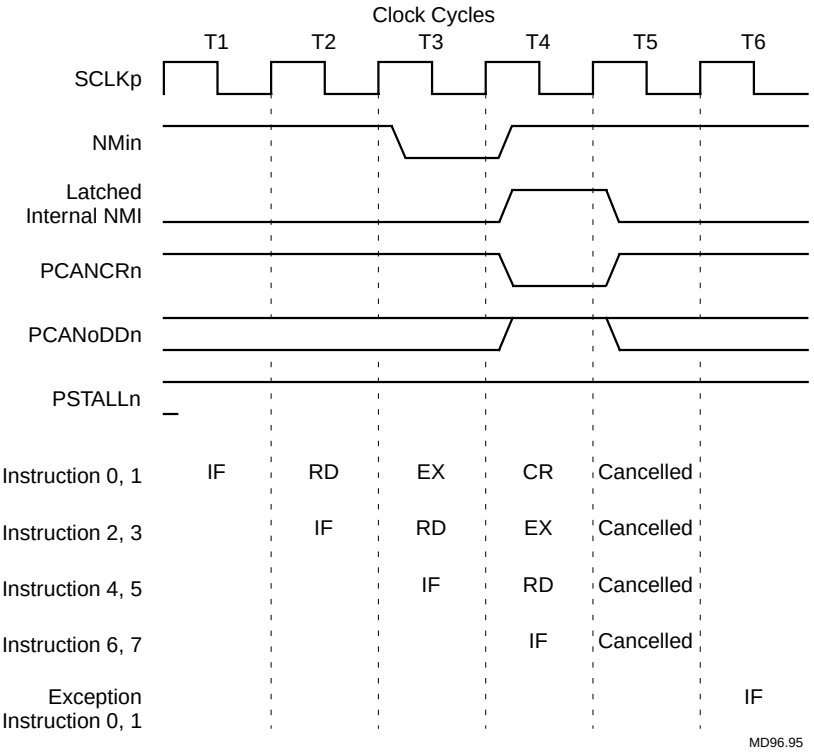
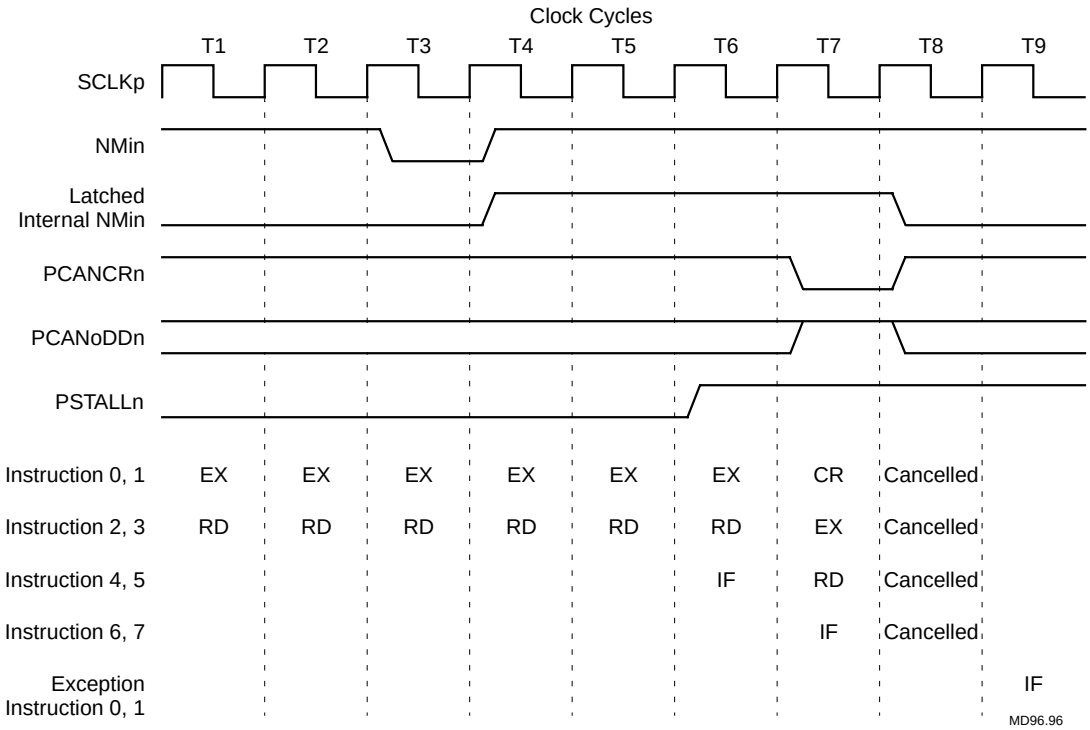


Figure 8.3
NMin and Pipeline
(NMin is not Detected
Immediately Due to
Stall)



8.1.4.1 Handling a Non-Maskable Interrupt

The reset exception vector (0xBFC00000) is also used for the NMin exception. The reset vector resides in unmapped and uncached CPU address space so that the hardware does not need to initialize the TLB or the cache to handle NMin. The SR bit of the Status register is set to differentiate this exception from a CRESETn exception.

Because an NMin could occur in the middle of another exception, program execution cannot continue after NMin has been serviced.

Unlike Cold and Warm Reset, but like other exceptions, a Non-Maskable Interrupt is taken only at instruction boundaries. The NMin exception preserves the state of the caches and memory system. For further

information on this subject refer to Section 4.4.6, “Warm Reset Exception” on page 4-33.

The contents of all registers in the CPU are preserved when this exception occurs, except for the following:

- The ErrorPC register, which contains the restart PC.
- The ERL and BEV bits of the Status register, which are set to one.
- The SR bit of the Status register, which is set to one.

8.1.4.2 Servicing a Non-Maskable Interrupt

To service the NMin exception save the current processor state for diagnostic purposes, and for reinitializing the system, including all processor registers, coprocessor registers, caches, and the memory system.

8.1.5 Bus Error (SCBERRn)

A Bus Error exception occurs when board-level circuitry detects events such as bus time-outs, bus parity errors, and invalid physical memory accesses. The SCBERRn exception is not maskable.

In the CW4010, Bus Errors are asynchronous events with respect to CPU instruction processing (much like the NMin interrupt), which means that there is no attempt to identify the instruction that was the root source of the error.

The SCBERRn input from the SCbus interface terminates a transaction and generates an exception to inform the CW4010 that an SCbus transaction has not been successfully completed. When the CW4010 is driving the SCbus, it detects the assertion of SCBERRn. SCBERRn assertion should be a synchronous one clock cycle strobe, which is latched in CW4010 until it is serviced. Figure 8.4 shows the timing diagram in which SCBERRn is serviced immediately and Figure 8.5 shows how the exception is serviced later because of stall cycles.

Figure 8.4
Bus Error and
Pipeline (Detected
Immediately)

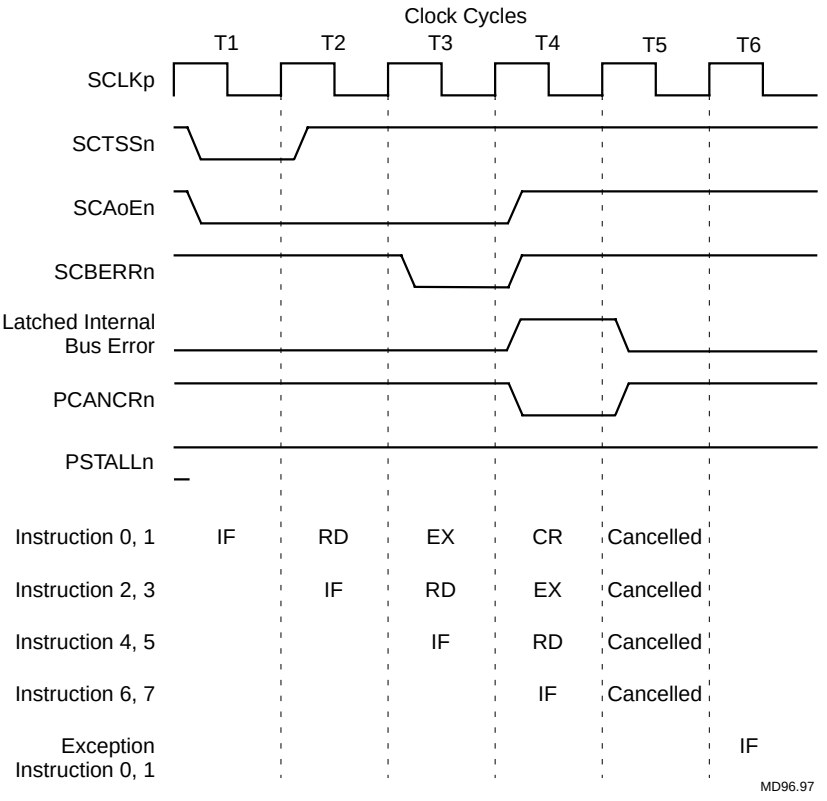
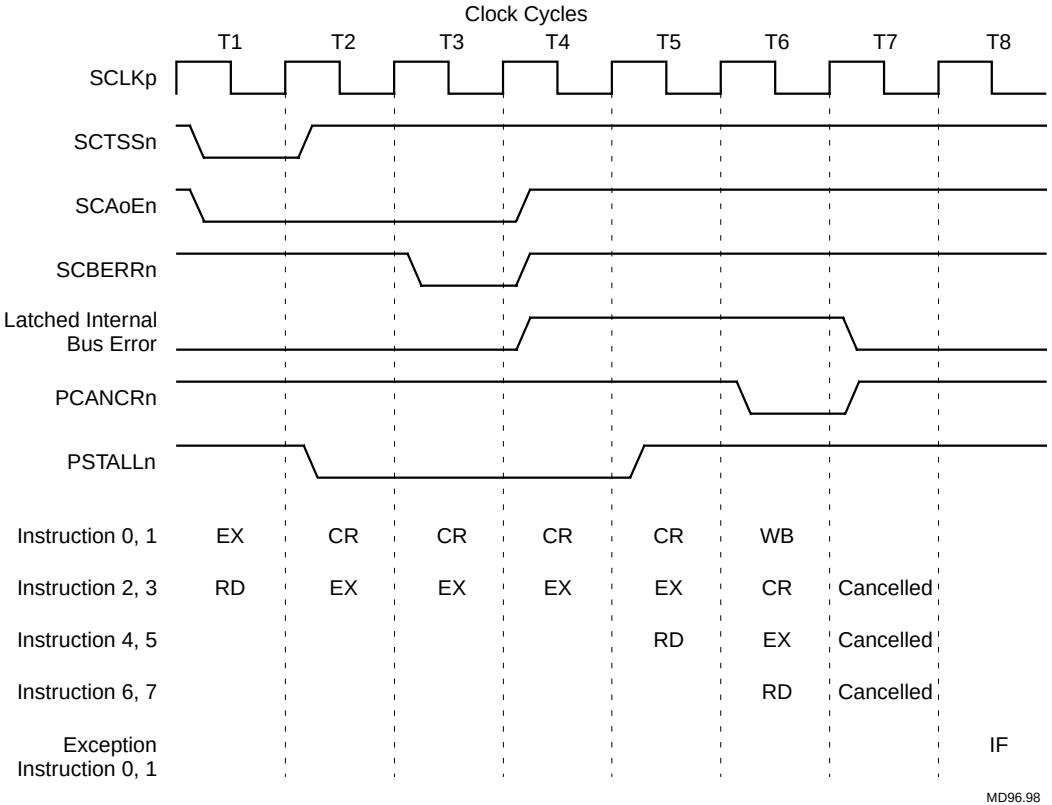


Figure 8.5
Bus Error and Pipeline
(With Stall Cycles)



8.1.5.1 Handling Bus Errors

The common exception vector, shown in Table 8.1, is used for the SCBERRn exception. The ExcCode field in the Cause register is set to Bus.

CommonExceptionVector

Status Register DEV	CCC Register R3000 Mode	CCC Register R4000 Mode
0	0x80000080	0x80000180
1	0xBFC00180	0xBFC00380

The EPC register points at the first instruction for which processing was not completed, unless this instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set.

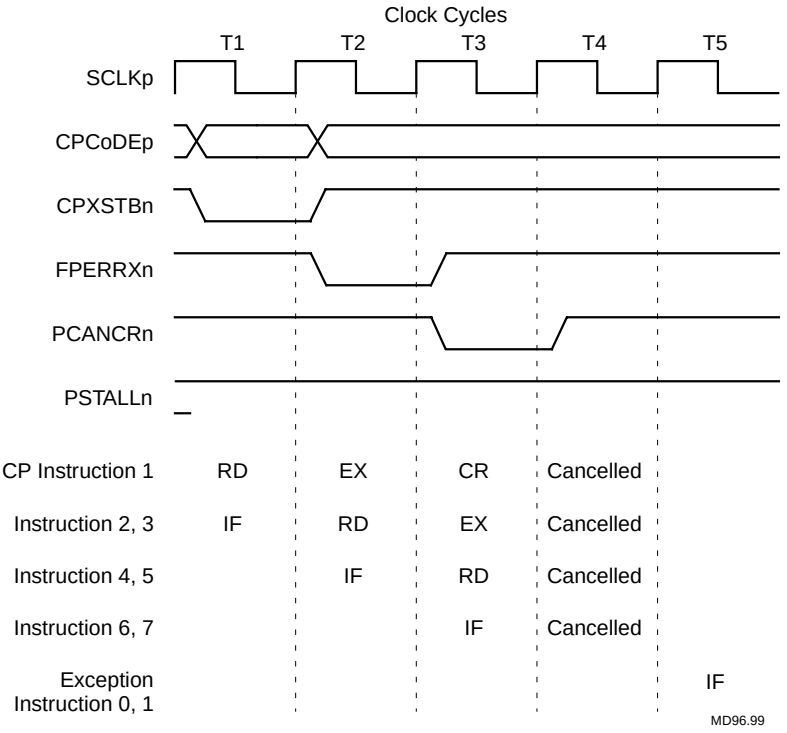
8.1.5.2 Servicing Bus Errors

The physical address at which the fault occurred is not available to the exception handler. The process executing at the time of the exception must be handed a bus error signal, which is usually fatal.

8.1.6 Floating-Point Unit (FPERRXn) Exceptions

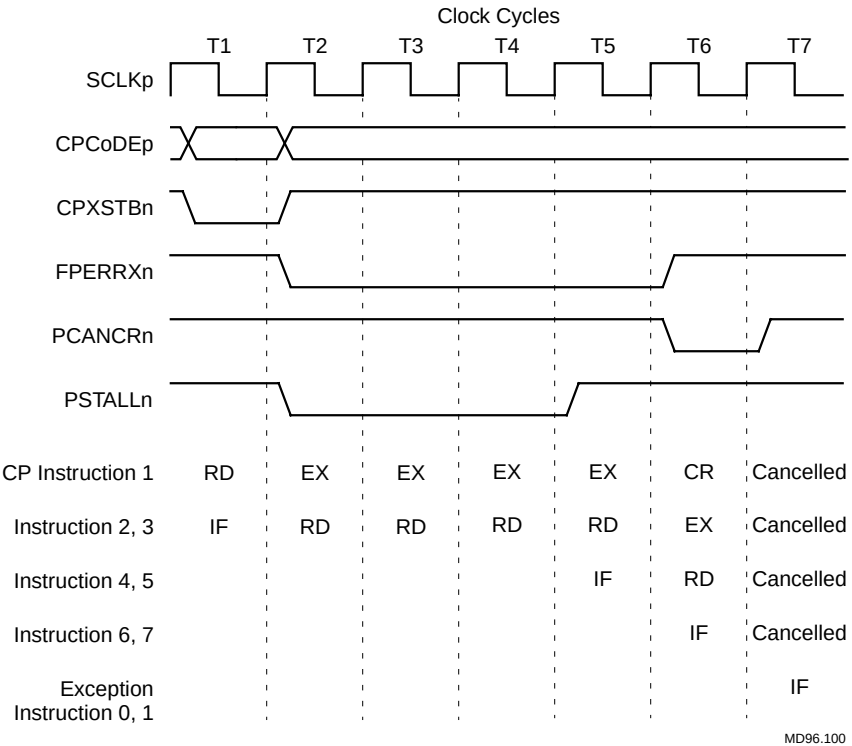
The CW4010 coprocessor interface uses the FPERRXn input to detect a Floating-Point Unit (FPU) exception. Refer to Section 8.5, “Coprocessor Interface Behavior” on page 8-46, for more details about the FPERRXn. Figure 8.6 shows the case where FPERRXn is serviced immediately.

Figure 8.6
FPU Exception and Pipeline (Detected Immediately)



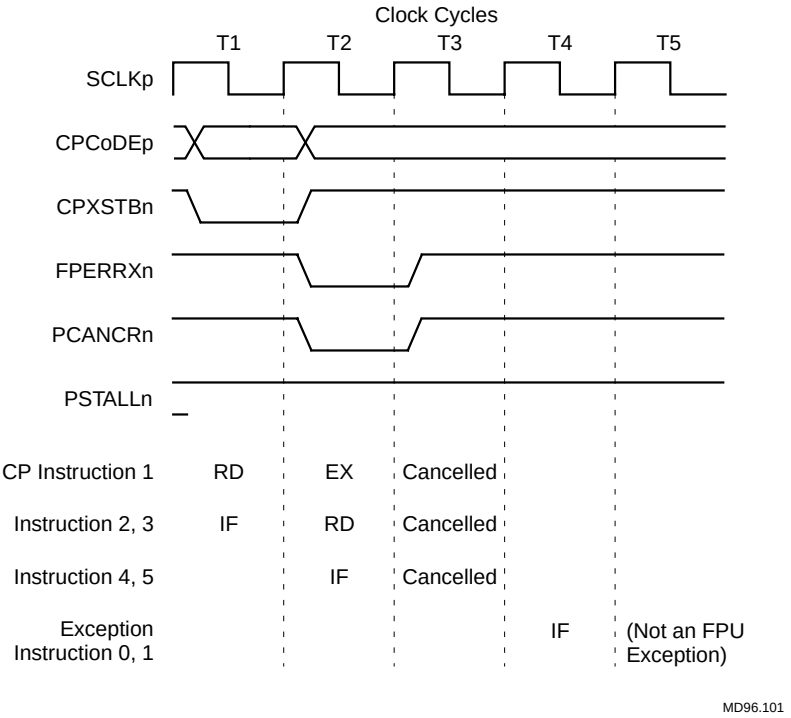
If the CW4010 enters a stall condition at the EX stage and FPERRXn is asserted, it should be asserted continuously until the stall condition is resolved. FPERRXn is not latched in the CW4010. This condition is shown in Figure 8.7.

Figure 8.7
FPU Exception and Pipeline (With Stall Cycles)



If the previous instruction cancels the pipeline (the pipeline cancel signal is asserted at the same time as FPERRXn), FPERRXn is ignored and must be deasserted. FPERRXn should be asserted later when the coprocessor instruction is re-executed. This condition is shown in Figure 8.8.

Figure 8.8
FPU Exception and Pipeline (Cancel, then not Serviced)



8.1.6.1 Handling FPU Exceptions

The common exception vector is used for the FPERRXn exception. The ExcCode field in the Cause register is set to FPE (15).

The coprocessor asserts FPERRXn at the CR stage and the CW4010 samples it at the end of the EX stage. The coprocessor instruction in this slot causes the FPERRXn exception.

The EPC register points at the coprocessor instruction, which starts at the RD stage just before the coprocessor asserts FPERRXn. If the instruction is in a branch delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set.

Refer to Section 4.4.18, “Floating-Point Exception” on page 4-45 for further information on this subject.

8.1.6.2 Servicing FPU Exceptions

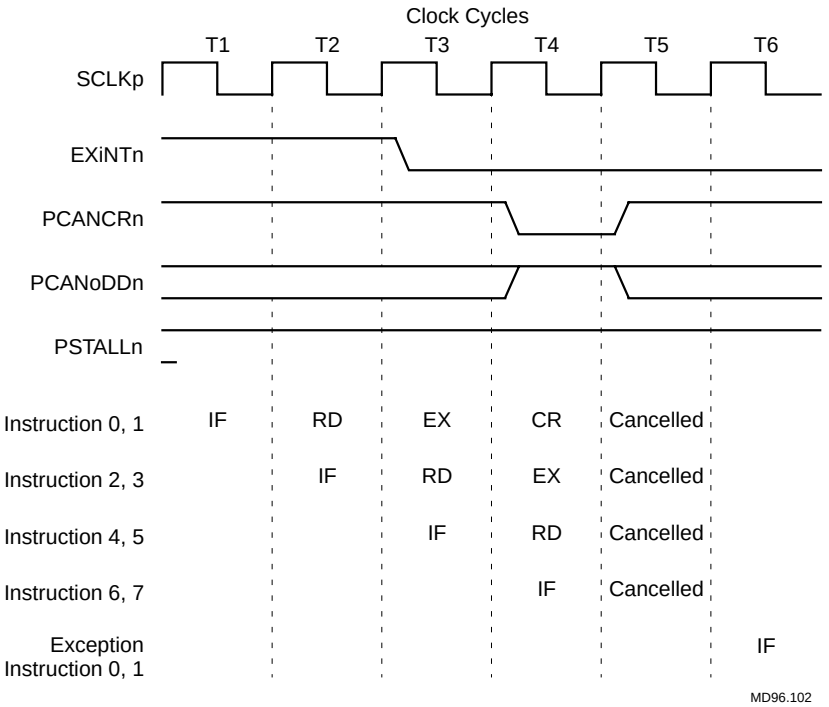
Coprocessor interface signals indicate whether the coprocessor instruction is in an even or odd slot. The coprocessor must control these signals correctly to synchronize the exception and the pipeline flow.

The FPERRXn exception input can be used as a general purpose exception input.

8.1.7 External Interrupts (EXTiNTn)

The CW4010 has six external interrupt inputs, EXiNTn[5:0], which must be asserted and deasserted on the rising edge of the system clock. To mask all six external interrupts at once, you can clear the IE bit of the Status register. To mask each interrupt individually, program the INT bits in the Status register (Refer to Section 4.3.6, “Status Register (12)” on page 4-9 for further information about this register.) The instruction fetch for the exception procedure starts two clocks after an external interrupt has been detected, provided that the pipeline is not in a stall state and there is no higher priority exception. Figure 8.9 shows the timing diagram where an interrupt is immediately detected.

Figure 8.9
Interrupt And
Pipeline (Interrupt
Is Detected
Immediately)



An EXTiNTn exception is similar to an NMin exception, except that external interrupts are not latched internally, and must be asserted until they are serviced. If the pipeline is in a stall cycle, the CW4010 does not service interrupts until the stall condition is resolved.

8.1.7.1 Handling External Interrupts

The common exception vector is used for the EXTiNTn exception. The ExcCode field in the Cause register is set to INT0.

The IP field of the Cause register indicates the current interrupt requests. More than one of the bits may be set at the same time. None of the bits may be set, if an interrupt is asserted and then deasserted before the CW4010 reads the Cause register.

The EPC register points at the first instruction for which processing was not completed, unless this instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set. Refer to Section 4.4.21, “Interrupt Exception” on page 4-47 for further information on this subject.

8.1.7.2 Servicing External Interrupts

If one of two software generated exceptions causes the interrupt, clear the corresponding Cause register bit to zero to clear the interrupt condition.

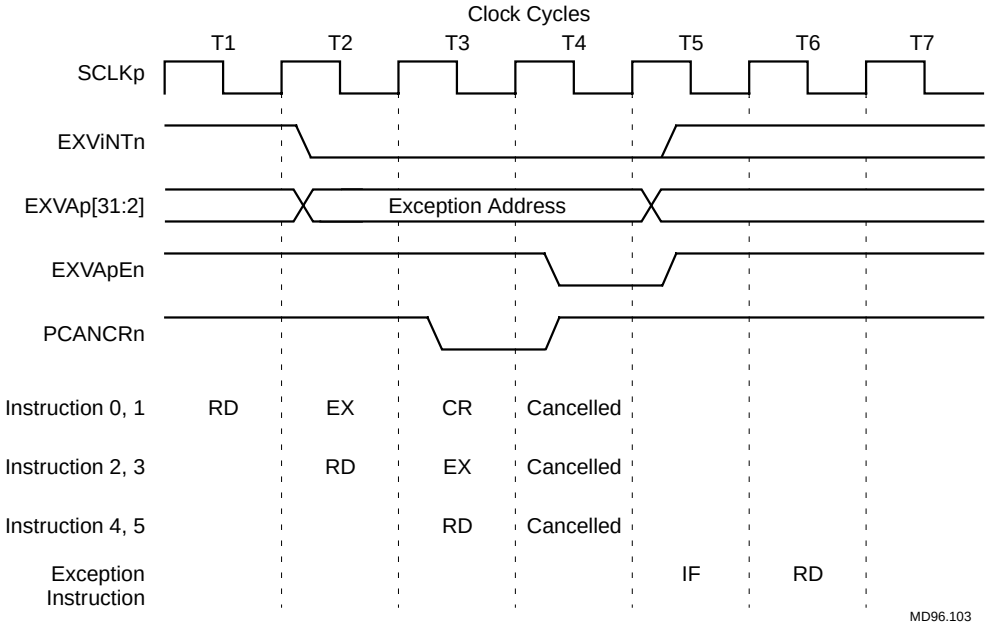
If the interrupt is hardware generated, correct the condition that caused the assertion of the interrupt pin to clear the interrupt condition.

8.1.8 External Vectored Interrupt (EXViNTn)

The CW4010 has an External Vectored Interrupt input, EXViNTn. The EXVAp[31:2] inputs provide the interrupt vector virtual address, so the common exception vector base and offset are not used.

EXViNTn must be asserted and deasserted on the rising edge of the system clock. When the EXViNTn has been sampled and found active on the rising edge of the clock, CP0 samples an exception vector from EXVAp[31:2], which is available when the enable bit EVI in the CCC is set. To mask the EXViNTn interrupt at once, you can clear the IE bit of the Status register. Figure 8.10 shows the fastest accepted case of EXViNTn. If the pipeline is stalled, it requires more clock cycles. When EXVApEn is asserted, the system may drive EXVAp[31:2].

Figure 8.10
Fastest Accepted Case
of External Vectored
Interrupt



8.1.8.1 Handling External Vectored Interrupts

The External Vectored Interrupt feature is available when the EVI bit in the CCC register is set.

EXViNTn has lower priority than the six external interrupts EXiNTn[5:0], but higher priority than the debug exception. To mask EXViNTn, you can use the interrupt enable bit in the Status Register in a similar way to that used for external interrupts.

If EXViNTn is accepted, the CP0 reads the exception vector address on EXVAp[31:2] and writes it into the Program Counter directly. A user-defined interrupt controller provides EXVAp[31:2], so that the CW4010 jumps to the interrupt handler directly when it is requested. EXVAp[31:2] must be stable until EXVApEn is asserted. EXViNTn does not alter anything in the Cause Register except the BD bit.

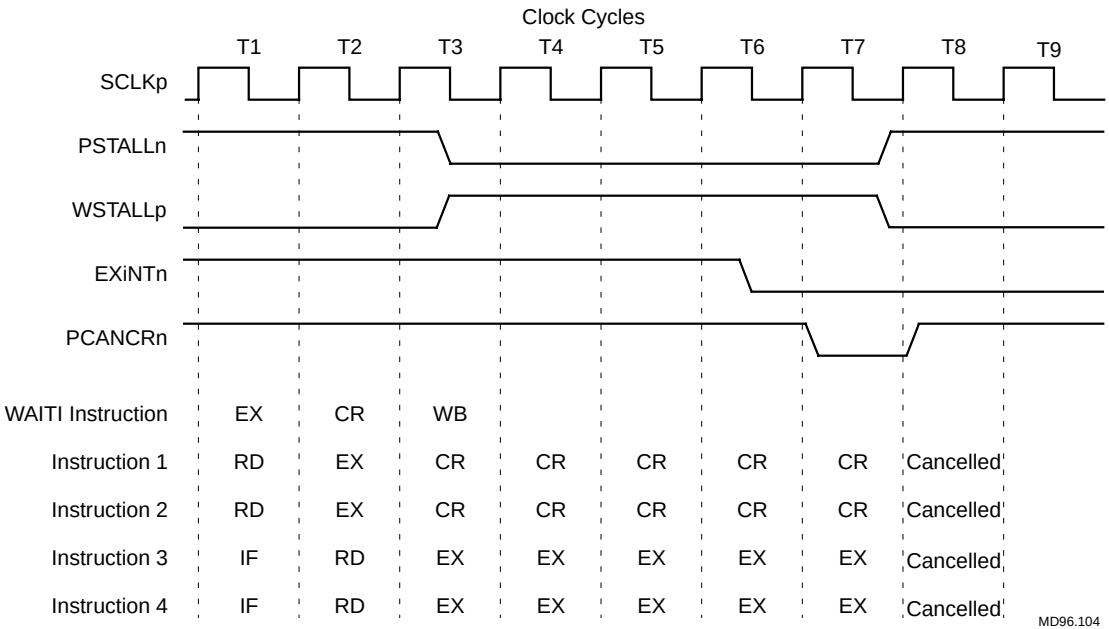
The EPC register points at the first instruction for which processing was not completed, unless this instruction is in a branch delay slot. If the

instruction is in a branch delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set. Refer to Section 4.4.22, “External Vectored Interrupt Exception” on page 4-48 for further information on this subject.

8.1.9 WAITI Instruction and WSTALLp

The CW4010 uses the WAITI instruction, which is one of its extended instructions, to initiate a wait state. This stalls the pipeline and reduces power consumption during the period that the CW4010 is inactive. The CW4010 wakes up when it detects an external exception input (enabled interrupt, NMin, warm reset, or cold reset). Figure 8.11 shows the timing diagram for the WAITI instruction.

Figure 8.11
WAITI and Pipeline Stall (WSTALLp)



The Coprocessor Interface signal, PSTALLn, is also asserted when the pipeline stage is in the stall condition.

At T1, the CP0 starts executing a WAITI instruction. At T3 (which occurs in the WB stage of the pipeline), the CP0 requests pipeline stall and the CW4010 asserts WSTALLp. At T6, an external interrupt input is asserted and the CW4010 wakes up from T7. At T8, the instructions in the pipeline stages are cancelled, and the IF stage for the exception is started from T9.

8.2 SCbus Interface Behavior

The CW4010 generates one or more external data read/write transaction(s) on the SCbus under any of the following conditions:

- Uncached area instruction fetch
- Icache-miss
- Uncached area data read/write
- Load Dcache-miss
- Any store execution in WriteThrough mode
- Dcache WriteBack

The SCbus is a flexible address/data bus. It is demultiplexed and synchronized to the system clock. It has a data width of 64 bits, but supports one type of bus sizing from a 64-bit width to a to 32-bit width. The SCbus has the following transaction data sizes: byte, halfword, tribyte, 32-bit word, 64-bit doubleword, or 8-word burst (4-doubleword burst), as shown in Table 8.2

Table 8.2
SCbus Transaction
Types

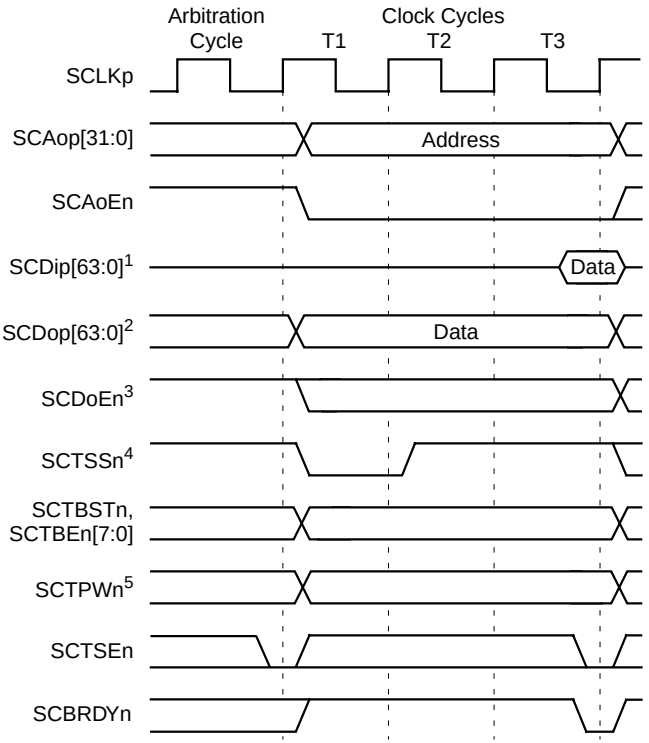
Cause of SCbus Transaction	Transaction Type	No. of Bytes
Uncached Instruction Fetch	Doubleword	8
Instruction Cache-Miss	8 words	32
Data Read by Uncached Load Instruction	Byte, halfword, tribyte, word	1, 2, 3, 4
Data Read by Dcache-Miss Load Instruction	8 words	32
Data Write by Uncached Store Instruction	Byte, halfword, tribyte, word, doubleword	1, 2, 3, 4, 8
Data Write by Dcache-Miss Store Instruction	Byte, halfword, tribyte, word, doubleword	1, 2, 3, 4, 8
Data Write by WriteThrough Store Instruction	Byte, halfword, tribyte, word, doubleword	1, 2, 3, 4, 8
Data Write by WriteBack	8 words	32

The CW4010 has a four line depth write buffer for uncached, Dcache Miss, or WriteThrough store operations. Each line in the buffer contains 32-bits of address and 64-bits of data. If word data is stored to a continuous same-doubleword alignment address, two words are stored in one line. The CW4010 then requests a doubleword write transaction on the SCbus, which the sizing function can separate into two 32-bit write transactions.

8.2.1 SCbus Basic Transaction

Figure 8.12 shows a basic SCbus transaction for a single read and write. It is a three-clock-cycle transaction, which means that the SCBRDYn assertion is sampled on the rising edge of the third clock edge from the beginning of the transaction cycle. The number of clock cycles for the fastest transaction is one clock, in which case SCTSSn is asserted continuously if the next transaction starts just after the current one. There is no limit to the maximum number of clock cycles for a transaction. A bus watch dog timer must be designed outside the core to assert the bus error signal SCBERRn, if necessary, when the transaction length is longer than the specification.

Figure 8.12
SCbus Basic
Transaction



At the beginning of a transaction, the transaction start strobe SCTSSn is asserted for one clock cycle. In addition, the address is output on the SCAop[31:0] lines and the address output enable signal SCAoEn is asserted to indicate that SCAop[31:0] is valid.

The Byte Enable signals SCTBEn[7:0] are also output. If the transaction is a four doubleword burst, SCTBSTn is asserted during the first transaction. If the transaction is an In-Page write, which means that the next transaction is in the same page, SCTPWn is asserted. It is not asserted for burst write transactions. The SCTBSTn and SCTPWn status indication signals are valid by the end of the transaction.

If the transaction is a data write, data is output to the SCDop[63:0] lines and SCDoEn is asserted from the beginning to the end of the transaction. If the transaction is a data read or instruction fetch, the SCDip[63:0] signal lines are sampled on the clock edge as the ready input SCBRDYn is asserted. The data output enable signal SCDoEn then indicates the read/write direction of the transaction and controls the three-state buffers external to the CW4010.

Asserting SCBRDYn terminates the transaction. At the same time, the size input bus signal SCB32n is sampled. According to the input, the Bus Interface Unit (BIU) of the CW4010 determines the valid byte positions for the read transaction bus sizing. If SCBRDYn is asserted for a doubleword transaction, the bus interface generates a subsequent transaction for bus sizing. The bus In-Page write accept input SCBPWAn is also sampled in an In-Page write transaction. If SCBPWAn is deasserted, the bus interface arbitrates bus requests even if the next transaction is a write transaction in the same memory page. If SCBPWAn is asserted, the bus interface does not arbitrate bus requests and the next transaction must be a write transaction in the same memory page. If SCBPWAn is asserted during the In-Page write transaction but SCTSEn is deasserted, the next transaction is a write transaction in the same page.

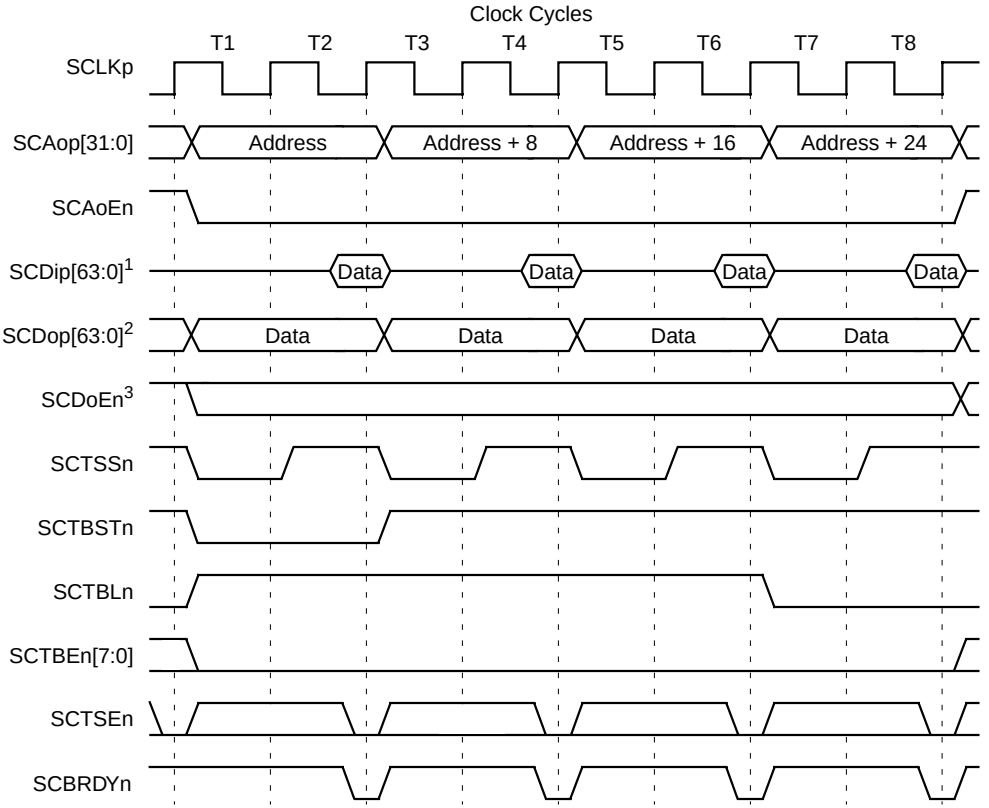
To perform an instruction fetch transaction, the CW4010 asserts SCiFETn during the same period as SCAoEn in order to monitor the transaction.

8.2.2 SCbus Burst Transaction

When an Icache miss occurs, the Instruction Scheduler Unit (ISU) requests an 8-word (4-doubleword) block burst read. When a Dcache miss occurs, the Load Store Unit (LSU) requests an 8-word block burst read. The LSU also requests an 8-word block burst write for Dcache WriteBack.

Figure 8.13 shows an eight-word burst read/write transaction that consists of four continuous transactions.

Figure 8.13
SCbus Eight-Word
Burst Transaction
Timing Chart



1. Read cycle.
2. Write cycle.
3. High = read, low = write.

MD96.106

In the first transaction, the burst transaction indicator signal, SCTBSTn, is asserted to indicate an 8-word burst transaction. Subsequent transactions are single doublewords transactions. Each transaction is terminated by an assertion of the bus ready signal, SCBRDYn. The transaction start signal, SCTSEn, is asserted for each transaction. Burst transactions can be suspended if SCTSEn is deasserted. The bus hold request signal, SCHRQn, is not accepted during a burst transaction if SCTSEn is not deasserted when SCBRDYn is asserted. SCHRQn is accepted if SCTSEn is asserted to insert one or more idle cycles when SCBRDYn is asserted.

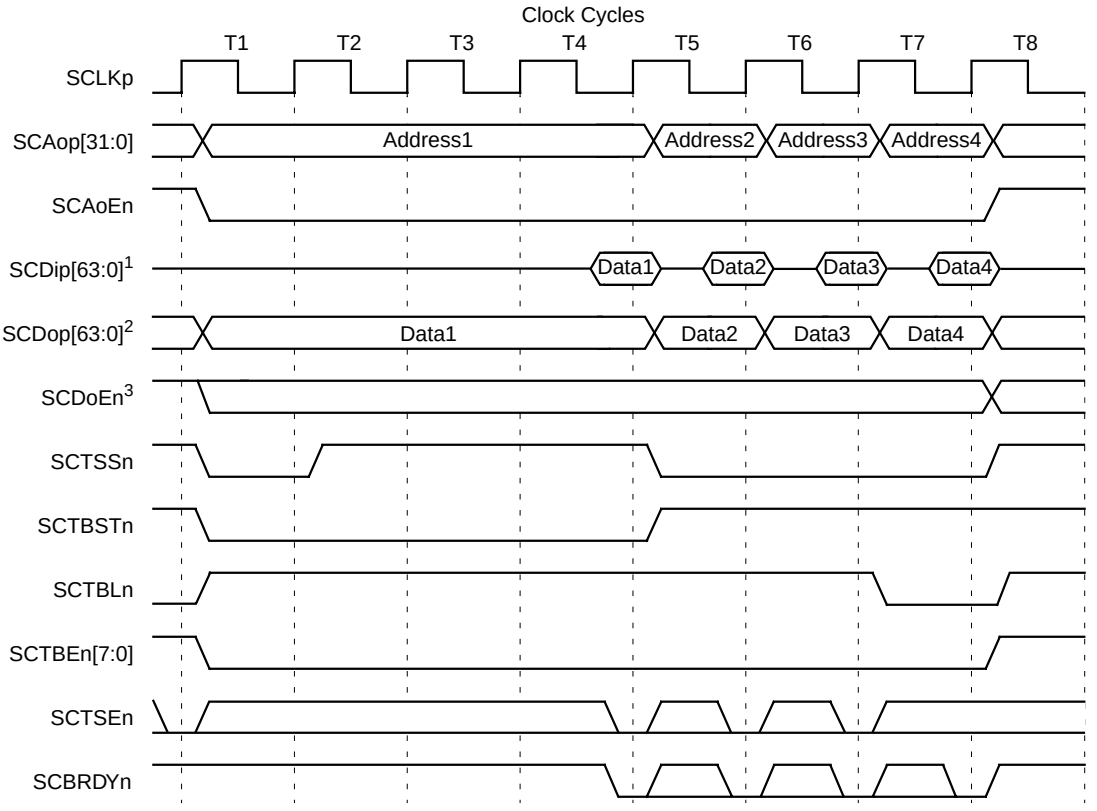
SCTBLn, which indicates whether the last transaction is a burst or a single transaction, is deasserted (HIGH) at the first, second, and third transactions of a four doubleword burst transaction.

For a burst read transaction, the first address is the missed address. The addresses of the subsequent transactions are rotative and wrap around ordering in the block. For a burst write transaction, the first address is the beginning of the block and subsequent addresses are incremental.

Bus sizing for a burst transaction is available to allow the SCbus to accomplish burst transactions to 32-bit width devices. The SCB32n input must be asserted for each transaction of burst transactions. If 32-bit sizing is requested for a burst transaction, eight word transactions are generated. SCTBLn is deasserted from the first to the sixth transaction. The In-Page write never occurs if the transaction is a burst write.

Figure 8.14 shows a timing diagram for an eight-word burst transaction. If the bus slave of the transaction is a synchronous DRAM system, there are some wait cycles for the first data transfer, but not for subsequent transfers. For a synchronous DRAM system, SCTSSn is asserted continuously for the second, third, and fourth data transfers. The DRAM controller generates addresses for these data transfers itself although SCAop also outputs addresses.

Figure 8.14
SCbus Eight-Word
Burst Transaction

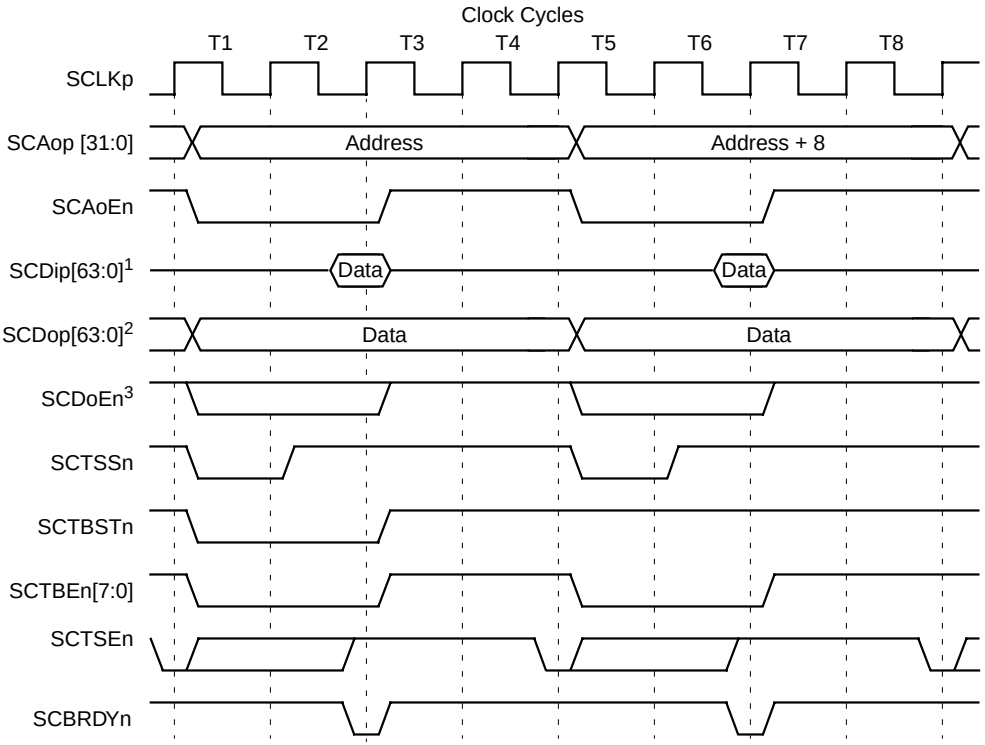


1. Read cycle.
2. Write cycle.
3. High = read, low = write.

MD96.107

Figure 8.15 shows the first and second transactions of an eight-word burst read/write. Transactions are suspended when SCTSEn is deasserted.

Figure 8.15
SCbus Eight-Word
Burst Transaction
Timing Chart



1. Read cycle.
2. Write cycle.
3. High = read, low = write.

MD96.108

If an individual transaction of a burst transaction is terminated with the deassertion of SCTSEn, this means the next transaction cannot proceed continuously. In that case, a hold request can be inserted. A hold request can also be inserted if a retry occurs while SCTSEn is deasserted during a burst transaction.

8.2.3 SCbus In-Page Write Transaction

An In-Page write transaction is one in which continuous write accesses are made to the same row and page in a given address area. Most types of DRAM support this type of fast access, which is used to perform burst read/write transactions.

The SCbus supports continuous write transactions that have the same upper address. The external write buffer in the Load Store Unit compares upper address bits of the current write request with those of the next write transaction in the buffer. It provides the bus interface with the result of the comparison. The address range is defined in the configuration register of the CW4010. If the two addresses have the same upper range, the In-Page write output SCTPWn is asserted to inform the external bus slave.

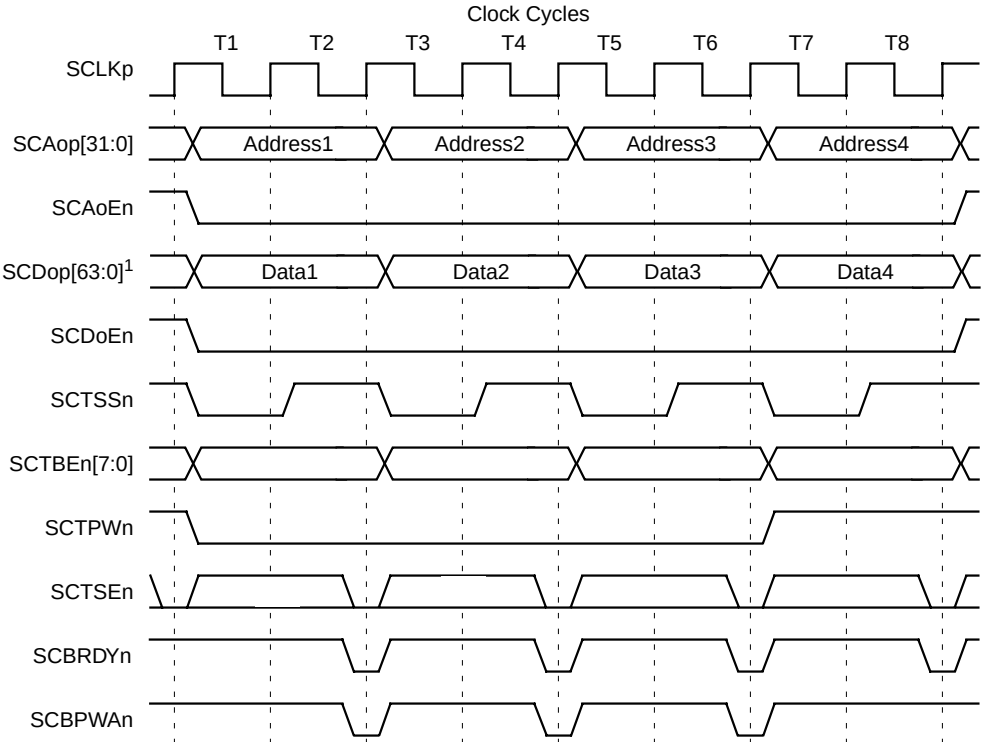
The In-Page write accept input SCBPWAn must be asserted if the slave is able to accept In-Page write transactions. If SCBPWAn is asserted, the interface does not arbitrate bus requests and the next transaction must be a write transaction. If SCBPWAn is deasserted, the bus interface performs the next transaction according to the arbitration result. SCBPWAn is sampled when the bus interface samples an assertion of the SCBRDYn signal. The bus interface performs a write transaction if SCBPWAn is deasserted and there are no higher requests. The SCBPWAn input has no meaning if the transaction is not an In-Page write, and it is ignored when SCTPWn is deasserted.

The bus interface does not count the number of continuous In-Page write transactions. It continues In-Page writes until the write buffer is empty, a write transaction is not in the same page address area, or SCBPWAn is deasserted.

When the BIU deasserts the transaction start enable signal, SCTSEn, the CW4010 inserts one or more bus idle states between two In-Page write transactions. However, the bus interface does not arbitrate requests during this idle state if the slave accepts the In-Page write transactions. A hold request is allowed if the BIU deasserts SCTSEn. The bus interface does not accept the bus hold request during In-Page write transactions if the BIU receives an asserted SCTSEn continuously.

Figure 8.16 shows an example of In-Page write transactions.

Figure 8.16
SCbus In-Page Write
Transaction Timing
Chart (four words)



8.2.4 SCbus Bus Hold

There are two ways to hold SCbus transactions:

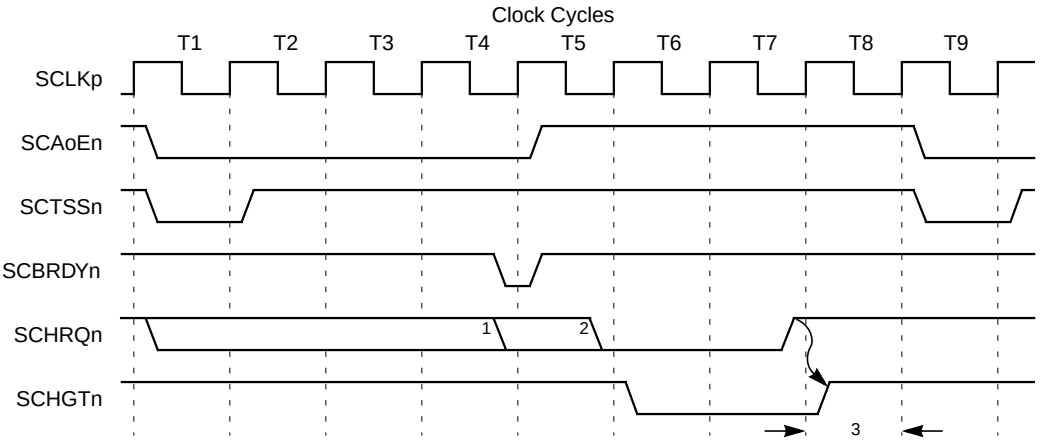
- External logic asserts the CW4010 bus hold input SCHRQn. The CW4010 acknowledges the request by issuing the bus hold grant signal, SCHGTn.
- External logic deasserts the transaction start enable signal, SCTSEn. Because there is no dedicated acknowledge signal associated with SCTSEn, the CW4010 deasserts the address output enable signal, SCAoEn, after the BIU deasserts SCTSEn to show that the bus interface does not own the bus.

The bus hold request signal, SCHRQn, cannot break In-Page write transactions and read/write burst transactions if SCTSEn is asserted continuously. The BIU can break the transactions when it deasserts SCTSEn.

To avoid a bus deadlock, a bus retry is requested with each hold request. The current SCbus transaction generated by the BIU is then terminated by the retry and the hold request must be accepted.

Figure 8.17 shows the timing diagram for a bus hold request and the associated grant signal, SCHGTn. The CW4010 asserts the grant signal until the BIU deasserts the request. During the period the bus is held, the CW4010 does not detect bus errors.

Figure 8.17
SCbus Hold Request
and Grant



1. If BIU has next transaction request, SCHRQn must be asserted before this cycle.
2. If BIU has no next transaction request, SCHGTn is asserted immediately.
3. Minimum 1 SCLK.

MD96.110

8.2.5 SCbus Bus Retry

The bus retry signal, SCBRTYn, is an input to the BIU. It is asserted to abort a transaction and to allow the transaction to be restarted later. The transaction state control goes to the idle state then restarts a transaction when SCTSEn is asserted. Bus retry is valid in a burst transaction. If SCBRDYn and SCBRTYn are asserted at the same time, SCBRTYn has higher priority. If SCBRTYn is asserted to hold the bus, SCHRQn should be asserted before or at the same time as SCBRTYn.

8.2.6 SCbus Bus Error

The external bus controller asserts the BIU bus error signal, SCBERRn, when the current transaction must be terminated as a bus error. If SCBRDYn is asserted at the same time, SCBERRn has higher priority.

Assertion of SCBERRn forces the CW4010 to exit the sequential transactions of In-Page write and read/write burst transactions. The states of service and transaction control go to the idle state. If the transaction is a burst (cache refill or WriteBack), the CW4010 invalidates the cache line.

When a bus error occurs, the CP0 issues a bus error exception. See the Section 8.1.5, “Bus Error (SCBERRn)” on page 8-7 for more details. A bus error exception is a fatal error for the CW4010.

8.2.7 SCbus Bus Sizing

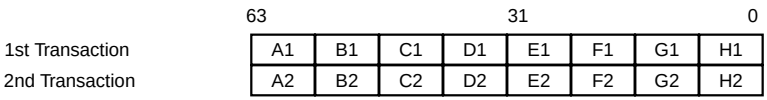
The SCbus supports bus sizing for slaves that need sequential address access to 32-bit data. When sizing is requested, the SCB32n input to the CW4010 is asserted to separate a doubleword transaction, including part of a burst transaction, into two singleword transactions. The bus interface also selects valid byte positions for a word or a partial word transaction if SCB32n is asserted. In the case of a word or a partial write transaction, the bus interface outputs word data to both the upper and lower 32 bits of the data output bus according to address bit 2. The bus interface then completely supports a 32-bit bus interface. Although SCB32n is sampled with the assertion of the ready signal input, the bus interface behaves as a normal 32-bit data width bus if SCB32n is always asserted.

If 16-bit or 8-bit width bus sizing is needed, it must be supported outside the CW4010 core.

8.2.7.1 Read Bus Sizing

When sizing occurs at a byte, halfword, tribyte, or word during a read transaction, the CW4010 BIU can move sampled 32-bit word data to the valid position according to the setting of address bit 2. If sizing is requested for a doubleword transaction, the BIU samples 32-bit data at the first transaction then generates a subsequent transaction and packs the first 32 bits and the subsequent 32 bits. The packed data is sent to the Instruction Scheduler Unit or Load Store Unit. Figure 8.18 shows the relationship between the valid byte positions of the first and subsequent transactions. In the case of a non-doubleword read, the behavior of a byte, a halfword, and a tribyte transaction is the same as that of a word transaction because sizing supports 32-bit mode only. You can assume that the bus interface samples a doubleword (8 bytes). Figure 8.18 shows an example in which the bus interface samples a doubleword (8 bytes).

Figure 8.18 Sampled Bytes of First and Second Transaction SCbus Data

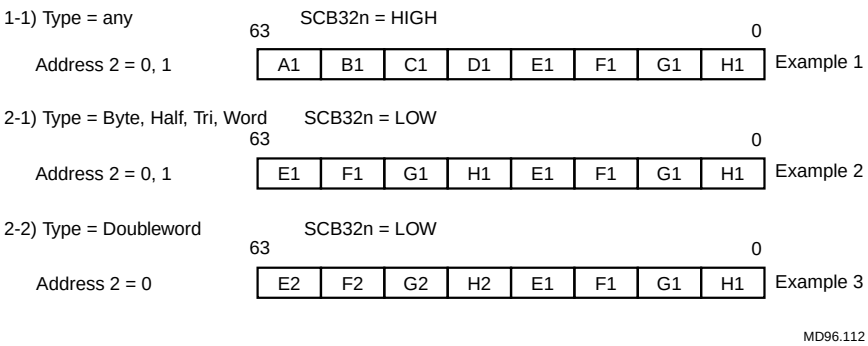


1. The 2nd transaction is generated when the transaction is a doubleword or a part of a burst with SCB32n = LOW.

MD96.111

If you are reading a doubleword and doing bus sizing, you will need a second transaction. Figure 8.19 shows the doubleword data that is sent to the Instruction Scheduler Unit or Load Store Unit.

Figure 8.19
Read Bytes to ISU and LSU with Sizing

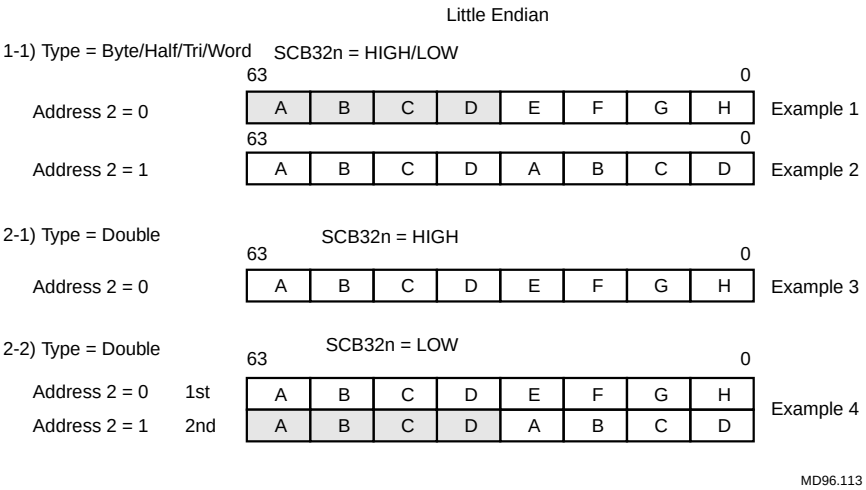


1. In Example 1, one transaction is initiated. The eight bytes sampled are transferred to the ISU and LSU without any change.
2. In Example 2, one transaction is initiated. Four bytes are sampled (bits [31:0]). They are transferred to bits [63:32] and [31:0] of the ISU and LSU.
3. In Example 3, two transactions are initiated. Bits [31:0] of the first transaction are output on bits [31:0], and bits [31:0] of the second transaction are output on bits [63:32]. This doubleword is transferred to the ISU or the LSU.

8.2.7.2 Write Bus Sizing

In the case of a non-doubleword write transaction, the bus interface selects the upper or lower 32 bits of data from the Load Store Unit and outputs the same 32-bit word data, which are valid bytes, to the SCbus according to address bit 2. The data is output to the SCbus before the bus interface detects the sizing input. In the case of a doubleword write transaction, the bus interface generates a subsequent sizing transaction if the sizing input is asserted at the first transaction. Figure 8.20 shows the relationship between the doubleword data from the Load Store Unit and the SCbus. Bytes shown in the shaded area have no meaning for the SCbus write transaction.

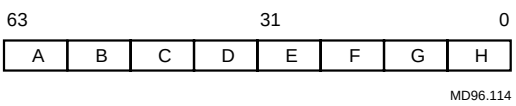
Figure 8.20
Write Bytes to the SCbus with Sizing



1. In Example 1, one transaction is initiated. A doubleword from the LSU is output on the date bus without any changes.
2. In Example 2, one transaction is initiated. Bits [63:0] from the LSU are output on bits [63:32] and [31:0] of the data bus.
3. In Example 3, one transaction is initiated. A doubleword from the LSU is output on the data bus without any change.
4. In Example 4, two transactions are initiated. In the first transaction, a doubleword from the LSU is output on the data bus without any change. In the second transaction, bits [63:32] from the first transaction are output to bits [31:0] of the data bus.

As shown in Figure 8.21, you can assume that the LSU sends a doubleword, regardless of the transaction type.

Figure 8.21
Write Data Bytes from LSU



8.2.8 SCbus Bus Lock

The CW4010 SCLoCKn output signal indicates that the SCbus is asking to lock ownership. The CW4010 asserts SCLoCKn when the CW4010 executes a LoadLink instruction to start a read transaction in an uncached area or WriteThrough cached area. It deasserts the signal just before it executes a StoreConditional instruction to start a write transaction. During the read transaction and the write transaction, the CW4010 asserts SCLoCKn continuously.

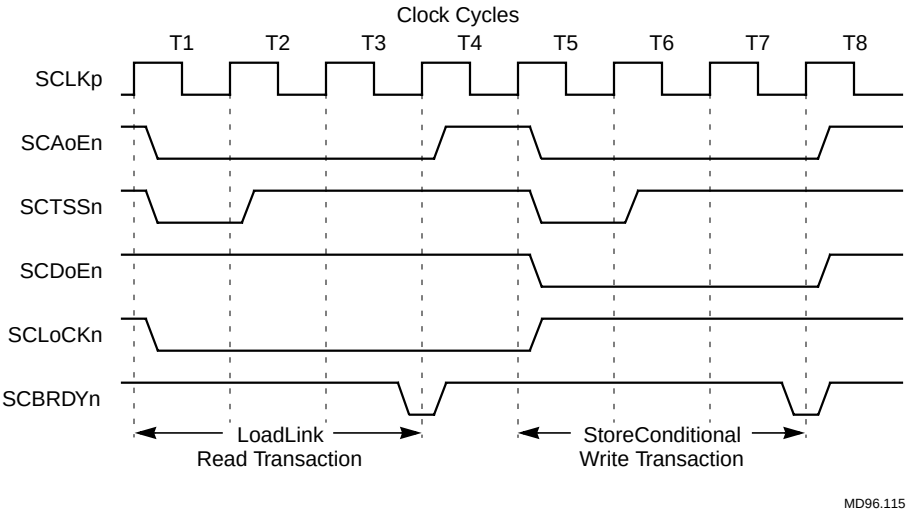
If an effective address for a LoadLink instruction is in the WriteBack cached area, the CW4010 does not assert SCLoCKn, even if it experiences a Dcache miss. The subsequent StoreConditional instruction does not generate a write transaction because it may hit the Dcache. If a StoreConditional instruction hits the Dcache in a WriteBack cached area when SCLoCKn is asserted, an incorrect condition occurs, and SCLoCKn is deasserted without any bus transactions being executed.

The effective virtual addresses of LoadLink and Store Instructions must be in kseg1. Additionally, a LoadLink instruction and a StoreConditional instruction must be used as a pair of instructions to the same address.

While the CW4010 asserts SCLoCKn, the bus interface does not exhibit any special behavior—for example, it accepts hold requests. If a hold request is not accepted while the CW4010 is asserting SCLoCKn, outside user logic must mask the hold request by asserting SCLoCKn.

Figure 8.22 shows the timing behavior for locked transactions. If there are other transactions between the read transaction of a LoadLink and write transaction of a StoreConditional, the CW4010 asserts SCLoCKn continuously.

Figure 8.22
SCbus Locked
Transaction



8.2.9 Big Endian Configuration

The CW4010 can support big endian address ordering, although the default configuration is little endian. To enable the big endian configuration, the CW4010 straps the BENDn input low. BENDn aligns byte positions in a 32-bit word in the LSU. Bits [31:0] must be swapped outside the CW4010 with bits [63:32], for both SCDip and SCDop, as shown in Table 8.3.

The ISU assumes the first instruction (even slot) is fetched through SCDip[31:0], and the second instruction (odd slot) is fetched through SCDip[63:32].

The LSU aligns byte positions in a 32-bit word according to the state of BENDn input. The LSU assumes the following:

- Full word or partial word data is accessed through the input bus bits SCDip[31:0] when address bit 2 is 0.
- Output bus bits SCDop[31:0] are accessed through SCDip[63:32] or SCDop[63:32], when address 2 is 1, regardless of the state of BENDn.

The byte enable output lines SCTBEn[7:0] indicate valid bytes according to the byte address for both big-endian and little-endian address

ordering. Table 8.3 shows the relationship of SCTBEn[7:0] to the SCDip and SCDop bit positions.

Table 8.3
SCTBEn and Valid SCDp

	Valid SCDp Bits	
SCTBEn	**Little Endian**	**Big Endian**
0	[7:0]	[63:56]
1	[15:8]	[55:48]
2	[23:16]	[47:40]
3	[31:24]	[39:32]
4	[39:32]	[31:24]
5	[47:40]	[23:16]
6	[55:48]	[15:8]
7	[63:56]	[7:0]

8.3 OCAbus Interface Behavior

The CW4010 On-Chip Access (OCA) bus enables access to on-chip modules at the CR stage without going through the SCbus. Section 7.4.6, “OCAbus Interface Signals” on page 7-16 provides additional information about the bus. This section describes certain OCAbus transactions and provides timing diagrams for them. These transactions include:

- A basic OCA access
- Rejection of an OCA access
- An OCAbus access with a stall at the EX stage of the CW4010 pipeline
- An OCAbus access with a stall at the CR stage of the CW4010 pipeline
- An OCAbus access with a stall request or wait state
- An OCAbus access with a pipeline cancellation

8.3.1 Basic OCAbus Transaction

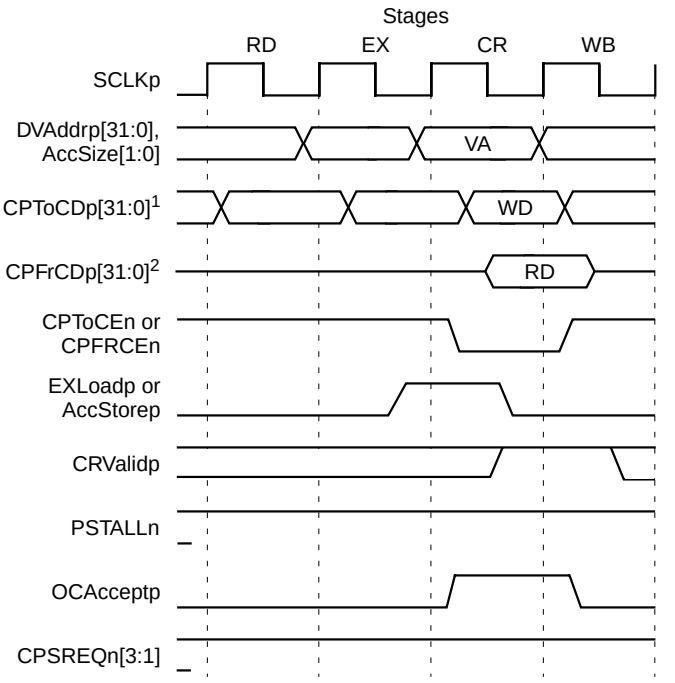
Regardless of the type of load or store execution, address and size are output at the EX stage of the CW4010 pipeline, and EXLoadp or AccStorep is asserted. The address bits (DVAddrp[31:0]) need to be decoded to determine whether or not OCAcceptp is asserted and the OCA module can accept the OCA transaction.

Typically, OCA modules should be located as uncached devices, so that the virtual address is in kseg1. This is done by setting address bits [31:29] to 1 0 1. The address bus must be latched on the rising edge of the system clock, between the EX and CR stages, as shown in Figure 8.23. The size information provided by AccSize is also latched at this time. Refer to subsection entitled “AccSize[1:0] OCAbus Transaction Size Output” on page 7-17 for more information on this subject.

At the CR stage, write data is output on CPToCDp provided that CPToCEn is asserted. If a read transaction is being executed, the CPFRCEn signal is asserted and data on the CPFRCDp bus is sampled on the rising edge of the system clock between stages CR and WB. The OCAcceptp signal must be asserted in the CR stage to inform the CW4010 that an OCA transaction is in progress.

The CRValidp signal is asserted to indicate that the CR stage is valid. If it is deasserted, write data must not be written and read data must not be sampled. The transaction is executed again later.

Figure 8.23
Typical OCAbus
Transaction



1. Write cycle.
2. Read cycle.

MD96.116

8.3.2 OCAbus Transaction Rejected

Figure 8.24 shows the timing for an OCAbus transaction that is rejected because OCAcceptp is deasserted. This occurs when the virtual address is decoded and found not to be an address for an OCA module. Under these conditions, the CW4010 reads from the Dcache, requests an SCbus read transaction, and then writes data to the Dcache Write Buffer or to a four-deep external write buffer. OCAcceptp is the only signal that determines whether an OCA transaction will take place.

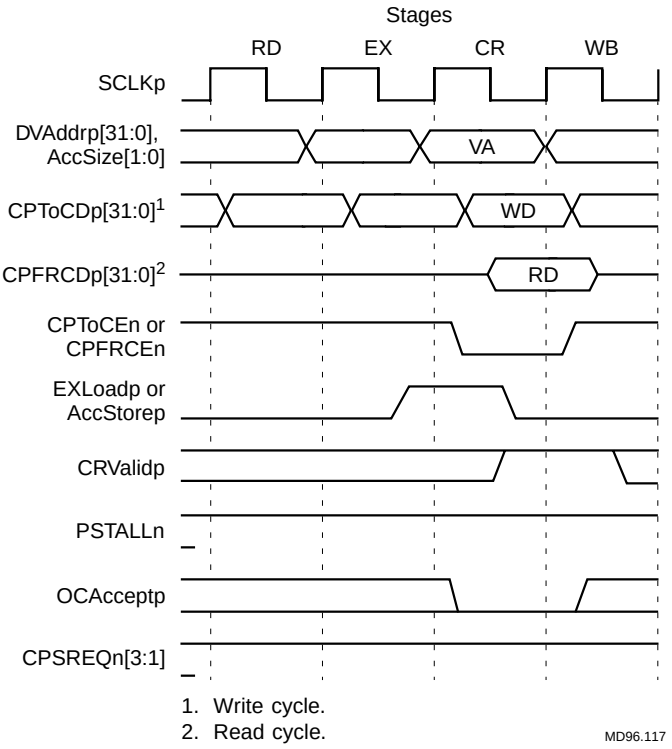
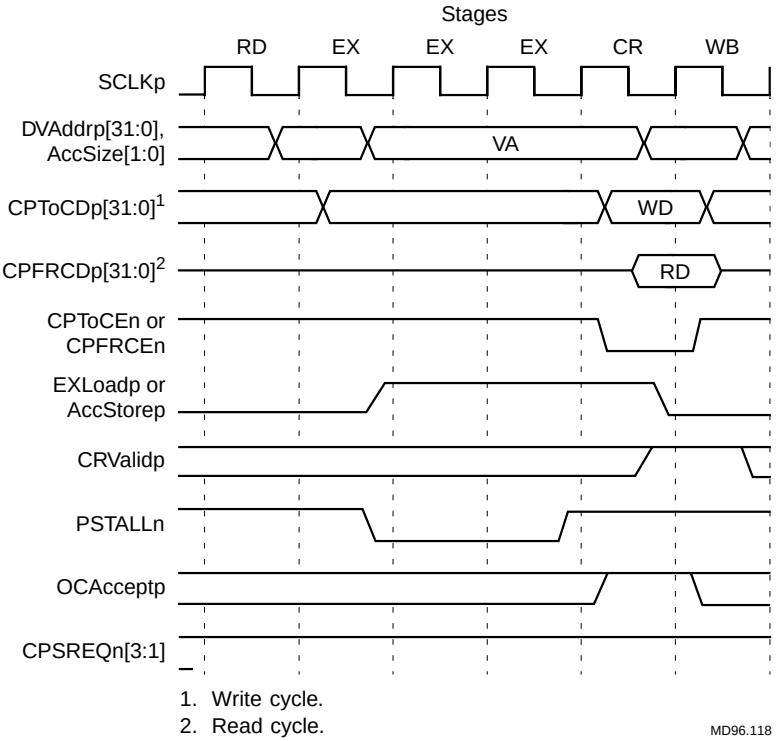


Figure 8.24 OCAbus Transaction Rejected by Address Decoder

8.3.3 OCAbus Access with Stall at EX Stage

Figure 8.25 shows an example where PSTALLn is asserted at the EX stage of the CW4010 pipeline, causing all pipeline stages to enter a stall state. When this happens, DVAddrp[31:0], AccSize[1:o], EXLoadp or AccStorep are held during the stall cycles.

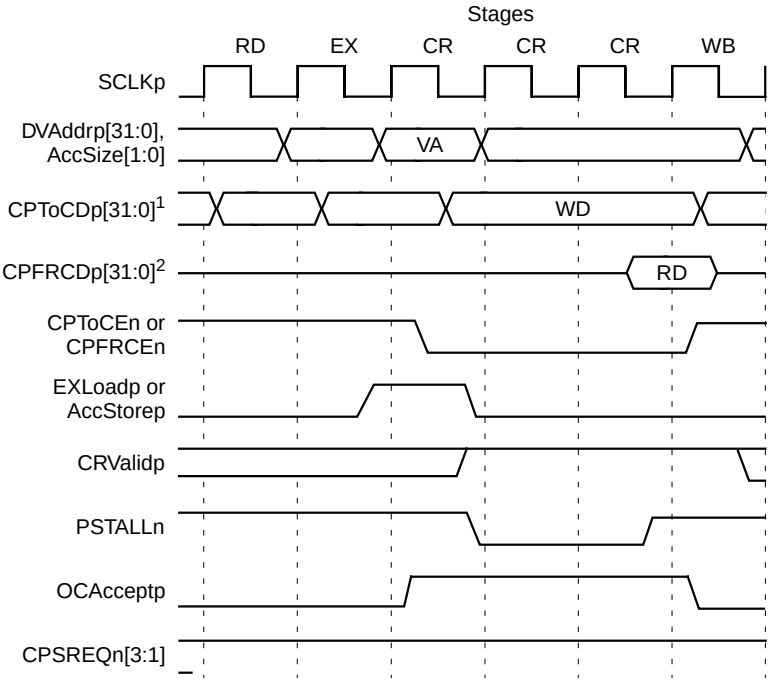
Figure 8.25
OCAbus with Stall at EX Stage



8.3.4 OCAbus Access with Stall at CR Stage

Figure 8.26 shows an example where PSTALLn is asserted at the CR stage of the CW4010 pipeline causing all pipeline stages to enter a stall state. When this happens, data on the CPToCDp bus, CRValidp, and OCAcceptp are held.

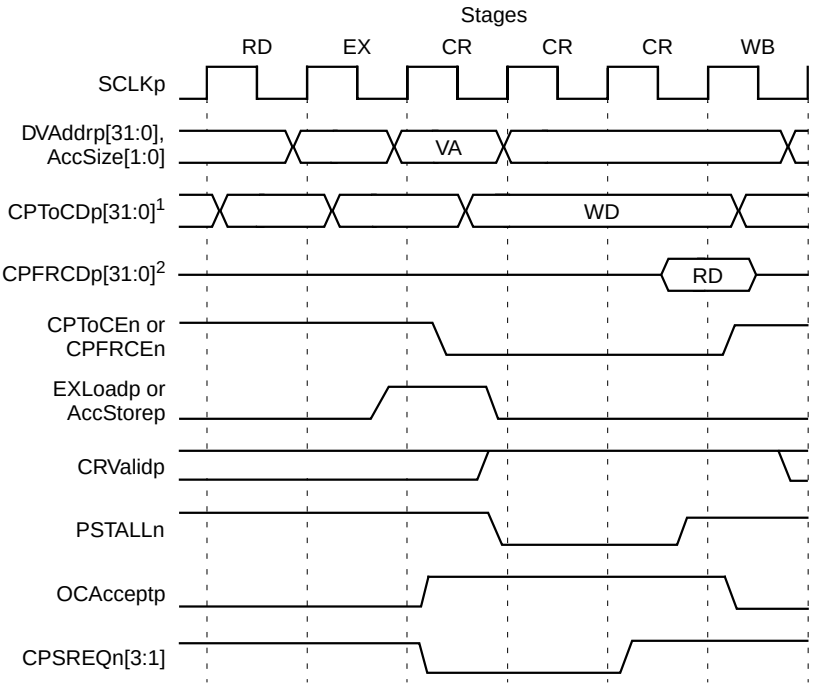
Figure 8.26 OCAbus Access with Stall at CR Stage



8.3.5 OCAbus Access with Stall Request

Figure 8.27 shows an example where the OCA bus device needs to insert some wait cycles before a read or write operation. To request a pipeline stall, the processor asserts CPSREQn from the beginning of the CR stage and this causes PSTALLn to be asserted. CPSREQn must be asserted and deasserted early in the clock cycle, since it is one of the critical path signals.

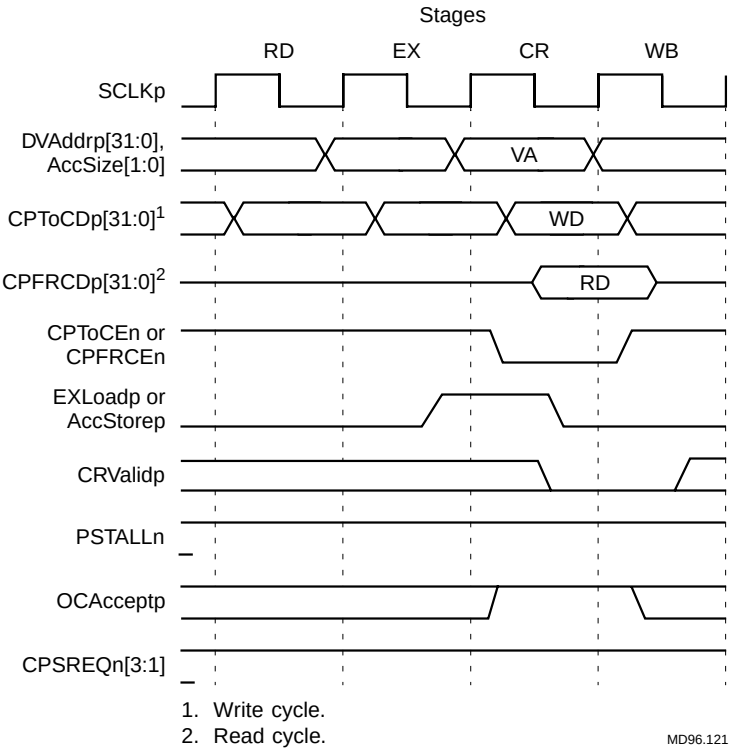
Figure 8.27 OCAbus Access with Stall Request



8.3.6 OCAbus Access with Pipeline Cancel

Figure 8.28 shows an example where a load or store instruction is cancelled by an exception. The exception is indicated when CRValidp is deasserted. When this happens, the write data must not be written into the OCA module. The read data being transferred to the CW4010 core is ignored. The cancelled load or store instruction may be executed later.

Figure 8.28 OCAbus Access with Pipeline Cancel



8.4 Cache Interface Behavior

When an external bus master writes data into main memory, it can invalidate the Dcache and Icache lines to maintain coherency between the main memory and the caches. The CW4010 has three signals to support this function. They are:

- CiNVAp[31:5] Cache Invalidate Address Bus bits
- DCiNVSn Dcache Invalidate Strobe
- ICiNVSn Icache Invalidate Strobe

When DCiNVSn or ICiNVSn is asserted, the address on the CiNVAp bus is latched and the CW4010 starts an invalidation process. DCiNVSn or ICiNVSn should be asserted for only one clock cycle. The Dcache or Icache line is invalidated when the cache physical address tag, whose line is valid, is coincident with the latched invalidate address. Both the V bit and the WB bit are cleared.

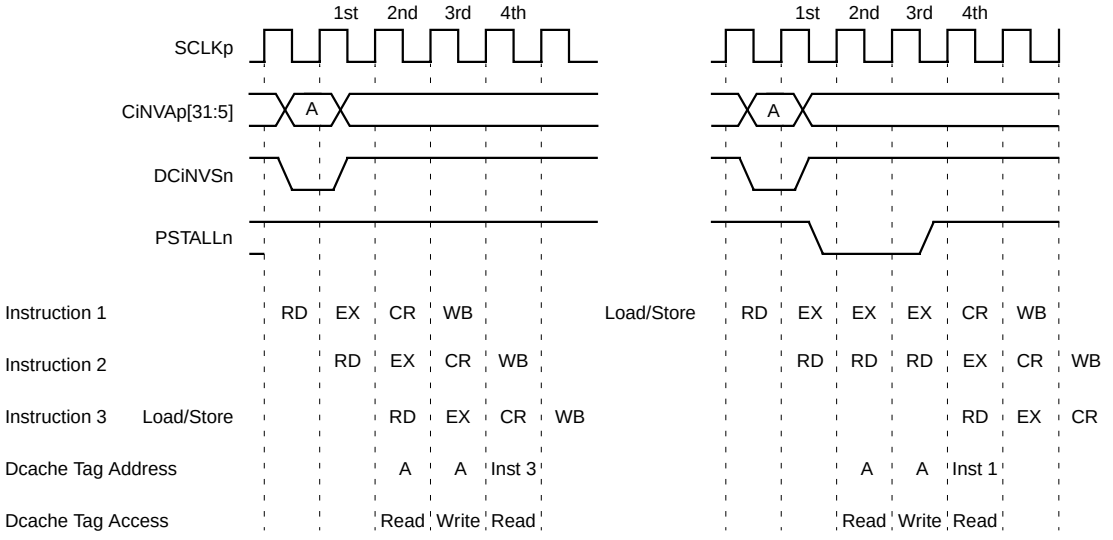
Figure 8.29 shows the timing diagram for Dcache invalidation implemented by bus snooping. In the first clock cycle after DCiNVSn is asserted, the Load Store Unit asserts the stall request signal if the EX stage is a load/store instruction.

In the second cycle, the Dcache Tag is read from the Dcache and compared with the address of the latched DCiNVAp bus. If they match and the EX stage is a load/store instruction, the pipeline stall request is asserted. To avoid timing problems, PSTALLn may not be deasserted during the second cycle.

In the third cycle, the V bit and WB bit of the Dcache line are cleared and the line is invalidated. If the addresses do not match at the third cycle, the Dcache is not accessed.

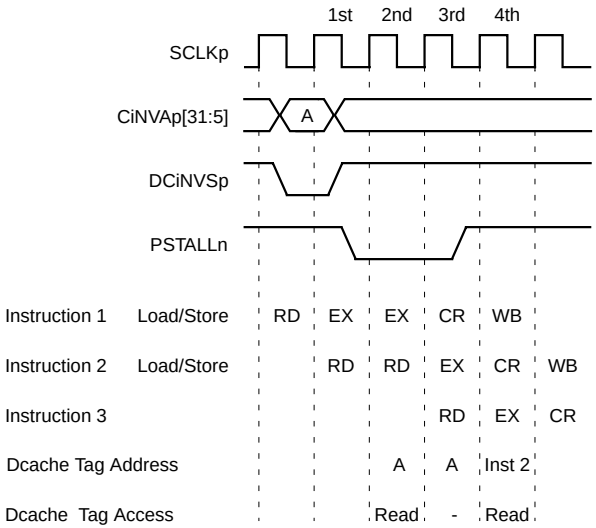
The stall cycle signal PSTALLn is asserted at the third clock cycle even if the address and Dcache tag do not match and the valid bit is not cleared.

Figure 8.29
Dcache Invalidation by
Snooping



Case 1. No Stall Cycle

Case 2. Two Stall Cycles



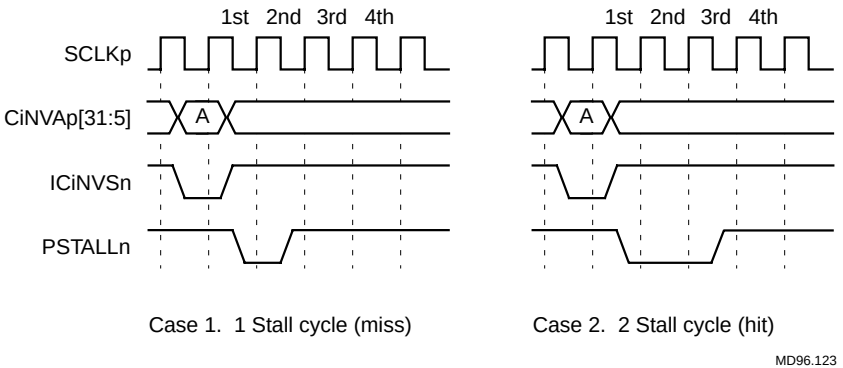
Case 3. One Stall Cycle (No Invalidation)

MD96.122

The Load Store Unit does not do anything if the external bus master read data from main memory and the address is dirty-cached by the CW4010. In this case, you may use WriteThrough mode for the page.

Figure 8.30 shows timing for Icache invalidation brought about by bus snooping. It needs a two-cycle stall if the invalidation address hits the tag or a one-cycle stall if it does not hit the tag.

Figure 8.30
Icache Invalidation by Snooping



8.5 Coprocessor Interface Behavior

The CW4010 supports up to four coprocessors, including an internal system coprocessor assigned as Coprocessor-0 (CP0). Coprocessor-1 is usually a floating point coprocessor. Coprocessors-2 and -3 are user-defined options.

The coprocessor interface has three 32-bit buses. The first bus is for an instruction code, the second one for data movement to a coprocessor, and the third for data movement from a coprocessor. The buses are common to all coprocessors.

Other control signals are divided into two categories: Coprocessor private signals, and Coprocessor common signals.

There are two kinds of coprocessor instructions: coprocessor operation instructions and data movement instructions. The CW4010 outputs an instruction code on the instruction code bus and asserts an execution signal at an RD stage for both types of instructions. The CW4010 executes only one coprocessor instruction even if a pair of instructions has no conflict. A coprocessor data movement instruction and a Load/Store instruction are not executed at the same time.

A coprocessor receives a data movement instruction at the RD stage, but data is moved at the CR stage, resulting in a one-delay slot for coprocessor data movement instructions. Since the NOP code is usually a SHIFT instruction, a coprocessor and a coprocessor instruction are executed in the same clock cycle. To have a one-delay slot between two coprocessor operations, three NOP codes must be inserted. Otherwise, each coprocessor needs to have *data bypassing* for the CR to EX stage or WB to CR stage (WB to CR gives better results).

If a coprocessor needs to support an LWCz (Load Word from Coprocessor) instruction, a Dcache miss may occur. The coprocessor cache-miss signal is asserted at the CR stage of the LWCz instruction and the pipeline enters a stall condition until valid data is read from the SCbus. During the stall cycles, correct data is placed on the data bus with the fixup cycle signal assertion. For simple coprocessor register access, the LWCz should not be used to perform memory data accesses. An LW (Load Word) instruction and a subsequent MTCz (Move to Coprocessor) can accomplish the same operation.

Coprocessor pipeline stages must enter a stall state when the CW4010 enters a stall state if the coprocessor is executing an instruction in the EX or CR stage. After the WB stage cycle of the CW4010 pipeline stage, the coprocessor does not need to enter stall because it is no longer cancelled.

Instruction executions must be cancelled if the instructions are in the EX or CR stage and the pipeline cancel signal is asserted. In this case, the Even/Odd slot instruction must be issued.

If the coprocessor cannot accept subsequent instructions for a time after the EX stage, the coprocessor channel CPBUSYn[3:1] signal should be asserted at the EX stage. The next coprocessor instruction is not started until CPBUSYn[3:1] is deasserted. If a coprocessor asserts CPSREQn[3:1] (pipeline stall request), the CW4010 enters the stall condition immediately until CPSREQn[3:1] is deasserted.

Coprocessor related instructions that include CP0 instructions have exclusive instruction bits in the opcode field. Bits [31:28] of the opcode field contain 0100_2, 1100_2, or 1110_2. Bits [27:26] define the coprocessor number (0 to 3).

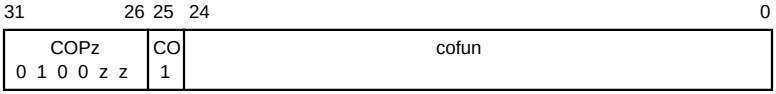
8.5.1 Coprocessor Functional Instruction

There is only one coprocessor operation instruction. As shown in Figure 8.31, 25 bits of the coprocessor function field define coprocessor operations.

Figure 8.31
Coprocessor Functional Instruction

COPz

Coprocessor-z Operation



8.5.2 Data Movement to and from CPU General Purpose Register Instructions

Each coprocessor (apart from CP0) can have up to 32 general-purpose registers and 32 control registers. CP0 does not have any control registers. Data in a CPU general purpose register selected by the rt field can be moved from or to a coprocessor register selected by the rd field and by bit 22, which selects between the general purpose and control register groups. The rt field is always for a CPU register and the rd field for a coprocessor register, regardless of the direction of data movement. Figure 8.32 shows the instructions used for data movement.

Figure 8.32
CPU General Purpose Register Data Movement Instructions

MFCz rt, rd

GPR [rt] ← CPR [rd]

MTCz rt, rd

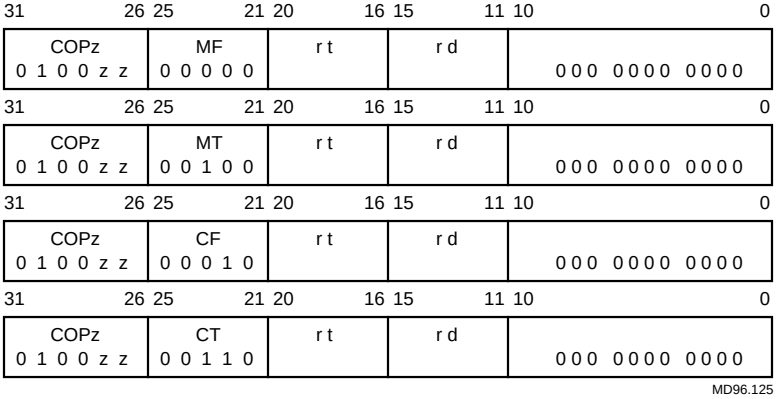
CPR [rd] ← GPR [rt]

CFCz rt, rd

GPR [rt] ← CCR [rd]

CTCz rt, rd

CCR [rd] ← GPR [rt]



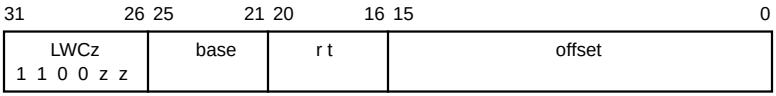
8.5.3 Instructions Moving Data from or to Memory

Coprocessor general purpose registers (not control registers), can move data directly to or from memory. The data is accessed through the Dcache if the address is in a cacheable area and it achieves a cache hit. In addition, the effective virtual address is translated to a physical address if the system supports an MMU. Instructions for data movement, shown in Figure 8.33 are for Coprocessors-1, -2, and -3 (CP1, CP2, and CP3) only. They are not supported for CP0.

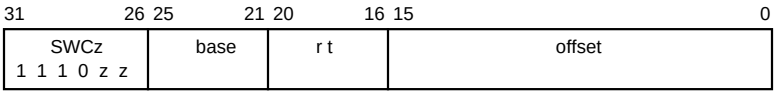
Because the CW4010 Load/Store Unit accesses the Dcache and JTLB, there is a possibility of a Dcache miss and a JTLB miss. When a Dcache miss occurs for LWCz, the pipeline enters a stall condition until the data is available from the SCbus. When a JTLB miss occurs, the pipeline cancel signal is asserted, and the coprocessor has to cancel the load operation at the CR stage. The CW4010 cancels the pipeline stages for other reasons.

Figure 8.33
Memory Data
Movement Instructions

LWCz rt, offset(base)
CPR [rt]
← mem(offset + GPR[base])



SWCz rt, offset(base)
CPR [rt]
→ mem(offset + GPR[base])



MD96.126

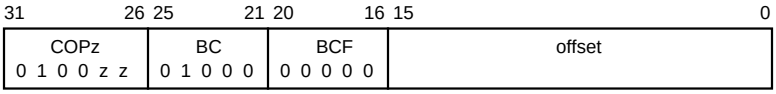
8.5.4 Branch on Coprocessor Condition Instructions

The CW4010 can branch to a target address, depending on the coprocessor condition input. The coprocessor interface has condition inputs for each coprocessor, and the branch unit examines these inputs. No coprocessor execution strobes are asserted.

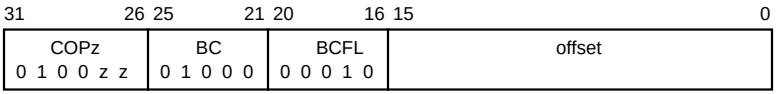
Both false and true condition branch instruction sets are defined. The MIPS-II instruction set has a non-branch slot type called *likely*. However, branching of MIPS-I instructions always requires a branch slot. The instructions associated with branching on coprocessor conditions are shown in Figure 8.34.

Figure 8.34
Branch on Coprocessor Condition Instructions

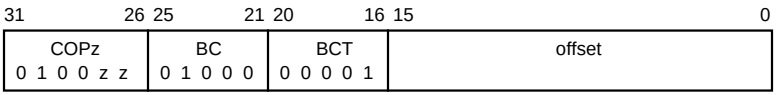
BCzF offset
If CPCoNDp = Low (False)
then PC ← PC+offset



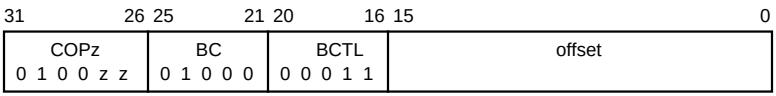
BCzFL offset
If CPCoNDp = Low (False)
then PC ← PC+offset



BCzT offset
If CPCoNDp = High (True)
then PC ← PC+offset



BCzTL offset
If CPCoNDp = High (True)
then PC ← PC+offset



MD96.127

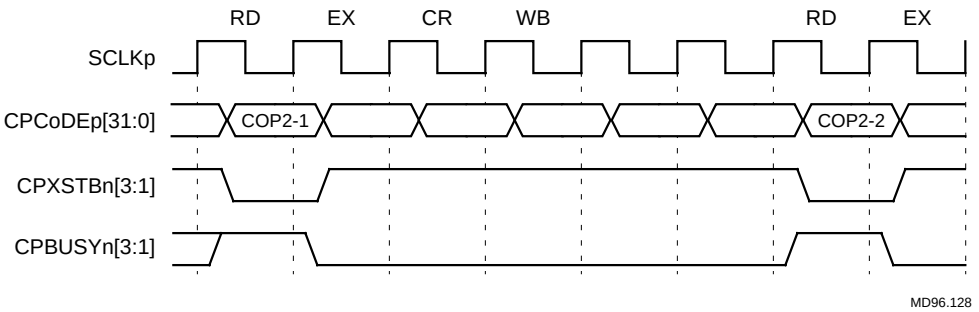
8.5.5 Coprocessor Operation (COPz)

When a coprocessor instruction is in the RD stage, the CW4010 coprocessor interface asserts one of the execution strobes, CPXSTBn[3:1], with a valid instruction code on CPCoDEp[31:0]. The result of the operation must be written at the WB stage or later in the coprocessor pipeline, because the instruction in the EX and CR stages may be cancelled by an exception. If the cancel signal, PCANCRn, is asserted before the WB stage of a coprocessor instruction, the instruction operation must be cancelled.

The coprocessor busy signal must be asserted if the operating coprocessor cannot accept another instruction after the first instruction begins executing. It is not necessary for the coprocessor to enter the stall condition when PSTALLn is asserted, except to keep the pipeline flow of the EX, CR, and WB stages. If PSTALLn is asserted before the WB stage and during data movement, the coprocessor pipeline must enter a stall cycle.

Figure 8.35 is a timing diagram for a normal COPz instruction execution. The busy signal may be asserted. The next CPXSTBn[3:1] is not asserted if the same coprocessor CPBUSYn[3:1] is asserted. If CPBUSYn[3:1] is not asserted, the execution strobe CPXSTBn[3:1] of the same coprocessor is asserted continuously to ensure continuous execution of coprocessor instructions. In the example shown in Figure 8.35, the second coprocessor instruction is placed in a **slip condition** by the busy signal.

Figure 8.35
COPz Execution



Instead of asserting CPBUSYn[3:1], a coprocessor can assert a pipeline stall request signal, CPSREQn[3:1], when the coprocessor cannot continue execution according to the results of instruction decoding.

CPSREQn[3:1] can assert the EX or CR stage, but CW4010 needs enough setup time for the stall request assertion/deassertion because it is a critical path. If a coprocessor asserts CPSREQn[3:1] at the CR stage, there may be another coprocessor instruction in the EX stage. When CPSREQn[3:1] Is asserted, PSTALLn is also immediately asserted. Similarly, when CPSREQn[3:1] is deasserted, PSTALLn is also immediately deasserted. The EX, CR, and WB stages of all execution units, including all coprocessors, enter a stall condition.

8.5.6 Data Movement to and from CPU Registers

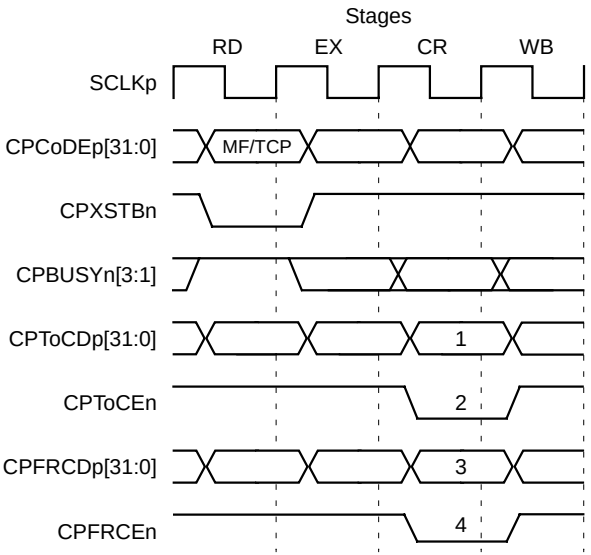
The coprocessor interface asserts one of the execution strobe signals, CPXSTBn[3:1], with a valid instruction code on the CPCoDEp[31:0] lines. At the CR stage, CPToCEn is asserted with valid data on the CPToCDp[31:0] bus when a MTCz or CTCz instruction is executed. In the case of a MFCz or CFCz instruction, CPFRCEn is asserted, then the coprocessor outputs data on the CPFRCDp[31:0] at the CR stage.

Coprocessor Data movement instructions to or from CPU registers always have one delay slot even if the busy signal is not asserted at the EX stage. When moving data from a coprocessor, data is valid at the CR stage and not valid at the EX stage. When moving data to a coprocessor, the CW4010 assumes the written data in the coprocessor is valid from the end of the CR stage, but it must be written into a coprocessor register at the WB stage.

When the assertion of the PSTALLn signal indicates a stall, the coprocessor pipeline stage from the EX and CR stages must stall in the same manner as the CW4010 core pipeline stage.

Figure 8.36 shows timing for a data movement instruction, with data transferring to and from the coprocessor. One data bus enable signal is asserted for each cycle.

Figure 8.36
Data Movement to/from CPU Registers Without Stall Cycles



1. Data to coprocessor.
2. Move to coprocessor.
3. Data from coprocessor.
4. Move from coprocessor.

MD96.129

Figure 8.37 shows the timing for the assertion of the stall signal at the EX stage. The coprocessor pipeline is in a stall state.

Figure 8.37
Data Movement to/from CPU Registers With Stall Cycles at EX Stage

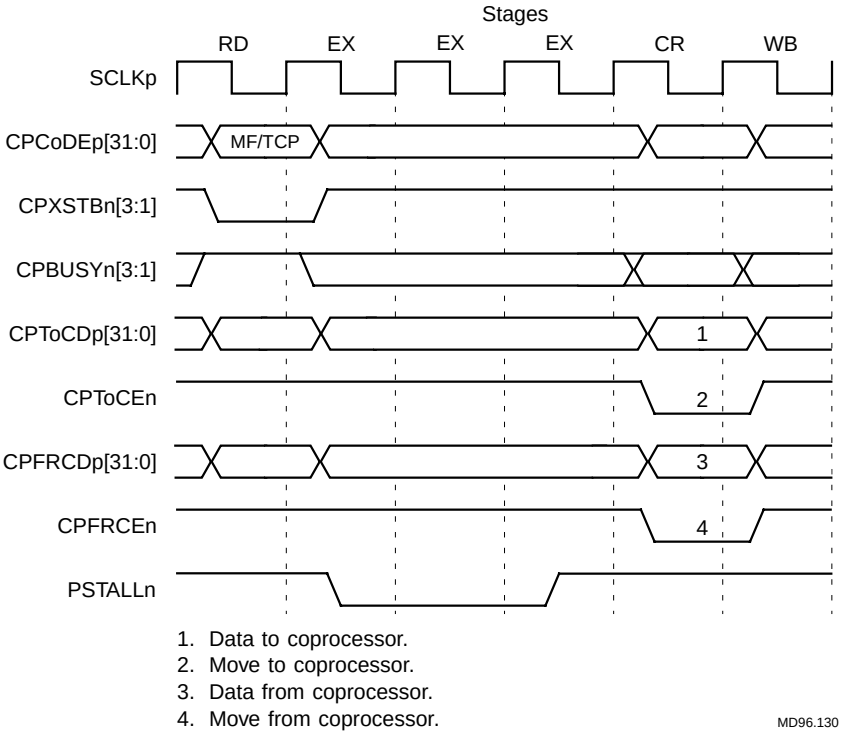
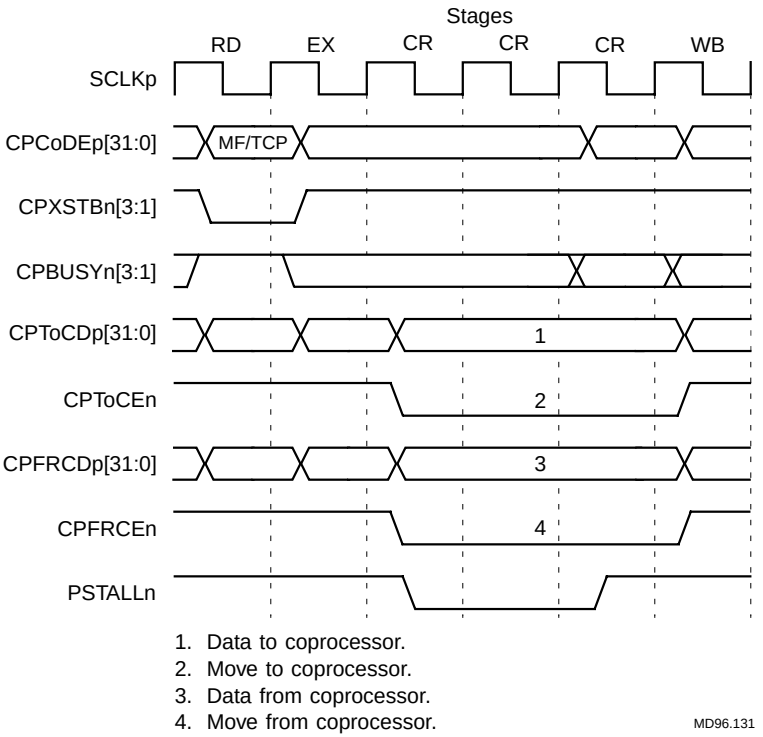


Figure 8.38 shows the timing for a stall signal assertion at the CR stage. The data to the coprocessor on the CPToCDp[31:0] lines is valid at the clock edge between the CR and WB stages. The coprocessor must output valid data at the last CR stage when the stall condition is resolved.

Figure 8.38
Data Movement to/from CPU Registers With Stall Cycles at CR Stage



8.5.7 Data Movement to or from Memories

The LWCz instruction moves data directly from memory to a coprocessor register. The SWCz instruction moves data from the coprocessor register to memory. The CW4010 accesses the Dcache if the address is cacheable. The CW4010 also accesses the JTLB if the address is mappable.

JTLB hit/miss is checked at the CR stage. In the case of a TLB miss, the pipeline is cancelled at the CR stage for both LWCz and SWCz. The data from memory must not be written into a coprocessor register until the WB stage; the pipeline is cancelled at the CR stage because of a JTLB miss.

In the case of an LWCz instruction, the read data from Dcache is output on the CPToCDp[31:0] lines when CPToCEn is asserted at the CR stage. In the same cycle, Dcache hit/miss is examined. If there is a cache hit, the data is written into the coprocessor register at the WB stage. If there is a cache miss, the coprocessor cache miss signal, CPMiSSn, is asserted at the CR stage and the pipeline stall signal, PSTALLn, is asserted from the WB stage. The pipeline then enters a stall state. When

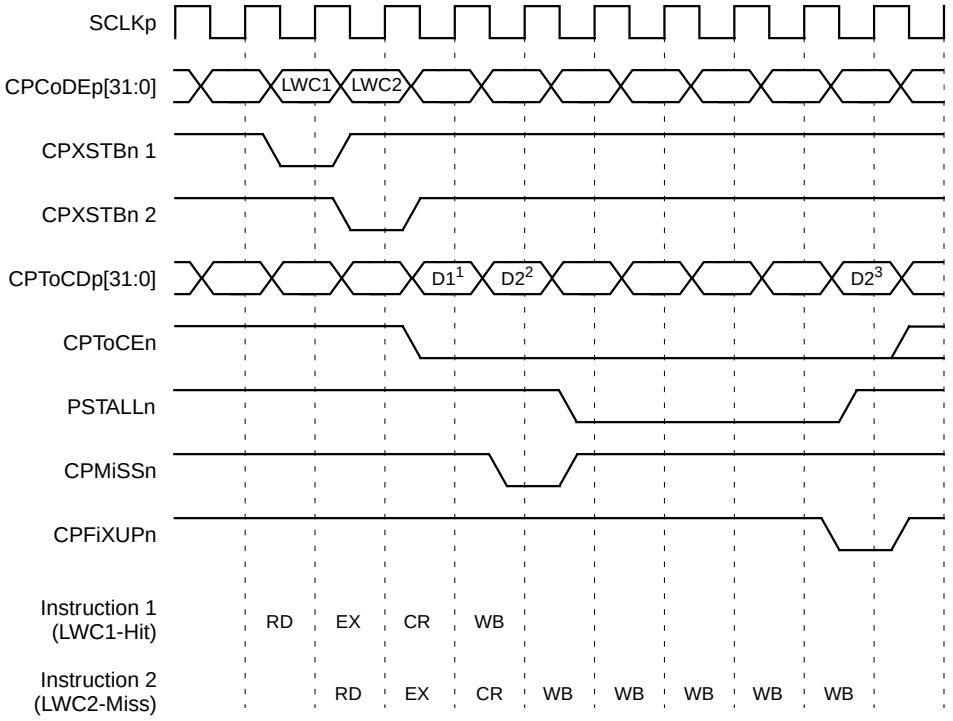
data is read from the memory, the data is output on the data bus CPToCDp[31:0] lines with the fixup cycle signal CPFiXUPn asserted.

In the case of an SWCz instruction, the coprocessor needs to output data on the CPFRCDp[31:0] lines at the CR stage. The timing is also indicated by the assertion of CPFRCEn. Dcache hit/miss is checked at the same CR stage. If there is a cache hit, data is stored in the Dcache. If there is a cache miss, data is stored in the external write buffer in the Load Store Unit. The CW4010 then generates a write transaction on the SCbus.

Data transferred from the coprocessor interface to a coprocessor at the CR stage for an LWCz instruction needs at least one delay slot. If the data is not valid until the WB stage and the coprocessor does not have register bypassing hardware like the CW4010 CPU, the busy signal should be asserted to stop the next coprocessor operation.

Figure 8.39 shows timing for the LWCz instruction Dcache hit and miss. The first LWCz instruction hits the Dcache, but the second instruction misses.

Figure 8.39
LWCz Dcache-Hit and Miss

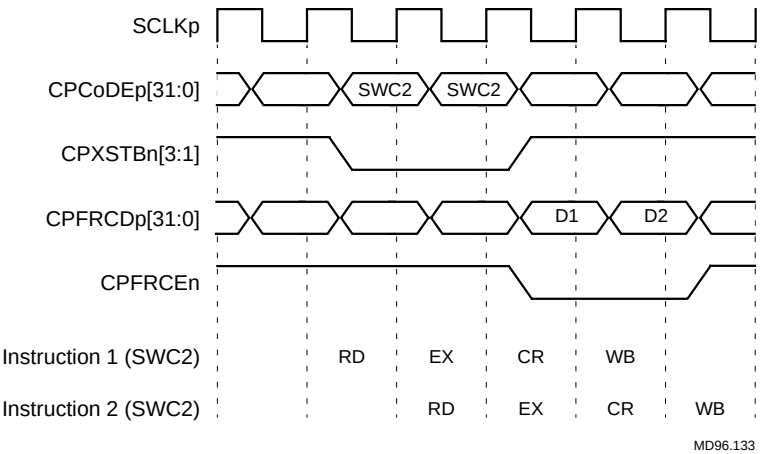


1. Dcache hit.
2. Dcache miss.
3. Valid data for instruction 2.

MD96.132

Figure 8.40 shows the timing diagram for two SWCz instructions. There is no difference in the coprocessor interface signals for a Dcache Hit or Dcache Miss.

Figure 8.40
SWCz Timing



8.5.8 Coprocessor Conditions

Coprocessor conditions are sampled on the rising edge of the clock at the end of the EX stage. Condition inputs are active-HIGH and must be synchronized with the system clock.

8.5.9 Even/Odd Slot and Pipeline Cancel

When an exception occurs, the CW4010 cancels pipeline stages at the CR stage. If a cancel occurs, the pipelines of the coprocessors must also be cancelled. Since the CW4010 executes two instructions in even and odd slots, it must find out whether the cause of the exception is in an even slot or odd slot. If the even slot is the cause of the exception, both even and odd instructions in the CR stage must be cancelled. If the odd slot is the cause, the even instruction in the CR stage must be completed and the odd instruction must be cancelled. Instructions in the EX stage even and odd slots must be cancelled.

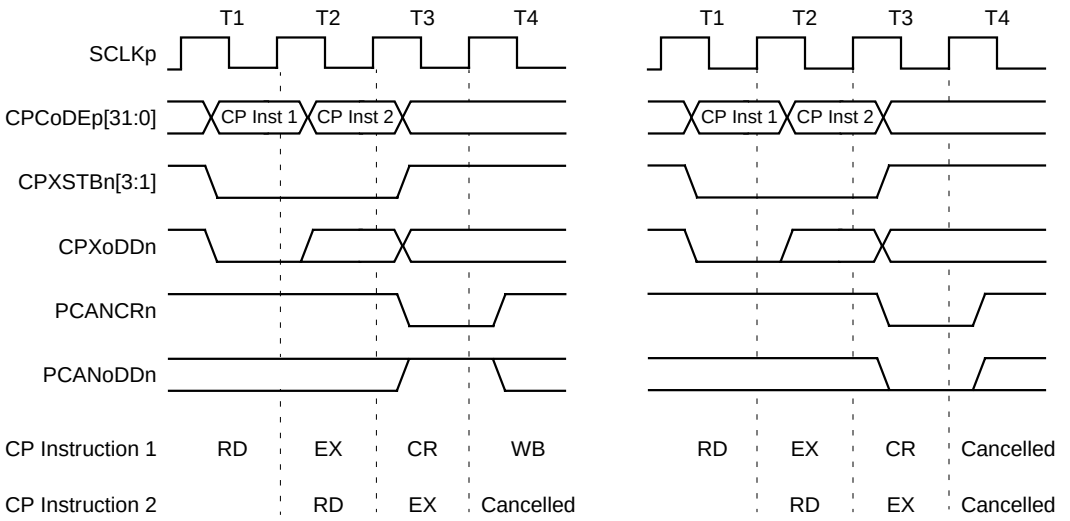
Coprocessors must know whether an executing instruction is in an even or odd slot at the RD stage. At the RD stage, the CW4010 issues only one coprocessor instruction even if there are two coprocessor instructions in even and odd slots. When the coprocessor execution signal, CPXSTBn[3:1], is asserted, the coprocessor interface outputs the CPXoDDn signal, which indicates that this instruction is in an even or odd slot. The coprocessor needs to keep this information in the EX and CR pipeline stages.

When the pipeline cancel signal, PCANCRn, is asserted, PCANoDDn is valid, which indicates that the even instruction at the CR stage is the cause of the cancellation or that the odd instruction is the cause. Coprocessors have to compare the PCANoDDn signal and the information kept at the CR pipeline stage. If the instruction in a coprocessor CR stage is an even instruction (CPXoDDn HIGH at RD) and PCANoDDn is asserted LOW (odd slot), the instruction must be completed (go to the WB stage). Otherwise, the instruction must be cancelled. Instructions in the EX stage must be cancelled.

The CW4010 does not cancel instructions in the WB stage or later cycles.

Figure 8.41 shows an example of even/odd slot and pipeline cancellation. Part (a) shows the case in which the instruction in the CR stage is not cancelled and part (b) shows the case where the instruction is cancelling.

Figure 8.41
Even/Odd Slot and
Pipeline Cancel



Case 1. Instruction in CR stage is not cancelled. Case 2. Instruction in CR stage is cancelled

MD96.134

8.5.10 BranchLikely in Even Slot is False

The MIPS-II instruction set adds the BranchLikely instructions. The instruction in the branch delay slot is executed only when the branch condition is true. If the branch condition is false, the delay slot instruction must be nullified. In the case of a normal branch instruction, the instruction in the branch delay slot is executed regardless of the branch condition.

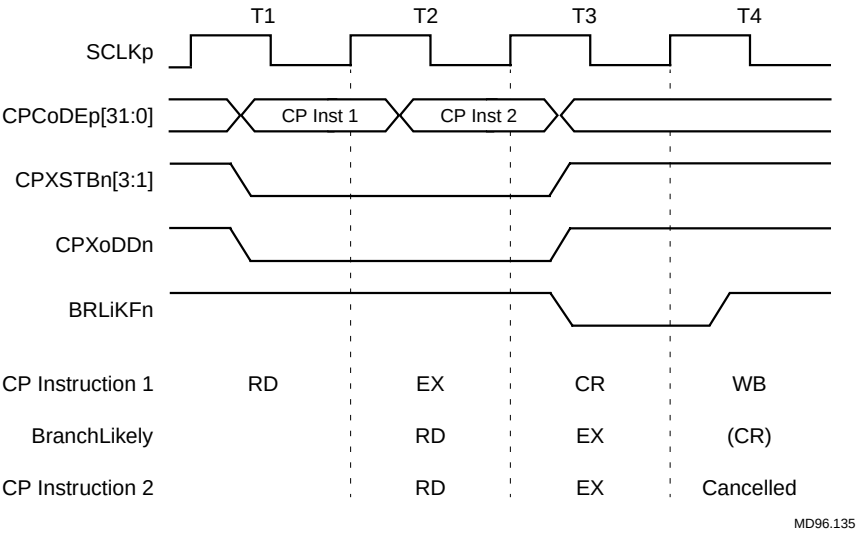
Because the CW4010 tries to execute two instructions in one clock cycle, the instruction in a branch delay slot is executed at the same time as a BranchLikely instruction, if the BranchLikely instruction is in an even slot. If the branch condition is false, the delay slot instruction, which is in the odd slot, must be cancelled.

The coprocessor interface provides a signal, BRLiKFn, to cancel the instruction that is in a delay slot of a BranchLikely instruction. This type of instruction resides in an even slot. If BRLiKFn is asserted, coprocessors need to cancel any instruction in the EX stage. Although coprocessors keep information about even and odd slots so that they can cancel the instruction at the CR stage correctly, this information is not needed to examine the even and odd slots for BRLiKFn.

If a BranchLikely is in an odd slot and it is false, BRLiKFn is not asserted because the execution strobe of the branch delay slot is not asserted at the RD stage.

Figure 8.42 shows an example of BFLiKFn assertion. Assuming that CP Inst1, BranchLikely, and CP Inst2 are fetched continuously and that the BranchLikely instruction is in an even slot, if the branch is not taken, CP Inst2 must not be executed, and must be cancelled in the coprocessor according to the assertion of BFLiKFn.

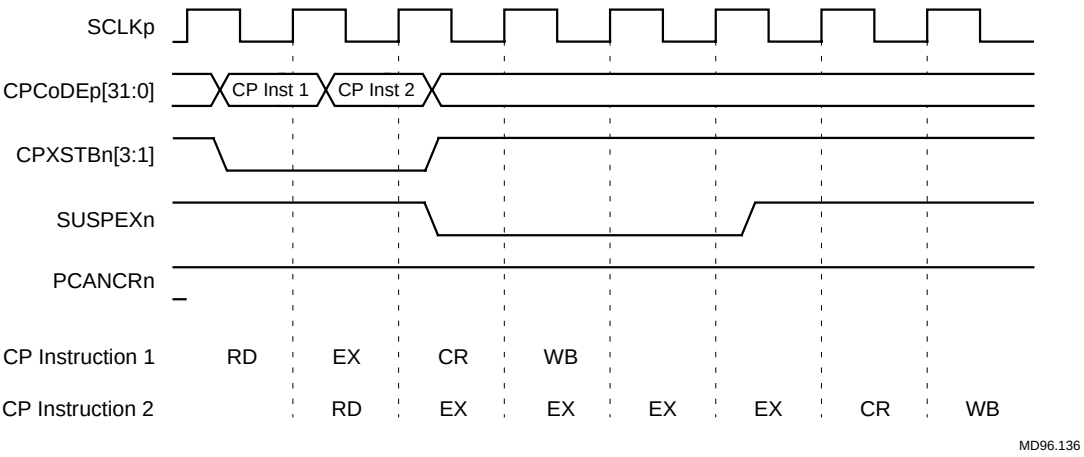
Figure 8.42
BranchLikely in
Even Slot is False



8.5.11 EX Stage Suspension

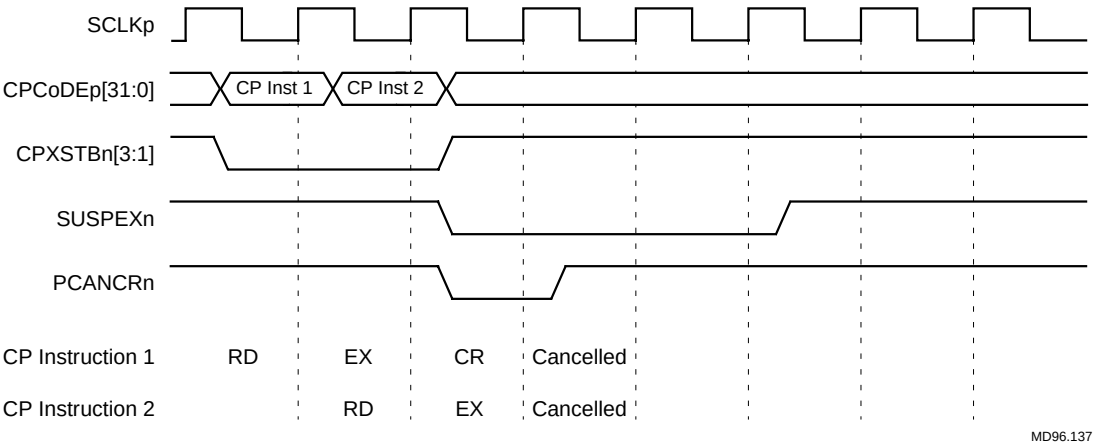
The CW4010 may request that the EX stage of a coprocessor be suspended when an Icache miss occurs just after a CP instruction is executed. If the suspension signal, SUSPEXn, is asserted and there is a valid instruction in the EX stage of the coprocessor, the EX stage must be suspended until SUSPEXn is deasserted. Instructions in other pipeline stages should not be stopped. Figure 8.43 shows an example where two coprocessor instructions are executed continuously, but the second one is suspended at the EX stage.

Figure 8.43
EX Stage Suspension



If the pipeline is cancelled by asserting PCANCRn, the instruction in the EX stage must be cancelled even if SUSPEXn is asserted on the same clock cycle, as shown in Figure 8.44.

Figure 8.44
EX Stage Suspension
(cancelled)



8.5.12 Floating-Point Unit Exception

The coprocessor interface has an exception input, FPERRXn, assigned as a Floating-Point Unit (FPU) exception. FPERRXn is sampled at the EX stage and issued at the CR stage. It has its own exception code (15) in the Cause Register of CP0. It can also be used as a user-defined coprocessor exception input if necessary. Although an FPU is usually assigned to CP1, the FPU exception is not related to the coprocessor number.

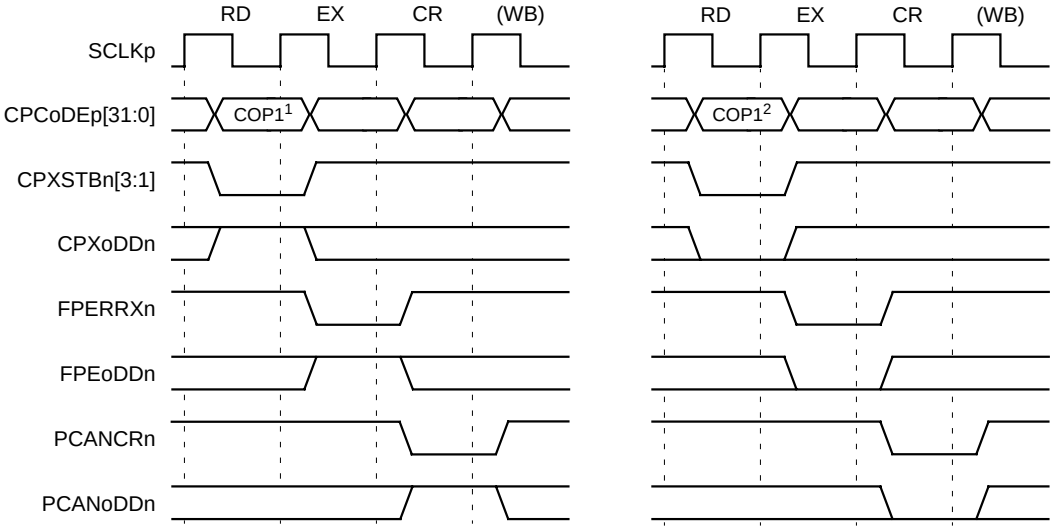
FPEoDDn is the coprocessor interface input for the FPU exception and is issued when FPERRXn is asserted at the EX stage.

When a coprocessor instruction is started at an RD stage, CPXoDDn indicates whether the instruction is in the even slot (CPXoDDn HIGH) or the odd slot (CPXoDDn LOW). The signal must be held in the EX and CR stages of the coprocessor pipeline so that the pipeline may be cancelled correctly. In addition, if the FPERRXn signal is asserted in the EX stage of the coprocessor instruction, FPEoDDn must be driven according to the even/odd status of the EX stage. If FPEoDDn is HIGH (Even) when FPERRXn is asserted, the CW4010 cancels instructions in both even and odd slots of the CR stage. If FPEoDDn is LOW (Odd), only the odd instruction is cancelled.

If FPERRXn is used for an user-defined coprocessor exception and is not asserted at the EX stage, the pipeline is not cancelled precisely and is treated in the same way as an interrupt exception if the FPEoDDn is strapped HIGH.

Figure 8.45 shows timing for a coprocessor instruction start, FPU exception, and pipeline cancel.

Figure 8.45
FPU Exception and
Pipeline Cancel Timing



1. COP1: Coprocessor instruction is in the Even slot.
2. COP1: Coprocessor instruction is in the Odd slot.

MD96.138

Chapter 9
Specifications

The CW4010 core specifications are available in an addendum entitled *MiniRISC™ CW4010 Superscalar Microprocessor Core Technical Manual Addendum*. The addendum includes information on AC timing, input and output loading, driver type, and power consumption.

Appendix A
Programmer's Notes

This appendix contains information that will be useful if you are writing software for the CW4010 core. The information is arranged in functional groups: instruction related, CP0 or TLB related, and cache related.

A.1 Instruction Related

The following are instruction related notes:

- The instruction prior to an ERET must not generate an exception. You can use a NOP (no operation) to make sure that this restriction is met.
- The WAITI instruction must be followed by at least one NOP.
- Trap instructions must not be placed in branch delay slots.

A.2 CP0 or TLB Related

The following are CP0 or TLB related notes:

- When the CW4010 is operating in R3000 exception compatibility mode, the RFE (Restore From Exception) instruction clears the LL (load linked) bit. This is consistent with R4000 mode ERET operation.
- If a TLB is not present or enabled in the system, CP0 will reflect a Coprocessor Unusable exception if an attempt is made to execute any of the TLB maintenance instructions: TBLP, TLBR, TLBWI, TLBWR.
- TLB instructions (TLBP, TLBR, TLBWI, TLBWR) cannot be preceded or followed by a data access instruction (load or store) that requires target address translation, that is kseg, kseg2.
- The instruction prior to a TLBWI or TLBWR instruction must not generate an exception. You can use a NOP to make sure that this restriction is met.

- Three instructions are required between a MTC0 instruction that targets any of the TLB support registers (that is, EntryHi, EntryLo, PageMask, and Index) and a subsequent TLBWI or TLBWR instruction. This ensures that the results of the prior MTC0 instruction will be seen by the TLB write operation
- Five instructions are required between a MTC0 Status register operation that updates the coprocessor usability field (Status[31:28]) and a subsequent coprocessor instruction that expects to see the updated value.
- Seven instructions are required between a MTC0 EPC register operation and a subsequent ERET instruction that expects to see the updated value.

A.3 Cache Related

The following is a Cache related note:

- When the CW4010 is operating in Isolate Cache mode, load and store operations to the cache are not allowed in the delay slot of BranchLikely instructions.

A.4 CW33300 Compatible Debug Extensions

The following is a CW33300 Compatible Debug Extension note:

- The existing CW33300 has some extensions to the CP0 that provide enhanced debugging and exception handler support. The CW4010 core remains compatible with these enhancements. Refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for further information.

Glossary

Big-endian This is a method of data formatting in which each field is addressed by referring to its most significant byte. This means that if you are accessing a four-byte, singleword, the most significant byte is byte 03, and the most significant bit is bit 31. See also little-endian.

Bus sizing Refers to the ability of the processor to support and interface with data buses of different sizes.

Bus snooping This is the method used by the cache controller to monitor memory accesses performed by other bus masters.

Direct-map caching In a direct-mapped cache, each memory location is mapped to one position in the cache. Direct mapping is useful if you are storing small loops and sequential operations. You can use this type of caching for both the Dcache and the Icache.

Encrypted Encrypted files are source code files that have been processed in a language such as HDL or Verilog so that they are only machine readable. This process enables you to have access to the behavior of the files but not to the intellectual property associated with them.

Fixup cycle This is a clock cycle during which the Load Miss data is funneled back to the instruction that requires it.

Little-endian This is a method of data formatting in which each field is addressed by referring to its least significant byte. This means that if you are accessing a four-byte, singleword, the most significant byte is byte 00, and the most significant bit is bit 00. The CW4010 supports little-endian format. See also big-endian.

Placement algorithms Information is placed in a cache using placement algorithms. These algorithms define the positions in the cache where the information from a particular memory location may be stored. The CW4010 uses two types of algorithm, direct mapping and two-way set associative mapping.

Slip condition A slip condition occurs when the pipeline stalls after the EX stage. In this situation, the previous instruction is executed and clears the pipeline. However, the earlier stages of the pipeline are stalled.

Two-way set associative caching In a two-way set associative cache, each memory location is stored in one of two possible positions. Two-way set associative mapping is well-suited for data references, which tend to be more scattered than instructions. You can use this type of caching for both the Dcache and the Icache.

Unencrypted Files that are unencrypted have not been subjected to the processing described in the entry Encrypted. These files are human readable and can be written using a text editor.

Verilog Model Verilog is an open standard language. A Verilog model represents a design in the language. It provides no indication of the level of abstraction.

Customer Feedback

We would appreciate your feedback on this document. Please copy the following page, add your comments, and fax it to us at the address on the following page.

If appropriate, please also fax copies of any marked-up pages from this document.

Important: Please include your name, phone number, fax number, and company address so that we may contact you directly for clarification or additional information.

Thank you for your help in improving the quality of our documents.

Reader's Comments

Fax your comments to:

LSI Logic Corporation
Technical Publications
M/S F-112
Fax: 408.433.4333

Please tell us how you rate this document: *MiniRISC CW4010 Superscalar Microprocessor Core Technical Manual.* Place a check mark in the appropriate blank for each category.

	Excellent	Good	Average	Fair	Poor
Completeness of information	____	____	____	____	____
Clarity of information	____	____	____	____	____
Ease of finding information	____	____	____	____	____
Technical content	____	____	____	____	____
Usefulness of examples and illustrations	____	____	____	____	____
Overall manual	____	____	____	____	____

What could we do to improve this document?

_____

_____

_____

_____

If you found errors in this document, please specify the error and page number. If appropriate, please fax a marked-up copy of the page(s).

_____

_____

_____

_____

Please complete the information below so that we may contact you directly for clarification or additional information.

Name _____ Date _____

Telephone _____ Fax _____

Title _____

Department _____ Mail Stop _____

Company Name _____

Street _____

City, State, Zip _____

U.S. Distributors by State

Alabama
Huntsville
Hamilton Hallmark
Tel: 800.633.2918

Wyle Electronics
Tel: 800.964.9953

Arizona
Phoenix
Hamilton Hallmark
Tel: 800.528.8471

Wyle Electronics
Tel: 602.804.7000

Tempe
Hamilton Hallmark
Tel: 602.414.7705

California
Culver City
Hamilton Hallmark
Tel: 310.558.2000

Irvine
Hamilton Hallmark
Tel: 714.789.4100

♦ Wyle Electronics
Tel: 714.789.9953

Los Angeles
Wyle Electronics
Tel: 818.880.9000

Rocklin
Hamilton Hallmark
Tel: 916.624.9781

Sacramento
Wyle Electronics
Tel: 916.638.5282

San Diego
Hamilton Hallmark
Tel: 619.571.7540

Wyle Electronics
Tel: 619.565.9171

San Jose
♦ Hamilton Hallmark
Tel: 408.435.3500

Santa Clara
Wyle Electronics
Tel: 408.727.2500

Woodland Hills
Hamilton Hallmark
Tel: 818.594.0404

Colorado
Colorado Springs
Hamilton Hallmark
Tel: 719.637.0055

Denver
♦ Wyle Electronics
Tel: 303.457.9953

Englewood
Hamilton Hallmark
Tel: 303.790.1662

Connecticut
Cheshire
Hamilton Hallmark
Tel: 203.271.2844

Florida
Fort Lauderdale
Hamilton Hallmark
Tel: 305.484.5482

Wyle Electronics
Tel: 305.420.0500

Largo
Hamilton Hallmark
Tel: 800.282.9350

Orlando
Wyle Electronics
Tel: 407.740.7450

Tampa/N. Florida
Wyle Electronics
Tel: 800.395.9953

Winter Park
Hamilton Hallmark
Tel: 407.657.3317

Georgia
Atlanta
Wyle Electronics
Tel: 800.876.9953

Duluth
Hamilton Hallmark
Tel: 800.241.8182

Illinois
Arlington Heights
♦ Hamilton Hallmark
Tel: 708.797.7300

Chicago
Wyle Electronics
Tel: 708.620.0969

Iowa
Carmel
Hamilton Hallmark
Tel: 800.829.0146

Kansas
Overland Park
Hamilton Hallmark
Tel: 800.332.4375

Kentucky
Lexington
Hamilton Hallmark
Tel: 800.235.6039

Maryland
Baltimore
Wyle Electronics
Tel: 410.312.4844

Columbia
Hamilton Hallmark
Tel: 800.638.5988

Massachusetts
Boston
♦ Wyle Electronics
Tel: 800.444.9953

Peabody
♦ Hamilton Hallmark
Tel: 508.532.3701

Michigan
Plymouth
Hamilton Hallmark
Tel: 313.416.5800

Minnesota
Bloomington
Hamilton Hallmark
Tel: 612.881.2600

Minneapolis
Wyle Electronics
Tel: 800.860.9953

Missouri
Earth City
Hamilton Hallmark
Tel: 314.291.5350

New Jersey
Mt. Laurel
Hamilton Hallmark
Tel: 609.222.6400

No. New Jersey
Wyle Electronics
Tel: 201.882.8358

Parsippany
Hamilton Hallmark
Tel: 201.515.1641

New Mexico
Alburquerque
Hamilton Hallmark
Tel: 505293.5119

New York
Hauppauge
Hamilton Hallmark
Tel: 516.737.7400

Long Island
Wyle Electronics
Tel: 516.293.8446

Rochester
Hamilton Hallmark
Tel: 800.462.6440

North Carolina
Raleigh
Hamilton Hallmark
Tel: 919.872.0712

Wyle Electronics
Tel: 919.469.1502

Ohio
Cleveland
Wyle Electronics
Tel: 216.248.9996

Dayton
Hamilton Hallmark
Tel: 800.423.4688

Wyle Electronics
Tel: 513.436.9953

Solon
Hamilton Hallmark
Tel: 216.498.1100

Toledo
Wyle Electronics
Tel: 419.861.2622

Worthington
Hamilton Hallmark
Tel: 614.888.3313

Oklahoma
Tulsa
Hamilton Hallmark
Tel: 918.254.6110

Oregon
Beaverton
Hamilton Hallmark
Tel: 503.526.6200

Portland
Wyle Electronics
Tel: 503.643.7900

Pennsylvania
Philadelphia
Wyle Electronics
Tel: 800.871.9953

Texas
Austin
Hamilton Hallmark
Tel: 512.258.8848

Wyle Electronics
Tel: 800.365.9953

Dallas
Hamilton Hallmark
Tel: 214.553.4302

Wyle Electronics
Tel: 800.955.9953

Houston
Hamilton Hallmark
Tel: 713.787.8300

Wyle Electronics
Tel: 713.784.9953

San Antonio
Wyle Electronics
Tel: 210.697.2816

Utah
Salt Lake City
Hamilton Hallmark
Tel: 801.266.2022

Wyle Electronics
Tel: 801.974.9953

Washington
Redmond
Hamilton Hallmark
Tel: 206.881.6697

Seattle
Wyle Electronics
Tel: 800.248.9953

Wisconsin
Milwaukee
Wyle Electronics
Tel: 800.867.9953

New Berlin
Hamilton Hallmark
Tel: 414.780.7200

♦ Dstributors with Design Resource Centers

Sales Offices and Design Resource Centers

LSI Logic Corporation
Corporate Headquarters
Tel: 408.433.8000
Fax: 408.433.8989

UNITED STATES

California
Irvine
♦ Tel: 714.553.5600
Fax: 714.474.8101

San Diego
Tel: 619.635.1300
Fax: 619.635.1350

Silicon Valley
Sales Office
Tel: 408.433.8000
Fax: 408.433.7783
Design Center
♦ Tel: 408.433.8000
Fax: 408.433.2820

Colorado
Boulder
Tel: 303.447.3800
Fax: 303.541.0641

Florida
Boca Raton
Tel: 407.395.6200
Fax: 407.394.2865

Georgia
Atlanta
Tel: 404.395.3800
Fax: 404.395.3811

Illinois
Schaumburg
♦ Tel: 708.995.1600
Fax: 708.995.1622

Kentucky
Bowling Green
Tel: 502.793.0010
Fax: 502.793.0040

Maryland
Bethesda
♦ Tel: 301.897.5800
Fax: 301.897.8389

Massachusetts
Waltham
♦ Tel: 617.890.0180
Fax: 617.890.6158

Minnesota
Minneapolis
♦ Tel: 612.921.8300
Fax: 612.921.8399

New Jersey
Edison
♦ Tel: 908.549.4500
Fax: 908.549.4802

New York
New York
Tel: 716.223.8820
Fax: 716.223.8822

North Carolina
Raleigh
Tel: 919.783.8833
Fax: 919.783.8909

Oregon
Beaverton
Tel: 503.645.9882
Fax: 503.645.6612

Texas
Austin
Tel: 512.388.7294
Fax: 512.388.4171

Dallas
♦ Tel: 214.788.2966
Fax: 214.233.9234

Houston
Tel: 713.379.7800
Fax: 713.379.7818

Washington
Bellevue
Tel: 206.822.4384
Fax: 206.827.2884

INTERNATIONAL

Australia
Reptechnic Pty Ltd
New South Wales
Tel: 612.9953.9844
Fax: 612.9953.9683

Canada
LSI Logic Corporation of Canada Inc
Alberta
Tel: 403.262.9292
Fax: 403.262.9494

Ontario
Ottawa
♦ Tel: 613.592.1263
Fax: 613.592.3253

Toronto
♦ Tel: 416.620.7400
Fax: 416.620.5005

Quebec
Pointe Claire
♦ Tel: 514.694.2417
Fax: 514.694.2699

France
LSI Logic S.A.
Paris
♦ Tel: 33.1.34.63.13.13
Fax: 33.1.34.63.13.19

Germany
LSI Logic GmbH
Munich
♦ Tel: 49.89.4.58.33.0
Fax: 49.89.4.58.33.108

Stuttgart
Tel: 49.711.13.96.90
Fax: 49.711.86.61.428

Hong Kong
AVT Industrial Ltd
Hong Kong
Tel: 852.2428.0888
Fax: 852.2401.2105

India
LogiCAD India Private Ltd
Bangalore
Tel: 91.80.526.2500
Fax: 91.80.338.6591

Israel
LSI Logic
Ramat Hashron
♦ Tel: 972.3.5.403741
Fax: 972.3.5.403747

Netanya
♦ Tel: 972.9.657190
Fax: 972.9.657194

Italy
LSI Logic S.P.A.
Milano
♦ Tel: 39.39.687371
Fax: 39.39.6057867

Japan
LSI Logic K.K.
Tokyo
♦ Tel: 81.3.5463.7821
Fax: 81.3.5463.7820

Osaka
♦ Tel: 81.6.947.5281
Fax: 81.6.947.5287

Korea
LSI Logic Corporation of Korea Ltd
Seoul
♦ Tel: 82.2.561.2921
Fax: 82.2.554.9327

Singapore
Desner Electronics Pte Ltd
Singapore
Tel: 65.285.1566
Fax: 65.284.9466

Electronic Resources Ltd
Tel: 65.298.0888
Fax: 65.298.1111

Spain
LSI Logic S.A.
Madrid
♦ Tel: 34.1.3672200
Fax: 34.1.3673151

Sweden
LSI Logic AB
Stockholm
♦ Tel: 46.8.7034680
Fax: 46.8.7506647

Switzerland
LSI Logic Sulzer AG
Brugg/Biel
Tel: 41.32.536363
Fax: 41.32.536367

Taiwan
LSI Logic Asia-Pacific Regional Office
Taipei
♦ Tel: 886.2.718.7828
Fax: 886.2.718.8869

Jeilin Technology Corporation
Tel: 886.2.248.4828
Fax: 886.2.248.9765

United Kingdom
LSI Logic Europe plc
Bracknell
♦ Tel: 44.1344.426544
Fax: 44.1344.481039

♦ Sales Offices with Design Resource Centers

ISO 9000 Certified

Printed in USA
796.1K.TP.G