User’s Manual

NEC

V$_R$5000™,V$_R$5000A™

64-/32-bit Microprocessor

μPD30500
μPD30500A

Document No. U11761EJ6V0UM00 (6th edition)
Date Published April 2001 N CP(K)

[MEMO]

NOTES FOR CMOS DEVICES

① PRECAUTION AGAINST ESD FOR SEMICONDUCTORS

Note:

Strong electric field, when exposed to a MOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop generation of static electricity as much as possible, and quickly dissipate it once, when it has occurred. Environmental control must be adequate. When it is dry, humidifier should be used. It is recommended to avoid using insulators that easily build static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work bench and floor should be grounded. The operator should be grounded using wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions need to be taken for PW boards with semiconductor devices on it.

② HANDLING OF UNUSED INPUT PINS FOR CMOS

Note:

No connection for CMOS device inputs can be cause of malfunction. If no connection is provided to the input pins, it is possible that an internal input level may be generated due to noise, etc., hence causing malfunction. CMOS devices behave differently than Bipolar or NMOS devices. Input levels of CMOS devices must be fixed high or low by using a pull-up or pull-down circuitry. Each unused pin should be connected to VDD or GND with a resistor, if it is considered to have a possibility of being an output pin. All handling related to the unused pins must be judged device by device and related specifications governing the devices.

③ STATUS BEFORE INITIALIZATION OF MOS DEVICES

Note:

Power-on does not necessarily define initial status of MOS device. Production process of MOS does not define the initial operation status of the device. Immediately after the power source is turned ON, the devices with reset function have not yet been initialized. Hence, power-on does not guarantee out-pin levels, I/O settings or contents of registers. Device is not initialized until the reset signal is received. Reset operation must be executed immediately after power-on for devices having reset function.

Exporting this product or equipment that includes this product may require a governmental license from the U.S.A. for some countries because this product utilizes technologies limited by the export control regulations of the U.S.A.

- **The information in this document is current as of March, 2001. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC semiconductor products. Not all products and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.**
- No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this document.
- NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC semiconductor products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC endeavours to enhance the quality, reliability and safety of NEC semiconductor products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC semiconductor products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment, and anti-failure features.
- NEC semiconductor products are classified into the following three quality grades:
  "Standard", "Special" and "Specific". The "Specific" quality grade applies only to semiconductor products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of a semiconductor product depend on its quality grade, as indicated below. Customers must check the quality grade of each semiconductor product before using it in a particular application.

  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots

  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support)

  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

  The quality grade of NEC semiconductor products is "Standard" unless otherwise expressly specified in NEC's data sheets or data books, etc. If customers wish to use NEC semiconductor products in applications not intended by NEC, they must contact an NEC sales representative in advance to determine NEC's willingness to support a given application.

  (Note)

  (1) "NEC" as used in this statement means NEC Corporation and also includes its majority-owned subsidiaries.

  (2) "NEC semiconductor products" means any semiconductor product developed or manufactured by or for NEC (as defined above).

M8E 00. 4

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-3067-5800
Fax: 01-3067-5899

NEC Electronics (France) S.A.
Madrid Office
Madrid, Spain
Tel: 091-504-2787
Fax: 091-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

J01.2

MAJOR REVISIONS IN THIS EDITION

Page	Description
p. 143	Correction of description in **7.2.5 (1) Status Register Format**
p. 212	Modification of description in **9.4.6 Unimplemented Instruction Exception (E)**

The mark ★ shows major revised points.

PREFACE

Readers

This manual targets users who wish to understand the functions of the V_R5000 (μPD30500), V_R5000A (μPD30500A) and design application systems using this microprocessor.

Purpose

This manual introduces the architecture and hardware functions of the V_R5000 and V_R5000A to users, following the organization described below.

Organization

This manual consists of the following contents:

- Introduction
- Pipeline operation
- Memory management system and cache organization
- Exception processing
- Floating-point operation
- System interface operation

How to read this manual

It is assumed that the reader of this manual has general knowledge in the fields of electric engineering, logic circuits, and microcomputers.

Unless otherwise specified, V_R5000 is described as a representative product in this manual. When using this manual as that for V_R5000A, read as follows.

V_R5000→V_R5000A

The V_R4400™ in this manual represents the V_R4000™.

The V_R4000 Series™ in this manual represents the V_R4100™, V_R4200™, VR4300™, V_R4305™, V_R4310™, and V_R4400.

To learn about detailed function of a specific instruction,
-> Refer to **Chapter 3 CPU Instruction Set Summary**, **Chapter 8 Floating Point Unit**, or **V_R5000**, **V_R10000™ User's Manual Instruction** which is separately available.

To learn about the overall functions of the V_R5000,
-> Read this manual in sequential order.

To learn about electrical specifications,
-> Refer to **Data Sheet** which is separately available.

Legend

Data significance: Higher on left and lower on right
Active low: XXX*
Numeric representation: binary ... XXXX or $XXXX_2$
decimal ... XXXX
hexadecimal ... 0xXXXX
Prefixes representing an exponent of 2 (for address space or memory capacity) :

K (kilo)	$2^{10} = 1024$
M (mega)	$2^{20} = 1024^2$
G (giga)	$2^{30} = 1024^3$
T (tera)	$2^{40} = 1024^4$
P (peta)	$2^{50} = 1024^5$
E (exa)	$2^{60} = 1024^6$

Related Documents

See also the following documents.
The related documents indicated here may include preliminary version. However, preliminary versions are not marked as such.

Documents Related to Devices

Document Name	Document No.
μPD30500, 30500A (V$_R$5000, V$_R$5000A) Data Sheet	U12031E
V$_R$5000, V$_R$5000A User's Manual	This Manual
μPD30700, 30700L, 30710 (V$_R$10000, V$_R$12000$^{\text{TM}}$) Data Sheet	U12703E
V$_R$10000 Series$^{\text{TM}}$ User's Manual	U10278E
V$_R$5000, V$_R$10000 INSTRUCTION User's Manual	U12754E

Application Note

Document Name	Document No.
V$_R$Series$^{\text{TM}}$ Application Note Programming Guide	U10710E

Table of Contents

List of Figures (1/5)

Fig. No. Title Page

List of Figures (2/5)

List of Figures (3/5)

Fig. No. **Title** **Page**

List of Figures (4/5)

List of Figures (5/5)

Fig. No. **Title** **Page**

List of Tables (1/3)

List of Tables (2/3)

List of Tables (3/3)

Chapter 1 Introduction

The V_R5000 and V_R5000A are members of the NEC V_R-Series RISC (Reduced Instruction Set Computer) microprocessors and are high-performance 64-/32-bit microprocessors employing the RISC architecture developed by MIPS™.

Their instructions are upward-compatible with those of the V_R3000 Series™ and V_R4000 Series and are completely compatible with those of the V_R10000. Therefore, existing applications can be used with the V_R5000 and V_R5000A.

1.1 Processor Characteristics

The V_R5000 and V_R5000A have the following fetaures:

- Maximum internal operating frequency:
  150MHz (μPD30500-150) /180MHz (μPD30500-180) /
  200MHz (μPD30500-200) /250MHz (μPD30500A-250)/
  266MHz (μPD30500A-266)
- 64-bit architecture supporting 64-bit data processing
- Dual-issue instruction mechanism
- High-speed translation lookaside buffer (TLB) supporting virtual addresses (of 48 double entires)
- Address space:

Address space:	Physical	36 bits
	Virtual	40 bits (64-bit mode)
		31 bits (32-bit mode)

- Supports single-precision and double-precision floating-point operations
- On-chip primary cache:

On-chip primary cache:	Instruction	32KB
	Data	32KB

- Up to 2MB optional Secondary cache
- Employs writeback system -> store operation via system bus decreased
- Up to 100 MHz external bus with frequency of /2, /2.5$^{\textbf{Note}}$, /3, /4, /5, /6, /7, /8 of internal operation
- Write buffer
- Upward-compatible with V_R3000 Series and V_R4000 Series and completely compatible with V_R10000
- Supply voltage: Vcc=3.3V±5% (V_R5000)
  Core : Vcc=2.4V±0.1V (V_R5000A, 100 to 235MHz),
  Vcc=2.5V+5% (V_R5000A, 236 to 250MHz),
  Vcc=2.6V±0.1V (V_R5000A, 251 to 266MHz)
  I/O : VccIO=3.3V±5%(V_R5000A)

Note Selectable only when external operating frequency=100MHz

1.2 Ordering Information

Part Number	Package	Maximum operating frequency (MHz)
μPD30500RJ-150	223-pin ceramic PGA (48×48)	150
μPD30500RJ-180	223-pin ceramic PGA (48×48)	180
μPD30500RJ-200	223-pin ceramic PGA (48×48)	200
μPD30500S2-150	272-pin plastic BGA (cavity down advanced type) (29×29)	150
μPD30500S2-180	272-pin plastic BGA (cavity down advanced type) (29×29)	180
μPD30500S2-200	272-pin plastic BGA (cavity down advanced type) (29×29)	200
μPD30500AS2-250	272-pin plastic BGA (cavity down advanced type) (29×29)	250
μPD30500AS2-266	272-pin plastic BGA (cavity down advanced type) (29×29)	266

1.3 64-Bit Architecture

The V_R5000 is a 64-bit high-performance microprocessor. It can also execute 32-bit applications.

1.4 V_R5000 Processor

Figure 1-1 shows the internal block diagram of the V_R5000.

The V_R5000 is equipped with a full-associative high-speed translation lookaside buffer (TLB) that has 48 entries with two pages corresponding to each entry; data cache and instruction cache; external secondary cache interface, in addition to dual-issue mechanism ALU.

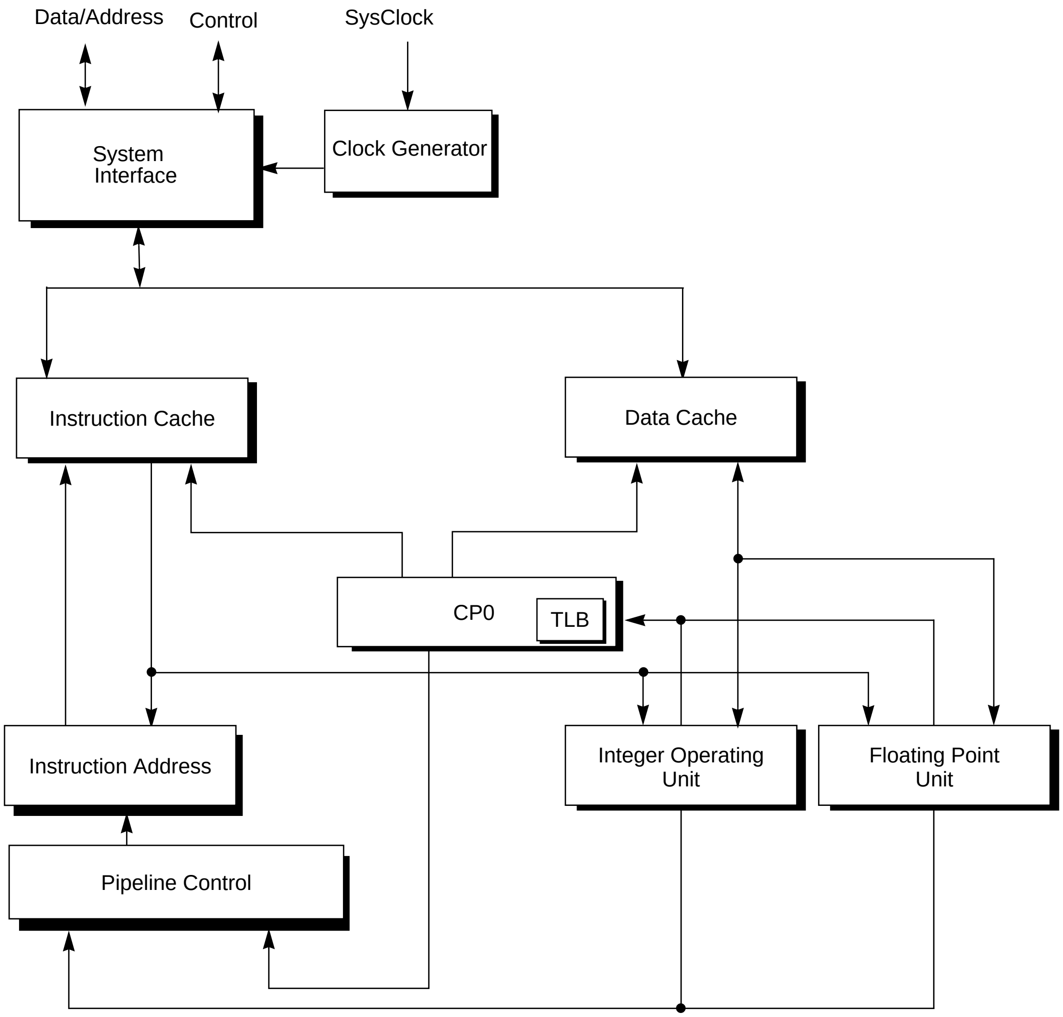


Figure 1-1 V_R5000 Processor Internal Block Diagram

1.4.1 Internal Block Configuration

System Interface allows the processor to access external resources such as memories and secondary cache. It contains a 64-bit multiplexed address/data bus, with per-byte parity, interrupt request signals, and various control signals included for secondary cache.

Clock Generator generates a pipeline clock (PClock) based on an externally input clock (SysClock). The ratio of frequency of SysClock to that of PClock can be set to 1:2, 1:2.5**Note**, 1:3, 1:4, 1:5, 1:6, 1:7, or 1:8.

Note V_R5000A only (Selectable only when SysClock=100MHz)

Instruction Cache is 2-way set associative, virtually-indexed, and physically-tagged. The capacity is 32KB.

Integer Operating Unit has the hardware resources to execute integer instruction. It has a 64-bit register file and 64-bit integer datapath. It is provided with a dedicated multiplier in order to process multiply instruction at a high speed.

Floating Point Unit has the hardware resources to execute floating point instruction. It has a 64-bit register file, 64-bit mantissa datapath, and 12-bit exponent datapath. It is provided with a dedicated multiplier and a dedicated div./sqrt. in order to process multiply/multiplyadd and div./sqrt. instructions at a high speed.

Coprocessor 0 (CP0) has the memory management unit (MMU) and handles exception processing. The MMU handles address translation and checks memory accesses that occur between different memory segments (user, supervisor, or kernel). The translation lookaside buffer (TLB) is used to translate virtual to physical addresses.

Data Cache is a 2-way set associative, virtually indexed and physically-tagged writeback cache. The capacity is 32KB.

Instruction Address calculates the effective address of the next instruction to be fetched. It contains the incrementer for the Program Counter (PC), the branch address adder, and the conditional branch selector.

Pipeline Control ensures the instruction pipeline operates properly causing either of pipeline stall or exception.

1.4.2 CPU Registers

The processor provides the following registers:

- 32 64-bit general purpose registers, *GPR*s
- 32 64-bit floating-point purpose registers, *FPR*s

In addition, the processor provides the following special registers:

- 64-bit Program Counter, the *PC* register
- 64-bit *HI* register, containing the integer multiply and divide high-order doubleword result
- 64-bit *LO* register, containing the integer multiply and divide low-order doubleword result
- 1-bit Load/Link *LLBit* register
- 32-bit floating-point *Implementation/Revision* register, FCR0
- 32-bit floating-point *Control/Status* register, FCR31

Two of the CPU general purpose registers have assigned functions:

- *r0* is hardwired to a value of zero, and can be used as the target register for any instruction whose result is to be discarded. *r0* can also be used as a source when a zero value is needed.
- *r31* is the link register used by JAL and JALR instructions. It can be used by other instructions. Make sure that other data used in calculations does not overlap with the register used by the JAL/JALR instruction.

Further more, the processor contains registers in the system control processor (CP0) which perform the exception processing and address management. CPU registers can operate as either 32-bit or 64-bit registers, depending on the V_R5000 processor mode of operation.

Figure 1-2 shows the V_R5000 processor registers.

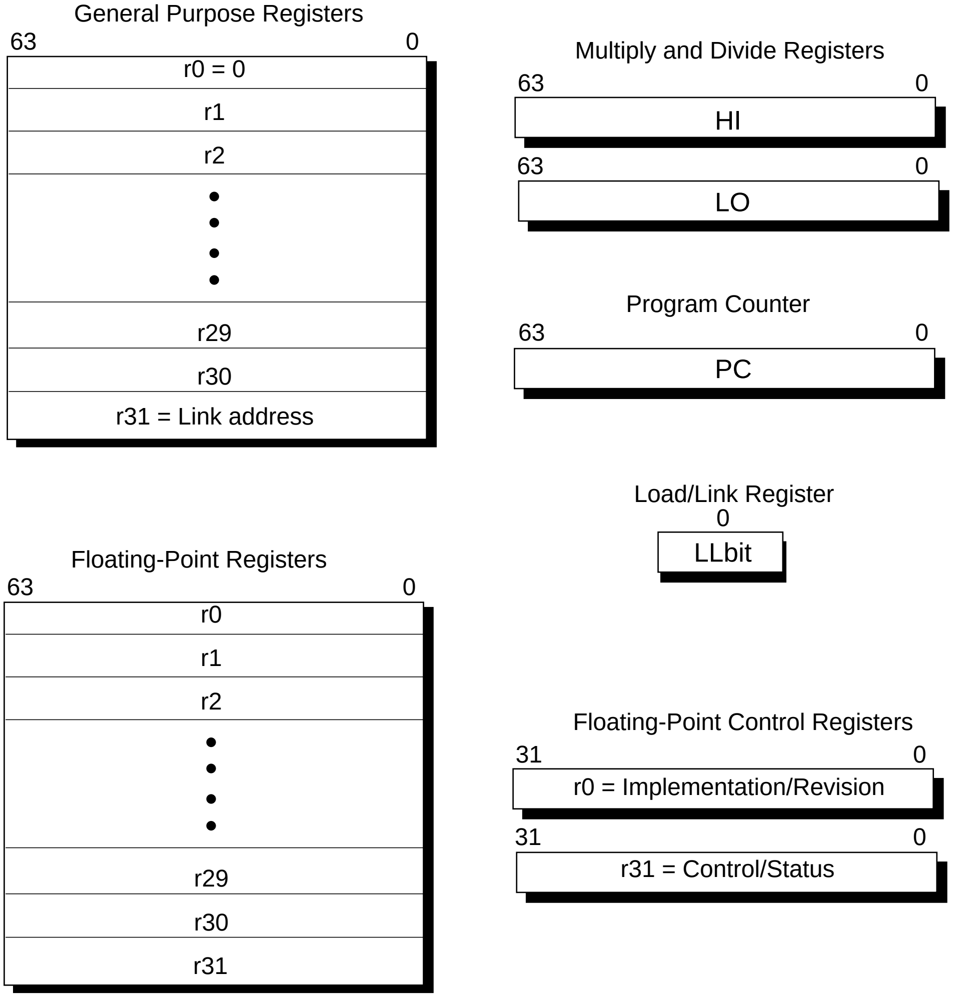


Figure 1-2 V_R5000 Processor Registers

The V_R5000 processor has no *Program Status Word* (PSW) register as such; this is covered by the *Status* and *Cause* registers incorporated within the System Control Coprocessor (CP0). CP0 registers are described later in this chapter.

1.4.3 CPU Instruction Set Overview

Each CPU instruction is 32 bits long. As shown in Figure 1-3, there are three instruction formats:

- immediate (I-type)
- jump (J-type)
- register (R-type)

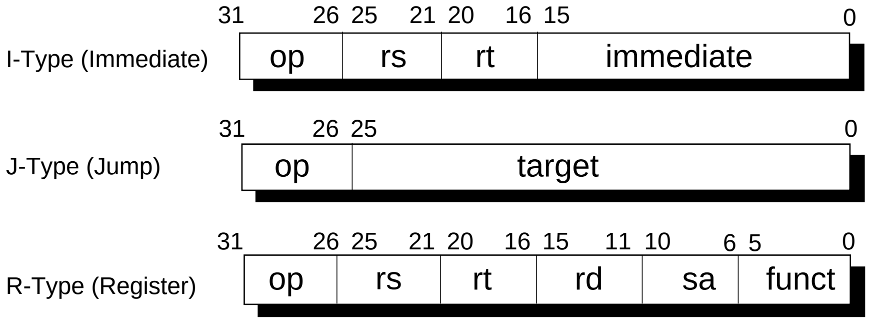


Figure 1-3 CPU Instruction Formats

The instruction set can be further divided into the following groupings:

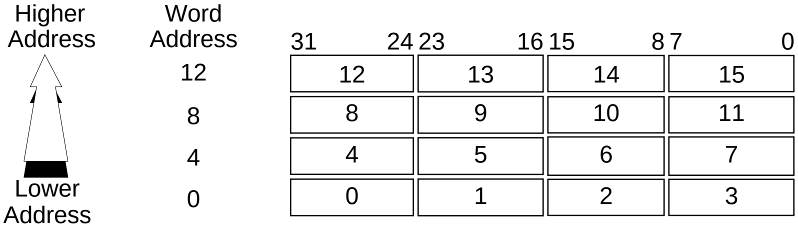
- **Load and Store** instructions move data between memory and general purpose registers. They are all immediate (I-type) instructions, since the only addressing mode supported is base register plus 16-bit, signed immediate offset.
- **Computational** instructions perform arithmetic, logical, shift, multiply, and divide operations on values in registers. They include register (R-type, in which both the operands and the result are stored in registers) and immediate (I-type, in which one operand is a 16-bit signed immediate value) formats.
- **Jump and Branch** instructions change the control flow of a program. Jumps are always made to an address formed by combining a 26-bit target address with the high-order bits of the Program Counter (J-type format) or register address (R-type format). Branch instructions are performed to the 16-bit offset address relative to the program counter (I-type). Jump And Link instructions save their return address in register 31.
- **Coprocessor** instructions (CPz) perform operations in the coprocessors. Coprocessor load and store instructions are I-type. As opposed to CP0 instructions, CPz instructions are not specific to any coprocessor. (Refer to **Chapter 8 Floating Point Unit**.)

- **Coprocessor 0** (system coprocessor, CP0) instructions perform operations on CP0 registers to control the memory-management and exception-handling facilities of the processor.
- **Special** instructions perform system call exception and breakpoint exception operations, or cause a branch to the general exception-handling vector based upon the result of a comparison. These instructions occur in both R-type (both the operands and the result are registers) and I-type (one operand is a 16-bit immediate value) formats.

For each instruction, refer to **Chapter 3 CPU Instruction Set Summary** and **V$_R$5000, V$_R$10000 User's Manual Instruction**.

1.4.4 Data Formats and Addressing

The V_R5000 processor uses four data formats: a 64-bit doubleword, a 32-bit word, a 16-bit halfword, and an 8-bit byte. Byte ordering within all of the larger data formats—halfword, word, doubleword—can be configured in either big-endian or little-endian. When the V_R5000 processor is configured as a big-endian system, byte 0 is the most-significant (leftmost) byte, thereby providing compatibility with MC 68000™ and IBM 370™ conventions. Figure 1-4 shows this configuration.



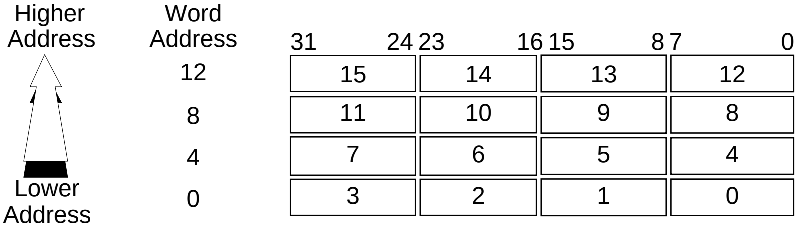
Word Address	31 24	23 16	15 8	7 0
12	12	13	14	15
8	8	9	10	11
4	4	5	6	7
0	0	1	2	3

Figure 1-4 Big-Endian Byte Ordering

Remarks **1.** The most-significant byte is the lowest address.
2. A word is addressed by the address of the most-significant byte.

When configured as a little-endian system, byte 0 is always the least-significant (rightmost) byte, which is compatible with iAPX™ x86 and DEC VAX™ conventions. Figure 1-5 shows this configuration.

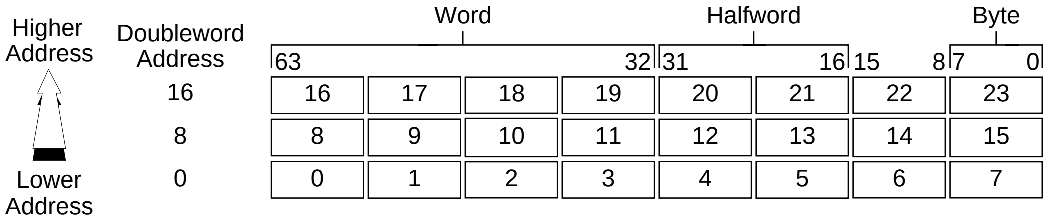
Unless otherwise specified, the little endian is used throughout this manual.



Word Address	31 24	23 16	15 8	7 0
12	15	14	13	12
8	11	10	9	8
4	7	6	5	4
0	3	2	1	0

Figure 1-5 Little-Endian Byte Ordering

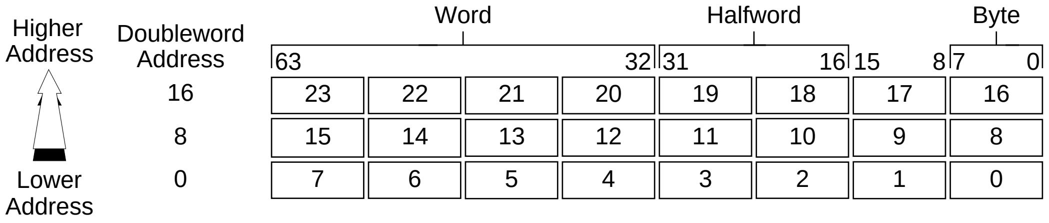
Remarks **1.** The least-significant byte is the lowest address.
2. A word is addressed by the address of the least-significant byte.



Doubleword Address								
16	16	17	18	19	20	21	22	23
8	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7

Figure 1-6 Big-Endian Data in a Doubleword

Remarks **1.** The most-significant byte is the lowest address.
2. A word is addressed by the address of the most-significant byte.



Doubleword Address								
16	23	22	21	20	19	18	17	16
8	15	14	13	12	11	10	9	8
0	7	6	5	4	3	2	1	0

Figure 1-7 Little-Endian Data in a Doubleword

Remarks **1.** The least-significant byte is the lowest address.
2. A word is addressed by the address of the least-significant byte.

The CPU uses byte addressing for halfword, word, and doubleword accesses with the following alignment constraints:

- Halfword accesses must be aligned on an even byte boundary (0, 2, 4...).
- Word accesses must be aligned on a byte boundary divisible by four (0, 4, 8...).
- Doubleword accesses must be aligned on a byte boundary divisible by eight (0, 8, 16...).

The following special instructions load and store words that are not aligned on 4-byte (word) or 8-word (doubleword) boundaries:

LWL **LWR** **SWL** **SWR**

LDL **LDR** **SDL** **SDR**

These instructions are always used in pairs to access data not aligned at an boundary. To access data not aligned at a boundary, additional 1P cycle is necessary as compared when accessing data aligned at a boundary.

Figure 1-8 illustrates how a word misaligned and having byte address 3 is accessed in big and little endian.

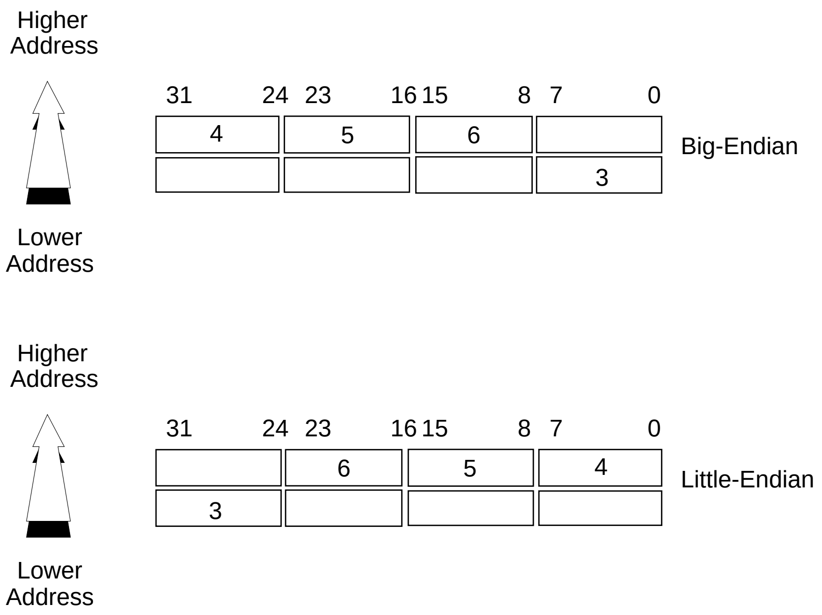


Figure 1-8   Misaligned Word Addressing

1.4.5 System Control Coprocessor (CP0)

The CPU can operate with up to four coprocessors (CP0 through CP3) closely coupled. Coprocessors 1 and 2 are reserved for future use. Coprocessor 3 is assigned for MIPS IV instruction set. Coprocessor 0 (CP0) is an internal system control coprocessor and supports the virtual memory system and exception processing. The virtual memory system is executed by the on-chip TLB and CP0 register.

CP0 converts virtual addresses into physical addresses, selects an operating mode (Kernel, supervisor, or user mode), and control exceptions. It also controls the cache subsystem to analyze causes and return execution from error processing. The CP0 register of the V_R5000 is the same as that of the V_R4000.

Figure 1-9 shows the CP0 register. Table 1-1 briefly explains each register. For the details of the registers related to the virtual memory system, refer to **Chapter 6 Memory Management Unit**, and for the details of the registers used for exception processing, refer to **Chapter 7 CPU Exception Processing.**

Register Name	Reg. #	Register Name	Reg. #
*Index**	*0*	*Config**	*16*
*Random**	*1*	*LLAddr**	*17*
*EntryLo0**	*2*	*RFU*	*18*
*EntryLo1**	*3*	*RFU*	*19*
*Context***	*4*	*XContext***	*20*
*PageMask**	*5*	*RFU*	*21*
*Wired**	*6*	*RFU*	*22*
RFU	*7*	*RFU*	*23*
*BadVAddr***	*8*	*RFU*	*24*
*Count***	*9*	*RFU*	*25*
*EntryHi**	*10*	*Parity Error***	*26*
*Compare***	*11*	*Cache Error***	*27*
*Status***	*12*	*TagLo**	*28*
*Cause***	*13*	*TagHi**	*29*
*EPC***	*14*	*ErrorEPC***	*30*
*PRId**	*15*	*RFU*	*31*

** For Memory Management*
*** For Exception Processing*
RFU Reserved for Future Use

Figure 1-9 CP0 Registers

Table 1-1   System Control Coprocessor (CP0) Register Definitions

Number	Register	Description
0	Index	Programmable pointer into TLB array
1	Random	Pseudorandom pointer into TLB array *(read only)*
2	EntryLo0	Low half of TLB entry for even virtual address (VPN)
3	EntryLo1	Low half of TLB entry for odd virtual address (VPN)
4	Context	Pointer to kernel virtual page table entry (PTE) in 32-bit mode
5	PageMask	Page size specification
6	Wired	Number of wired TLB entries
7	—	Reserved for future use
8	BadVAddr	Display of virtual address that occurred an error last
9	Count	Timer Count
10	EntryHi	High half of TLB entry (including ASID)
11	Compare	Timer Compare Value
12	Status	Operation status setting
13	Cause	Display of cause of last exception
14	EPC	Exception Program Counter
15	PRId	Processor Revision Identifier
16	Config	Memory system mode setting
17	LLAddr	Load Linked instruction address display
18, 19	—	Reserved for future use
20	XContext	Pointer to Kernel virtual PTE table in 64-bit mode
21–25	—	Reserved for future use
26	Parity Error	Cache parity bits
27	Cache Error	Cache Error and Status register
28	TagLo	Cache Tag register low
29	TagHi	Cache Tag register high
30	ErrorEPC	Error Exception Program Counter
31	—	Reserved for future use

1.4.6 Floating-Point Unit (FPU)

The floating-point unit (FPU) performs arithmetic operations on floating-point values. The FPU, with associated system software, fully conforms to the requirements of ANSI/IEEE Standard 754–1985, *IEEE Standard for Binary Floating-Point Arithmetic*.

The FPU includes:

- **Full 64-bit Operation**. The FPU can contain either 16 64-bit registers to hold single-precision or double-precision values. Another sixteen floating-point registers can be used by setting the FR bit of the *Status* register to 1. Moreover, a 32-bit *Control/Status* register is provided, conforming to the IEEE exception processing standard.
- **Load and Store Instruction Set**. Like the CPU, the FPU uses a load- and store-based instruction set. Floating-point operations are started in a single cycle.

1.4.7 Internal Cache

The V_R5000 has an instruction cache and a data cache to enhance the efficiency of pipelining. Each cache has a data width of 64 bits and can be accessed in 1 clock. The instruction cache and data cache can be accessed in parallel. Both of the instruction cache and data cache have a capacity of 32KB.

For the details of the cache, refer to **Chapter 12 Cache Organization and Operation**.

1.5 Memory Management System (MMU)

The V_R5000 processor has a 36-bit physical addressing range of 64 GB. However, since it is rare for systems to implement a physical memory space this large, the CPU provides a logical expansion of memory space to the programmer by translating addresses into the large virtual address space. The V_R5000 processor supports the following two addressing modes:

- 32-bit mode, in which the virtual address space is divided into 2 GB per user process and 2 GB for the kernel.
- 64-bit mode, in which the virtual address is expanded to 1 TB (2^{40} bytes) of user virtual address space.

A detailed description of these address spaces is given in **Chapter 6 Memory Management Unit**.

1.5.1 Translation Lookaside Buffer (TLB)

Virtual memory mapping is assisted by a translation lookaside buffer, which holds virtual-to-physical address translations. This fully-associative, on-chip TLB contains 48 entries, each of which maps a pair of variable-sized pages of either 4 KB or 16 MB.

Joint TLB (JTLB)

The TLB can hold both instruction and data addresses, and is thus also referred to as a joint TLB (JTLB).

An address translation value is tagged with the most-significant bits of its virtual address (the number of these bits depends upon the size of the page) and a per-process identifier. If there is no matching entry in the TLB, an exception occurs and software writes the entry contents to the on-chip TLB from a page table in memory. The JTLB entry to be rewritten is selected by a value in either the *Random* or *Index* register.

1.5.2 Operating Modes

The V_R5000 processor has three operating modes:

- User mode
- Supervisor mode
- Kernel mode

The manner in which memory addresses are translated or *mapped* depends on the operating mode of the CPU; this is described in **Chapter 6 Memory Management Unit**.

1.6 Instruction Pipeline

The V_R5000 incorporates a simple dual-issue mechanism which allows a floating-point ALU instruction to be issued simultaneously with any other instruction type and has a five-stage instruction pipeline. For details, refer to **Chapter 4 V_R5000 Processor Pipeline** and **Chapter 5 Superscalar Issue Mechanism**.

Chapter 2 V_R5000 Processor Signal Descriptions

This chapter describes the signals used by and in conjunction with the V_R5000 processor. The signals include the System interface, the Clock interface, the Secondary Cache interface, the Interrupt interface, and the Initialization interface.

Signals are listed in bold, and low active signals have a trailing asterisk—for instance, the low-active Read Ready signal is **RdRdy***. The arrows used in each signal for each signals tells if the signal is an input (the processor receives it), an output (the processor sends it out), or bidirectional.

Figure 2-1 illustrates the functional groupings of the processor signals.

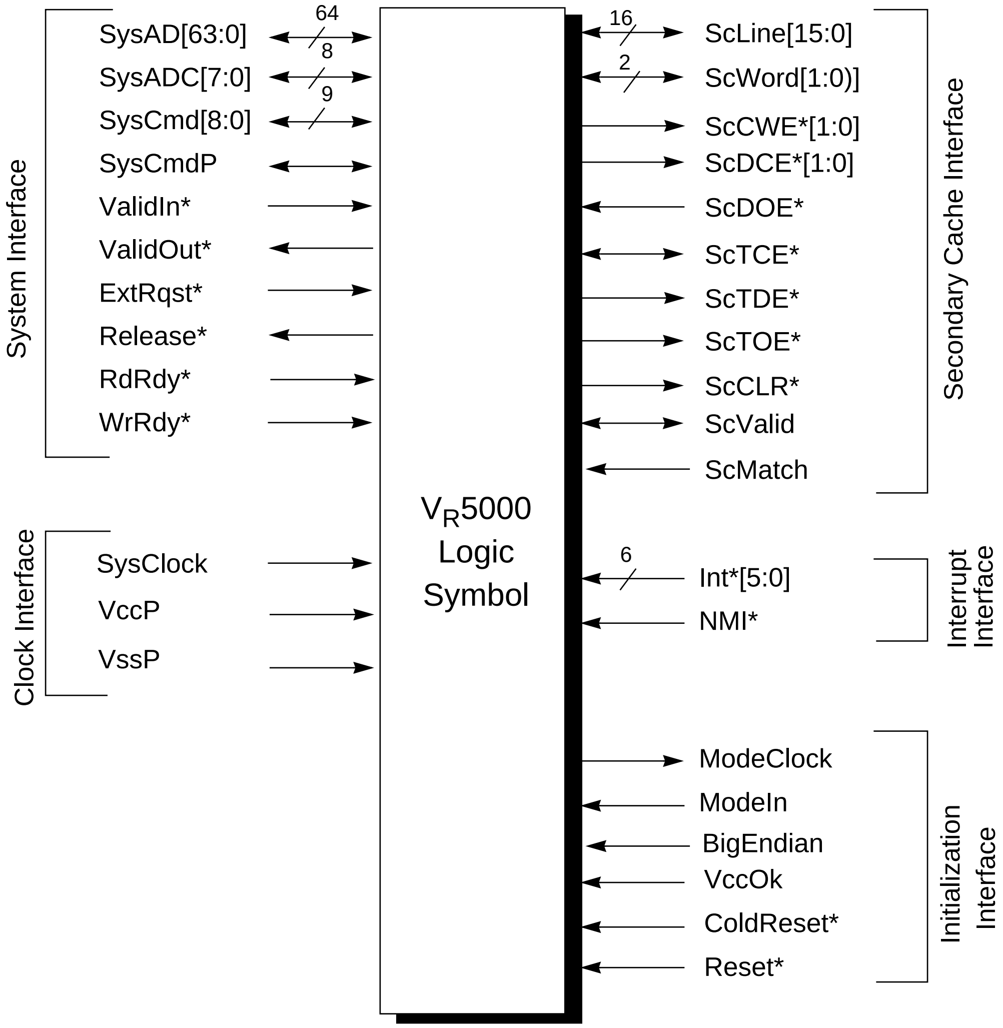


Figure 2-1 V_R5000 Processor Signals

2.1 System Interface Signals

System interface signals provide the connection between the V_R5000 processor and the other components in the system. Table 2-1 lists the system interface signals.

Table 2-1 System Interface Signals

Name	Definition	Direction	Description
ExtRqst*	External request	Input	An external agent asserts **ExtRqst*** to request use of the System interface. The processor grants the request by asserting **Release***.
Release*	Release interface	Output	In response to the assertion of **ExtRqst***, the processor asserts **Release***, signalling to the requesting device that the System interface is available.
RdRdy*	Read ready	Input	The external agent asserts **RdRdy*** to indicate that it can accept processor read requests in either secondary or no-secondary cache mode.
SysAD(63:0)	System address/ data bus	Input/ Output	A 64-bit address and data bus for communication between the processor, the secondary cache, and an external agent.
SysADC(7:0)	System address/ data check bus	Input/ Output	An 8-bit bus containing parity for the **SysAD** bus. **SysADC** is valid on data cycles only.
SysCmd(8:0)	System command/ data identifier	Input/ Output	A 9-bit bus for command and data identifier transmission between the processor and an external agent.
SysCmdP	System command/ data identifier bus parity	Input/ Output	Always zero when driven by the processor. Never checked by the processor. This signal is defined to maintain V_R4000 compatiblility.
ValidIn*	Valid input	Input	The external agent asserts **ValidIn*** when it is driving a valid address or data on the **SysAD** bus and a valid command or data identifier on the **SysCmd** bus.
ValidOut*	Valid output	Output	The processor asserts **ValidOut*** when it is driving a valid address or data on the **SysAD** bus and a valid command or data identifier on the **SysCmd** bus to the external agent.
WrRdy*	Write ready	Input	The external agent asserts **WrRdy*** when it can accept a processor write request.

2.2 Clock Interface Signals

The Clock interface signals make up the interface for clocking. Table 2-2 lists the Clock interface signals.

Table 2-2 Clock Interface Signals

Name	Definition	Direction	Description
SysClock	System Clock	Input	System clock input that establishes the system interface operating frequency and phase.
VccP	Quiet Vcc for PLL	Input	Quiet Vcc for the internal phase locked loop.
VssP	Quiet Vss for PLL	Input	Quiet Vss for the internal phase locked loop.

2.3 Secondary Cache Interface Signals

Secondary Cache interface signals constitute the interface between the V_R5000 processor and secondary cache. Table 2-3 lists the Secondary Cache interface signals in alphabetical order.

Table 2-3 Secondary Cache Interface Signals

Name	Definition	Direction	Description
ScCLR*	Secondary Cache Flash Clear	Output	Clears all valid bits in those Tag RAMs which support this function.
ScCWE*(1:0)	Secondary Cache Write Enable	Output	Asserted during writes to the secondary cache. Two signals are provided to minimize loading from the cache RAMs.
ScDCE*(1:0)	Data RAM Chip Enable	Output	Chip Enable for Secondary Cache Data RAM. Two signals are provided to minimize loading from the cache RAMs.
ScDOE*	Data RAM Output Enable	Input	Asserted by the external agent to enable data onto the **SysAD** bus
ScLine (15:0)	Secondary Cache Line Index	Output	Cache line index for secondary cache
ScMatch	Secondary cache Tag Match	Input	Asserted by Tag RAM on Secondary cache tag match
ScTCE*	Secondary cache Tag RAM Chip Enable	Output	Chip enable for secondary cache tag RAM.
ScTDE*	Secondary cache Tag RAM Data Enable	Output	Data Enable for Secondary Cache Tag RAM.
ScTOE*	Secondary cache Tag RAM Output Enable	Output	Tag RAM Output enable for Secondary Cache Tag RAM
ScWord (1:0)	Secondary cache Word Index	Input/Output	Determines the double-word within the indexed secondary cache Index
ScValid	Secondary cache Valid	Input/Output	Always driven by the CPU except during a CACHE Probe operation, where it is driven by the Tag RAM.

2.4 Interrupt Interface Signals

The Interrupt interface signals make up the interface used by external agents to interrupt the V_R5000 processor. Table 2-4 lists the Interrupt interface signals.

Table 2-4 Interrupt Interface Signals

Name	Definition	Direction	Description
Int*(5:0)	Interrupt	Input	General processor interrupts, bit-wise ORed with bits 5:0 of the interrupt register.
NMI*	Nonmaskable interrupt	Input	Nonmaskable interrupt, ORed with bit 6 of the interrupt register.

2.5 Initialization Interface Signals

The Initialization interface signals make up the interface by which an external agent initializes the processor operating parameters. Table 2-5 lists the Initialization interface signals.

Table 2-5 Initialization Interface Signals

Name	Definition	Direction	Description
BigEndian	Endian Mode Select	Input	Allows the system to change the processor addressing mode without rewriting the mode ROM. If endianness is to be specified via the **BigEndian** pin, program mode ROM bit 8 to zero. If endianness is to be specified by the mode ROM, ground the **BigEndian** pin.
ColdReset*	Cold reset	Input	This signal must be asserted for a power on reset or a cold reset. **ColdReset*** must be deasserted synchronously with **SysClock**.
ModeClock	Boot mode clock	Output	Serial boot-mode data clock output; runs at the system clock frequency divided by 256: (**SysClock**/256).
ModeIn	Boot mode data in	Input	Serial boot-mode data input.
Reset*	Reset	Input	This signal must be asserted for any reset sequence. It can be asserted synchronously or asynchronously for a cold reset, or synchronously to initiate a warm reset. **Reset*** must be deasserted synchronously with **SysClock**.
VccOk	Vcc and VccIO**Note** are valid	Input	When asserted, this signal indicates to the processor that the +3.3 volt power supply has been above 3.135 volts for more than 100 milliseconds and will remain stable. The assertion of **VccOk** initiates the initialization sequence.

Note VccIO is only for V_R5000A.

2.6 Power Supply

Table 2-6 Power Supply

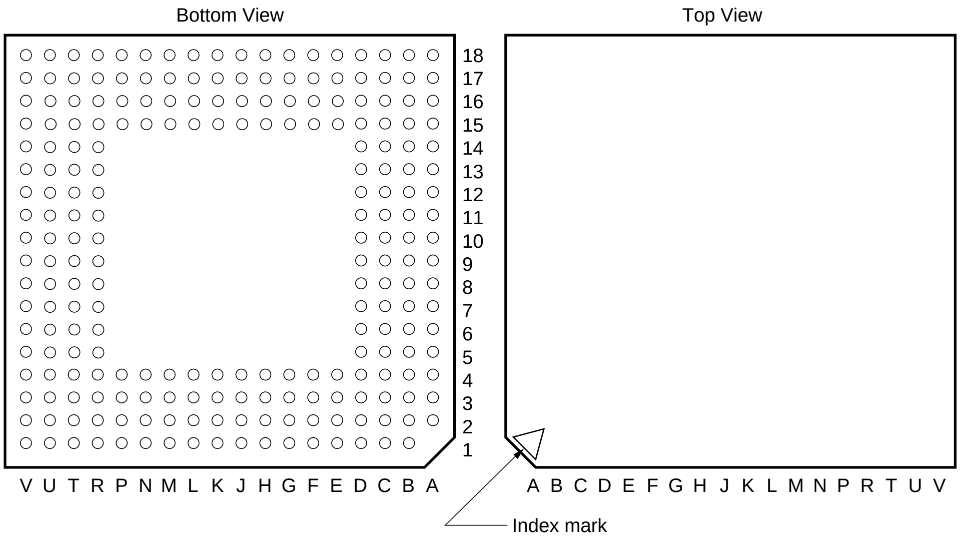
Name	Definition	Direction	Description
Vss	Vss for Processor Core and Processor I/O	–	Ground for the internal core logic and processor I/O interface.
Vcc	V_R5000 : Power supply	–	Positive power supply pin (3.3V)
	V_R5000A : Power supply for Processor Core		Power supply pin for core (100 to 235MHz: 2.4V, 236 to 250MHz: 2.5V, 251 to 266MHz: 2.6V)
VccIO[Note]	Power supply for Processor I/O	–	Power supply pin for I/O (3.3V)

Note V_R5000A only

Caution **Two kind of power sources are provided with the V_R5000A. The sequence of the power application order is not fixed. However, make sure that either of the power supplies does not remain turned on for 1 second or more while the other remains off.**

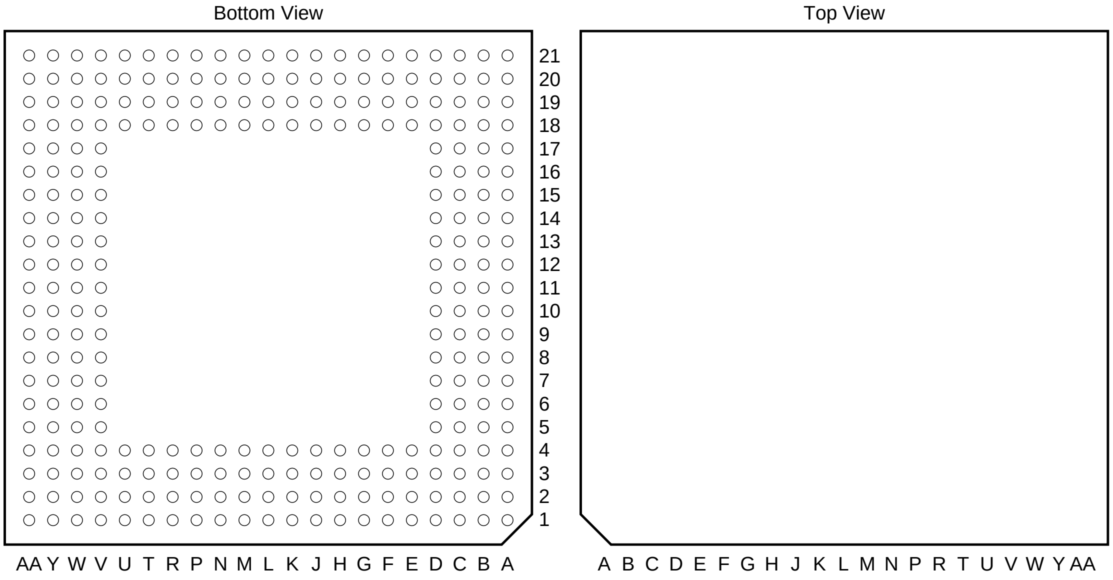
2.7 Pin Configuration

- **223-pin ceramic PGA (48 × 48)**
  μPD30500RJ-150
  μPD30500RJ-180
  μPD30500RJ-200


Bottom View
Top View
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
V U T R P N M L K J H G F E D C B A
A B C D E F G H J K L M N P R T U V
Index mark

Location.......Name	Location.......Name	Location ...... Name	Location....... Name	Location....... Name	Location ...... Name
A2 Vcc	C5 ...... SysADC[6]	E18 Vcc	K17 .... VssP	R6 .... SysAD[51]	U9 .... SysAD[63]
A3 .... Vss	C6 .... SysAD[16]	F1 Vcc	K18 .... Vss	R7 .... SysAD[55]	U10 ...... SysAD[13]
A4 Vcc	C7 .... SysAD[50]	F2 .... Reserved	L1 .... Vss	R8 .... SysAD[27]	U11 ...... SysAD[11]
A5 .... Vss	C8 .... SysAD[22]	F3 .... ScValid	L2 .... SysCmd[8]	R9 .... SysAD[31]	U12 .... SysAD[9]
A6 .... Vss	C9 .... SysAD[24]	F4 .... INT[1]*	L3 .... SysCmd[7]	R10 ..... SysAD[43]	U13 ...... SysAD[37]
A7 Vcc	C10 ..... SysAD[28]	F15 ..... ScDCE[0]*	L4 .... SysCmd[5]	R11 ..... SysAD[39]	U14 .... SysAD[3]
A8 .... Vss	C11 ..... SysAD[62]	F16 .... ScCWE[0]*	L15 ...... ScLine[12]	R12 ..... SysAD[35]	U15 ...... ScWord[0]
A9 Vcc	C12 ..... SysAD[44]	F17 .... ScTDE*	L16 ...... ScLine[14]	R13 .... SysAD[1]	U16 Vcc
A10 .... Vss	C13 ..... SysAD[10]	F18 .... Vss	L17 ...... ScLine[15]	R14 ...... ScWord[1]	U17 .... Vss
A11 Vcc	C14 ..... SysAD[38]	G1 .... Vss	L18 Vcc	R15 .... ScLine[0]	U18 .... Vss
A12 .... Vss	C15 .... SysAD[4]	G2 .... Reserved	M1 Vcc	R16 .... ScLine[3]	V1 .... Vss
A13 Vcc	C16 ..... SysAD[34]	G3 .... Reserved	M2 ...... SysCmd[6]	R17 .... ScLine[6]	V2 .... Vss
A14 .... Vss	C17 .... SysAD[2]	G4 .... Reserved	M3 ...... SysCmd[4]	R18 .... Vss	V3 Vcc
A15 .... Vss	C18 .... Vss	G15 .... ScCLR*	M4 ...... SysCmd[1]	T1 .... Vss	V4 .... Vss
A16 Vcc	D1 .... Vss	G16 .... ScTCE*	M15 .... ScLine[8]	T2 .... SysAD[15]	V5 .... Vss
A17 .... Vss	D2 .... INT3*	G17 .... ModeIn	M16 ..... ScLine[10]	T3 .... SysAD[47]	V6 Vcc
A18 .... Vss	D3 .... INT5*	G18 Vcc	M17 ..... ScLine[13]	T4 .... SysAD[17]	V7 .... Vss
B1 .... Vss	D4 .... Release*	H1 Vcc	M18 .... Vss	T5 .... SysAD[19]	V8 Vcc
B2 .... Vss	D5 Vcc	H2 .... Reserved	N1 .... Vss	T6 .... SysAD[23]	V9 .... Vss
B3 Vcc	D6 ...... SysADC[2]	H3 .... Reserved	N2 .... SysCmd[3]	T7 .... SysAD[57]	V10 Vcc
B4 ...... SysADC[4]	D7 .... SysAD[48]	H4 .... Reserved	N3 .... SysCmd[2]	T8 .... SysAD[29]	V11 .... Vss
B5 ...... SysADC[0]	D8 .... SysAD[52]	H15 .... VccOK	N4 .... SysADC[7]	T9 Vcc	V12 Vcc
B6 .... SysAD[18]	D9 .... SysAD[56]	H16 ..... ModeClock	N15 .... ScLine[5]	T10 ..... SysAD[45]	V13 .... Vss
B7 .... SysAD[20]	D10 ..... SysAD[60]	H17 .... SysClock	N16 .... ScLine[7]	T11 ..... SysAD[41]	V14 Vcc
B8 .... SysAD[54]	D11 ..... SysAD[14]	H18 .... Vss	N17 ...... ScLine[11]	T12 .... SysAD[7]	V15 .... Vss
B9 .... SysAD[26]	D12 ..... SysAD[42]	J1 .... Vss	N18 Vcc	T13 .... SysAD[5]	V16 .... Vss
B10 ..... SysAD[58]	D13 .... SysAD[8]	J2 .... WrRdy*	P1 Vcc	T14 ..... SysAD[33]	V17 Vcc
B11 ..... SysAD[30]	D14 ..... SysAD[36]	J3 .... ValidIn*	P2 .... SysCmd[0]	T15 .... Reset*	V18 .... Vss
B12 ..... SysAD[46]	D15 ..... ColdReset*	J4 .... ExtReq*	P3 .... SysCmdP	T16 .... ScLine[1]	
B13 ..... SysAD[12]	D16 .... SysAD[0]	J15 .... Reserved	P4 .... SysADC[1]	T17 Vcc	
B14 ..... SysAD[40]	D17 .... ScTOE*	J16 .... Reserved	P15 .... ScLine[2]	T18 Vcc	
B15 .... SysAD[6]	D18 Vcc	J17 .... Reserved	P16 .... ScLine[4]	U1 Vcc	
B16 .... Vss	E1 .... Vss	J18 Vcc	P17 .... ScLine[9]	U2 Vcc	
B17 Vcc	E2 .... INT[0]*	K1 Vcc	P18 .... Vss	U3 .... Vss	
B18 Vcc	E3 .... INT[2]*	K2 .... ScMatch	R1 Vcc	U4 .... SysAD[21]	
C1 Vcc	E4 .... INT[4]*	K3 .... RdRdy*	R2 .... SysADC[5]	U5 .... SysAD[53]	
C2 Vcc	E15 ...... SysAD[32]	K4 .... ScDOE*	R3 .... SysADC[3]	U6 .... SysAD[25]	
C3 .... ValidOut*	E16 ...... ScDCE[1]*	K15 .... Reserved	R4 .... BigEndian	U7 .... SysAD[59]	
C4 .... NMI*	E17 ..... ScCWE[1]*	K16 .... VccP	R5 .... SysAD[49]	U8 .... SysAD[61]	

- **272-pin plastic BGA (cavity down advanced type) (29 × 29)**
  μPD30500S2-150
  μPD30500S2-180
  μPD30500S2-200
  μPD30500AS2-250
  μPD30500AS2-266


Bottom View
Top View
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
AA Y W V U T R P N M L K J H G F E D C B A
A B C D E F G H J K L M N P R T U V W Y AA

(1) μPD30500

Location.......Name	Location.......Name	Location...... Name	Location.......Name	Location.......Name	Location...... Name
A1 .... Vss	C1 .... Vss	E1 .... Vss	K3....... SysAD[62]	R18 Vcc	W2........ Vcc
A2 Vcc	C2 Vcc	E2........SysAD[36]	K4........ Vcc	R19..... SysAD[53]	W3........ Vcc
A3 .... Vss	C3 ....ColdReset*	E3.........SysAD[4]	K18........ Vcc	R20..... SysAD[23]	W4........ Vcc
A4 .... SysAD[32]	C4 .... SysAD[34]	E4........ Vcc	K19..... SysAD[11]	R21........ Vss	W5..............Int*[5]
A5 .... Vss	C5 ....ScDCE*[1]	E18........ Vcc	K20..... SysAD[43]	T1 .... SysAD[16]	W6..............Int*[4]
A6 ......ScCWE*[1]	C6 ....ScDCE*[0]	E19...... ScWord[1]	K21..... SysAD[13]	T2 ....SysADC[0]	W7..............Int*[1]
A7 .... Vss	C7 ......ScCWE*[0]	E20...... ScWord[0]	L1 .... Vss	T3 ....SysADC[2]	W8..........Reserved
A8 .... VccOK	C8 ....ScTCE*	E21........ Vss	L2 .... SysAD[58]	T4 .... Vss	W9..........Reserved
A9 .... Vss	C9 ....ModeIn	F1 ....SysAD[8]	L3 .... SysAD[28]	T18........ Vss	W10........Reserved
A10 ....SysClock	C10 .... Reserved	F2 ....SysAD[38]	L4 Vcc	T19..... SysAD[19]	W11......... ValidIn*
A11 .... Vss	C11 .... VssP	F3 ....SysAD[6]	L18 Vcc	T20..... SysAD[51]	W12.........ScDOE*
A12..... ScLine[15]	C12 .... Reserved	F4 .... Vss	L19 ..... SysAD[45]	T21..... SysAD[21]	W13.....SysCmd[7]
A13 .... Vss	C13..... ScLine[13]	F18........ Vss	L120 ... SysAD[63]	U1........ Vss	W14.....SysCmd[4]
A14..... ScLine[12]	C14..... ScLine[11]	F19 ....SysAD[1]	L21 .... Vss	U2.......SysADC[4]	W15.....SysCmd[1]
A15 .... Vss	C15 .... ScLine[8]	F20 ......SysAD[33]	M1 ...... SysAD[26]	U3.......SysADC[6]	W16....SysADC[7]
A16 .... ScLine[7]	C16 .... ScLine[5]	F21 ....SysAD[3]	M2 ...... SysAD[56]	U4........ Vcc	W17....SysADC[5]
A17 .... Vss	C17 .... ScLine[4]	G1........ Vss	M3 ...... SysAD[24]	U18........ Vcc	W18.....SysAD[47]
A18 .... ScLine[2]	C18 .... ScLine[0]	G2........SysAD[10]	M4 Vcc	U19..... SysAD[17]	W19......BigEndian
A19 .... Vss	C19 .... Reset*	G3........SysAD[40]	M18........ Vcc	U20..... SysAD[49]	W20........ Vcc
A20 Vcc	C20 Vcc	G4........ Vcc	M19 .... SysAD[29]	U21........ Vss	W21........ Vss
A21 .... Vss	C21 .... Vss	G18........ Vcc	M20 .... SysAD[61]	V1........ Vcc	Y1........ Vcc
B1 Vcc	D1 Vcc	G19......SysAD[35]	M21 .... SysAD[31]	V2........ Vcc	Y2........ Vcc
B2 Vcc	D2 Vcc	G20........SysAD[5]	N1........ Vss	V3........ Vcc	Y3........ Vcc
B3 Vcc	D3 Vcc	G21........ Vss	N2....... SysAD[54]	V4........ Vss	Y4........... Release*
B4 .... SysAD[2]	D4 .... Vss	H1........SysAD[42]	N3....... SysAD[22]	V5................NMI*	Y5...............Int*[3]
B5 .... SysAD[0]	D5 Vcc	H2........SysAD[44]	N4........ Vss	V6........ Vss	Y6...............Int*[2]
B6 ....ScTOE*	D6 .... Vss	H3........SysAD[12]	N18........ Vss	V7........ Vcc	Y7............. ScValid
B7 .... ScCLR*	D7 Vcc	H4........ Vcc	N19..... SysAD[27]	V8........ Vcc	Y8...........Reserved
B8 .... ScTDE*	D8 Vcc	H18........ Vcc	N20..... SysAD[59]	V9........ Vss	Y9...........Reserved
B9 ...... ModeClock	D9 .... Vss	H19........SysAD[7]	N21........ Vss	V10........ Vcc	Y10.........Reserved
B10 .... Reserved	D10 Vcc	H20......SysAD[39]	P1........ SysAD[50]	V11........ Vcc	Y11..........ExtReq*
B11 .... Reserved	D11 ....VccP	H21......SysAD[37]	P2........ SysAD[52]	V12........ Vcc	Y12.......... RdRdy*
B12 ....NC	D12 Vcc	J1........ Vss	P3........ SysAD[20]	V13........ Vss	Y13......SysCmd[8]
B13 ..... ScLine[14]	D13 .... Vss	J2.........SysAD[46]	P4........ Vcc	V14........ Vcc	Y14......SysCmd[5]
B14 ..... ScLine[10]	D14 Vcc	J3.........SysAD[14]	P18........ Vcc	V15........ Vcc	Y15......SysCmd[3]
B15 .... ScLine[9]	D15 Vcc	J4........ Vss	P19...... SysAD[25]	V16........ Vss	Y16......SysCmd[0]
B16 .... ScLine[6]	D16 .... Vss	J18........ Vss	P20...... SysAD[57]	V17........ Vcc	Y17........ SysCmdP
B17 .... ScLine[3]	D17 Vcc	J19.........SysAD[9]	P21...... SysAD[55]	V18........ Vss	Y18..... SysADC[1]
B18 .... ScLine[1]	D18 .... Vss	J20.......SysAD[41]	R1 .... Vss	V19........ Vcc	Y19......SysAD[15]
B19 Vcc	D19 Vcc	J21........ Vss	R2 .... SysAD[18]	V20........ Vcc	Y20........ Vcc
B20 Vcc	D20 Vcc	K1........SysAD[60]	R3 .... SysAD[48]	V21........ Vcc	Y21........ Vcc
B21 Vcc	D21 Vcc	K2........SysAD[30]	R4 Vcc	W1........ Vss	AA1........ Vss

Continued on next page

Location.......Name	Location.......Name	Location ...... Name	Location....... Name	Location....... Name	Location ...... Name
AA2 Vcc	AA7 .... Vss	AA12.......ScMatch	AA17 ....Vss		
AA3 .... Vss	AA8 .... Reserved	AA13..............Vss	AA18 ..SysADC[3]		
AA4 ...... ValidOut*	AA9 .... Vss	AA14...SysCmd[6]	AA19 ....Vss		
AA5.................Vss	AA10..........WrRdy*	AA15................Vss	AA20.............Vcc		
AA6............Int*[0]	AA11................Vss	AA16...SysCmd[2]	AA21.............Vss		

(2) μPD30500A

Location.......Name	Location.......Name	Location...... Name	Location....... Name	Location....... Name	Location...... Name
A1 ...... Vss	C1 ...... Vss	E1 ...... Vss	K3 ...... SysAD[62]	R18 ...... VccIO	W2 ...... VccIO
A2 ...... VccIO	C2 ...... VccIO	E2 ...... SysAD[36]	K4 ...... VccIO	R19 ...... SysAD[53]	W3 ...... VccIO
A3 ...... Vss	C3 ...... ColdReset*	E3 ...... SysAD[4]	K18 ...... VccIO	R20 ...... SysAD[23]	W4 ...... VccIO
A4 ...... SysAD[32]	C4 ...... SysAD[34]	E4 Vcc	K19 ...... SysAD[11]	R21 ...... Vss	W5 ...... Int*[5]
A5 ...... Vss	C5 ...... ScDCE*[1]	E18 Vcc	K20 ...... SysAD[43]	T1 ...... SysAD[16]	W6 ...... Int*[4]
A6 ...... ScCWE*[1]	C6 ...... ScDCE*[0]	E19 ...... ScWord[1]	K21 ...... SysAD[13]	T2 ...... SysADC[0]	W7 ...... Int*[1]
A7 ...... Vss	C7 ...... ScCWE*[0]	E20 ...... ScWord[0]	L1 ...... Vss	T3 ...... SysADC[2]	W8 ...... Vss
A8 ...... VccOK	C8 ...... ScTCE*	E21 ...... Vss	L2 ...... SysAD[58]	T4 ...... Vss	W9 ...... Vss
A9 ...... Vss	C9 ...... ModeIn	F1 ...... SysAD[8]	L3 ...... SysAD[28]	T18 ...... Vss	W10 Vcc
A10 ...... SysClock	C10 ...... NC	F2 ...... SysAD[38]	L4 Vcc	T19 ...... SysAD[19]	W11 ...... ValidIn*
A11 ...... Vss	C11 ...... VssP	F3 ...... SysAD[6]	L18 Vcc	T20 ...... SysAD[51]	W12 ...... ScDOE*
A12 ...... ScLine[15]	C12 ...... Vss	F4 ...... Vss	L19 ...... SysAD[45]	T21 ...... SysAD[21]	W13 ...... SysCmd[7]
A13 ...... Vss	C13 ...... ScLine[13]	F18 ...... Vss	L120 ... SysAD[63]	U1 ...... Vss	W14 ...... SysCmd[4]
A14 ...... ScLine[12]	C14 ...... ScLine[11]	F19 ...... SysAD[1]	L21 ...... Vss	U2 ...... SysADC[4]	W15 ...... SysCmd[1]
A15 ...... Vss	C15 ...... ScLine[8]	F20 ...... SysAD[33]	M1 ...... SysAD[26]	U3 ...... SysADC[6]	W16 ...... SysADC[7]
A16 ...... ScLine[7]	C16 ...... ScLine[5]	F21 ...... SysAD[3]	M2 ...... SysAD[56]	U4 Vcc	W17 ...... SysADC[5]
A17 ...... Vss	C17 ...... ScLine[4]	G1 ...... Vss	M3 ...... SysAD[24]	U18 Vcc	W18 ...... SysAD[47]
A18 ...... ScLine[2]	C18 ...... ScLine[0]	G2 ...... SysAD[10]	M4 ...... VccIO	U19 ...... SysAD[17]	W19 ...... BigEndian
A19 ...... Vss	C19 ...... Reset*	G3 ...... SysAD[40]	M18 ...... VccIO	U20 ...... SysAD[49]	W20 ...... VccIO
A20 ...... VccIO	C20 ...... VccIO	G4 ...... VccIO	M19 ...... SysAD[29]	U21 ...... Vss	W21 ...... Vss
A21 ...... Vss	C21 ...... Vss	G18 ...... VccIO	M20 ...... SysAD[61]	V1 Vcc	Y1 ...... VccIO
B1 ...... VccIO	D1 Vcc	G19 ...... SysAD[35]	M21 ...... SysAD[31]	V2 Vcc	Y2 ...... VccIO
B2 ...... VccIO	D2 Vcc	G20 ...... SysAD[5]	N1 ...... Vss	V3 Vcc	Y3 ...... VccIO
B3 ...... VccIO	D3 Vcc	G21 ...... Vss	N2 ...... SysAD[54]	V4 ...... Vss	Y4 ...... Release*
B4 ...... SysAD[2]	D4 ...... Vss	H1 ...... SysAD[42]	N3 ...... SysAD[22]	V5 ...... NMI*	Y5 ...... Int*[3]
B5 ...... SysAD[0]	D5 Vcc	H2 ...... SysAD[44]	N4 ...... Vss	V6 ...... Vss	Y6 ...... Int*[2]
B6 ...... ScTOE*	D6 ...... Vss	H3 ...... SysAD[12]	N18 ...... Vss	V7 Vcc	Y7 ...... ScValid
B7 ...... ScCLR*	D7 ...... VccIO	H4 Vcc	N19 ...... SysAD[27]	V8 ...... VccIO	Y8 ...... Vss
B8 ...... ScTDE*	D8 Vcc	H18 Vcc	N20 ...... SysAD[59]	V9 ...... Vss	Y9 ...... Vss
B9 ...... ModeClock	D9 ...... Vss	H19 ...... SysAD[7]	N21 ...... Vss	V10 Vcc	Y10 ...... Vss
B10 ...... Vss	D10 ...... VccIO	H20 ...... SysAD[39]	P1 ...... SysAD[50]	V11 ...... VccIO	Y11 ...... ExtReq*
B11 ...... Vss	D11 ...... VccP	H21 ...... SysAD[37]	P2 ...... SysAD[52]	V12 Vcc	Y12 ...... RdRdy*
B12 ...... Vss	D12 Vcc	J1 ...... Vss	P3 ...... SysAD[20]	V13 ...... Vss	Y13 ...... SysCmd[8]
B13 ...... ScLine[14]	D13 ...... Vss	J2 ...... SysAD[46]	P4 Vcc	V14 ...... VccIO	Y14 ...... SysCmd[5]
B14 ...... ScLine[10]	D14 ...... VccIO	J3 ...... SysAD[14]	P18 Vcc	V15 Vcc	Y15 ...... SysCmd[3]
B15 ...... ScLine[9]	D15 Vcc	J4 ...... Vss	P19 ...... SysAD[25]	V16 ...... Vss	Y16 ...... SysCmd[0]
B16 ...... ScLine[6]	D16 ...... Vss	J18 ...... Vss	P20 ...... SysAD[57]	V17 ...... VccIO	Y17 ...... SysCmdP
B17 ...... ScLine[3]	D17 ...... VccIO	J19 ...... SysAD[9]	P21 ...... SysAD[55]	V18 ...... Vss	Y18 ...... SysADC[1]
B18 ...... ScLine[1]	D18 ...... Vss	J20 ...... SysAD[41]	R1 ...... Vss	V19 Vcc	Y19 ...... SysAD[15]
B19 ...... VccIO	D19 Vcc	J21 ...... Vss	R2 ...... SysAD[18]	V20 Vcc	Y20 ...... VccIO
B20 ...... VccIO	D20 Vcc	K1 ...... SysAD[60]	R3 ...... SysAD[48]	V21 Vcc	Y21 ...... VccIO
B21 ...... VccIO	D21 Vcc	K2 ...... SysAD[30]	R4 ...... VccIO	W1 ...... Vss	AA1 ...... Vss
Continued on next page					

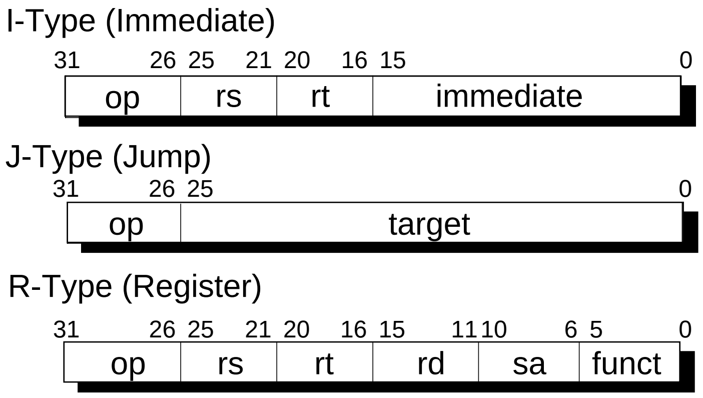
Location.......Name	Location.......Name	Location ...... Name	Location....... Name	Location....... Name	Location ...... Name
AA2 ....VccIO	AA7 .... Vss	AA12.......ScMatch	AA17 ....Vss		
AA3 .... Vss	AA8 .... Vss	AA13..............Vss	AA18 ..SysADC[3]		
AA4 ...... ValidOut*	AA9 .... Vss	AA14...SysCmd[6]	AA19 ....Vss		
AA5.................Vss	AA10..........WrRdy*	AA15................Vss	AA20 .... VccIO		
AA6...........Int*[0]	AA11................Vss	AA16...SysCmd[2]	AA21.............Vss		

Chapter 3 CPU Instruction Set Summary

The V_R5000 processor executes the MIPS IV instruction set, which is a superset of the MIPS III instruction set and is backward compatible. Each CPU instruction consists of a single 32-bit word, aligned on a word boundary. There are three instruction formats—immediate (I-type), jump (J-type), and register (R-type). The use of a small number of instruction formats simplifies instruction decoding, allowing the compiler to synthesize more complicated (and less frequently used) operations and addressing modes from these three formats as needed.

A summary of the MIPS IV instruction set additions is listed along with a brief explanation of each instruction. For more information on the MIPS IV instruction set, refer to **V_R5000, V_R10000 User's Manual Instruction**.

There are three types of instruction types as shown in Figure 3-1.



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

Figure 3-1 CPU Instruction Formats

In the MIPS architecture, coprocessor instructions are implementation-dependent.

3.1 Load and Store Instructions

Load and store are immediate (I-type) instructions that move data between memory and the general registers. The only addressing mode that integer load and store instructions directly support is *base register plus 16-bit signed immediate offset*. Floating point load and store instructions also support an indexed addressing, register+ register, addressing mode.

3.1.1 Scheduling a Load Delay Slot

In the V_R5000 processor, the instruction immediately following a load instruction can use the contents of the loaded register, however in such cases hardware interlocks insert additional real cycles. Consequently, scheduling load delay slots can be desirable, both for performance and V_R-Series processor compatibility. However, the scheduling of load delay slots is not absolutely required.

3.1.2 Defining Access Types

Access type indicates the size of a V_R5000 processor data item to be loaded or stored, set by the load or store instruction opcode.

Regardless of access type or byte ordering (endianness), the address given specifies the low-order byte in the addressed field. For a big-endian configuration, the low-order byte is the most-significant byte; for a little-endian configuration, the low-order byte is the least-significant byte.

The access type, together with the three low-order bits of the address, define the bytes accessed within the addressed doubleword (shown in Table 3-1). Only the combinations shown in Table 3-1 are permissible; other combinations cause address error exceptions.

Table 3-1   Byte Access within a Doubleword

Access Type Mnemonic (*Value*)	**Low Order Address Bits**			**Bytes Accessed**															
				Big endian (63-----------31------------0) Byte								**Little endian (63-----------31------------0) Byte**							
	2	**1**	**0**																
Doubleword (*7*)	0	0	0	0	1	2	3	4	5	6	7	7	6	5	4	3	2	1	0
Septibyte (*6*)	0	0	0	0	1	2	3	4	5	6			6	5	4	3	2	1	0
	0	0	1		1	2	3	4	5	6	7	7	6	5	4	3	2	1	
Sextibyte (*5*)	0	0	0	0	1	2	3	4	5					5	4	3	2	1	0
	0	1	0			2	3	4	5	6	7	7	6	5	4	3	2		
Quintibyte (*4*)	0	0	0	0	1	2	3	4							4	3	2	1	0
	0	1	1				3	4	5	6	7	7	6	5	4	3			
Word (*3*)	0	0	0	0	1	2	3									3	2	1	0
	1	0	0					4	5	6	7	7	6	5	4				
Triplebyte (*2*)	0	0	0	0	1	2											2	1	0
	0	0	1		1	2	3									3	2	1	
	1	0	0					4	5	6			6	5	4				
	1	0	1						5	6	7	7	6	5					
Halfword (*1*)	0	0	0	0	1													1	0
	0	1	0			2	3									3	2		
	1	0	0					4	5					5	4				
	1	1	0							6	7	7	6						
Byte (*0*)	0	0	0	0															0
	0	0	1		1													1	
	0	1	0			2											2		
	0	1	1				3									3			
	1	0	0					4							4				
	1	0	1						5					5					
	1	1	0							6			6						
	1	1	1								7	7							

3.2 Computational Instructions

Computational instructions can be either in register (R-type) format, in which both operands are registers, or in immediate (I-type) format, in which one operand is a 16-bit immediate.

Computational instructions perform the following operations on register values:

- arithmetic
- logical
- shift
- multiply
- divide

These operations fit in the following four categories of computational instructions:

- ALU Immediate instructions
- three-Operand Register-Type instructions
- shift instructions
- multiply and divide instructions

3.2.1 64-bit Operations

The V_R5000 microprocessor is a 64-bit architecture which supports 32-bit operands. These operands must be sign extended. Thirty-two bit operand opcodes include all non-doubleword operations, such as: ADD, ADDU, SUB, SUBU, ADDI, SLL, SRA, SLLV, etc. The result of operations that use incorrect sign-extended 32-bit values is unpredictable. In addition, 32-bit data is stored sign-extended in a 64-bit register.

3.2.2 Cycle Timing for Multiply and Divide Instructions

MFHI and MFLO instructions are interlocked so that any attempt to read them before prior instructions complete delays the execution of these instructions until the prior instructions finish.

Table 3-2 gives the number of processor cycles (PCycles) required to resolve an interlock or stall between various multiply or divide instructions, and a subsequent MFHI or MFLO instruction.

Table 3-2 Multiply/Divide Instruction Latency and Repeat Rates

Instruction	Latency	Repeat Rate
MULT (32-bit × 16-bit)	4	3
MULT (32-bit × 32-bit)	5	4
MULTU	5	4
DIV	36	36
DIVU	36	36
DMULT	9	8
DMULTU	9	8
DDIV	68	68
DDIVU	68	68

3.2.3 Jump and Branch Instructions

Jump and branch instructions change the control flow of a program. All jump and branch instructions occur with a delay of one instruction: that is, the instruction immediately following the jump or branch (this is known as the instruction in the *delay slot*) always executes while the target instruction is being fetched from storage.

(1) Overview of Jump Instructions

Subroutine calls in high-level languages are usually implemented with Jump or Jump and Link instructions, both of which are J-type instructions. In J-type format, the 26-bit target address shifts left 2 bits and combines with the high-order 4 bits of the current program counter to form an absolute address.

Returns, dispatches, and large cross-page jumps are usually implemented with the Jump Register or Jump and Link Register instructions. Both are R-type instructions that take the 64-bit byte address contained in one of the general purpose registers.

(2) Overview of Branch Instructions

All branch instruction target addresses are computed by adding the address of the instruction in the delay slot to the 16-bit *offset* (shifts left 2 bits and is sign-extended to 64 bits). All branches occur with a delay of one instruction.

If a conditional branch is not taken, the instruction in the delay slot is nullified.

3.2.4 Special Instructions

Special instructions allow the software to initiate traps; they are always R-type. Exception instructions are extensions to the MIPS ISA.

3.2.5 Coprocessor Instructions

Coprocessor instructions perform operations in their respective coprocessors. Coprocessor loads and stores are I-type, and coprocessor computational instructions have coprocessor-dependent formats.

CP0 instructions perform operations specifically on the System Control Coprocessor registers to manipulate the memory management and exception handling facilities of the processor.

3.3 MIPS IV Instruction Set Additions

The V_R5000 Microprocessor runs the MIPS IV instruction set, which is a superset of the MIPS III instruction set and is backward compatible. The additions of these new instructions enables the MIPS architecture to compete in the high-end numeric processing market which has traditionally been dominated by vector architectures.

A set of compound multiply-add instructions has been added, taking advantage of the fact that the majority of floating point computations use the chained multiply-add paradigm. The intermediate multiply result is rounded before the addition is performed.

A register + register addressing mode for floating point loads and stores has been added which eliminates the extra integer add required in many array accesses. However, issuing of a Register + Register load causes a one cycle stall in the pipeline, which makes it useful only for compatibility with other MIPS IV implementations. Register + register addressing for integer memory operations is not supported.

A set of four conditional move operators allows floating point arithmetic ‘IF’ statements to be represented without branches. ‘THEN’ and ‘ELSE’ clauses are computed unconditionally and the results placed in a temporary register. Conditional move operators then transfer the temporary results to their true register. Conditional moves must be able to test both integer and floating point conditions in order to supply the full range of IF statements. Integer tests are performed by comparing a general register against a zero value.

Floating point tests are performed by examining the floating point condition codes. Since floating point conditional moves test the floating point condition code, the V_R5000 microprocessor provides 8 condition codes to give the compiler increased flexibility in scheduling the comparison and the conditional moves. Table 3-3 lists in alphabetical order the new instructions which comprise the MIPS IV instruction set.

Table 3-3 MIPS IV Instruction Set Additions and Extensions

Instruction	Definition
BC1F	Branch on FP Condition Code False
BC1T	Branch on FP Condition Code True
BC1FL	Branch on FP Condition Code False Likely
BC1TL	Branch on FP Condition Code True Likely
C.cond.fmt (cc)	Floating Point Compare
LDXC1	Load Double Word indexed to COP1
LWXC1	Load Word indexed to COP1
MADD.fmt	Floating Point Multiply-Add
MOVF	Move conditional on FP Condition Code False
MOVN	Move on Register Not Equal to Zero
MOVT	Move conditional on FP Condition Code True
MOVZ	Move on Register Equal to Zero
MOVF.fmt	FP Move conditional on Condition Code False
MOVN.fmt	FP Move on Register Not Equal to Zero
MOVT.fmt	FP Move conditional on Condition Code True

Table 3-3   MIPS IV Instruction Set Additions and Extensions (Continued)

Instruction	Definition
MOVZ.fmt	FP Move conditional on Register Equal to Zero
MSUB.fmt	Floating Point Multiply-Subtract
NMADD.fmt	Floating Point Negative Multipy-Add
NMSUB.fmt	Floating Point Negative Multiply-Subtract
PREFX[a]	Prefetch Indexed --- Register + Register
PREF[a]	Prefetch --- Register + Offset
RECIP.fmt	Reciprocal Approximation
RSQRT.fmt	Reciprocal Square Root Approximation
SDXC1	Store Double Word indexed to COP1
SWXC1	Store Word indexed to COP1

a. Prefetch is not implemented in the V$_R$5000 microprocessor and these instructions are treated as no-ops.

Table 3-4 lists the COP0 instructions for the V$_R$5000 processor. COP0 instructions are those which are not architecturally visible and are used by the kernel.

Table 3-4   V$_R$5000 COP0 Instrucitons

COP0 Instruction	Definition
ERET	Return from Exception
TLBP	Probe for TLB Entry
TLBR	Read Indexed TLB Entry
TLBWI	Write Indexed TLB Entry
TLBWR	Write Random TLB Entry
WAIT	Enter Standby Mode

3.3.1 Summary of Instruction Set Additions

The following is a brief description of the additions to the MIPS III instruction set. These additions comprise the MIPS IV instruction set.

(1) Indexed Floating Point Load

LWXC1 - Load word indexed to Coprocessor 1.

LDXC1 - Load doubleword indexed to Coprocessor 1.

The two Index Floating Point Load instructions are exclusive to the MIPS IV instruction set and transfer floating-point data types from memory to the floating point registers using register + register addressing mode. There are no indexed loads to general registers. The contents of the general register specified by the base is added to the contents of the general register specified by the index to form a virtual address. The contents of the word or doubleword specified by the effective address are loaded into the floating point register specified in the instruction.

The region bits (63:62) of the effective address must be supplied by the base. If the addition alters these bits an address exception occurs. Also, if the address is not aligned, an address exception occurs.

(2) Indexed Floating Point Store

SWXC1 - Store word indexed to Coprocessor 1.

SDXC1 - Store doubleword indexed to Coprocessor 1.

The two Index Floating Point Store instructions are exclusive to the MIPS IV instruction set and transfer floating-point data types from the floating point registers to memory using register + register addressing mode. There are no indexed loads to general registers. The contents of the general register specified by the base is added to the contents of the general register specified by the index to form a virtual address. The contents of the floating point register specified in the instruction is stored to the memory location specified by the effective address.

The region bits (63:62) of the effective address must be supplied by the base. If the addition alters these bits an address exception occurs. Also, if the address is not aligned, an address exception occurs.

(3) Prefetch

PREF - Register + offset format

PREFX - Register + register format

The two prefetch instructions are exclusive to the MIPS IV instruction set and allow the compiler to issue instructions early so the corresponding data can be fetched and placed as close as possible to the CPU. Each instruction contains a 5-bit ‘hint’ field which gives the coherency status of the line being prefetched. The line can be either shared, exclusive clean, or exclusive dirty. The contents of the general register specified by the base is added either to the 16 bit sign-extended offset or to the contents of the general register specified by the index to form a virtual address. This address together with the ‘hint’ field is sent to the cache controller and a memory access is initiated.

The region bits (63:62) of the effective address must be supplied by the base. If the addition alters these bits an address exception occurs. The prefetch instruction never generates TLB-related exceptions. The PREF instruction is considered a standard processor instruction while the PREFX instruction is considered a standard Coprocessor 1 instruction. The V_R5000 microprocessor does not implement prefetch and these instruction are executed as no-ops.

(4) Branch on Floating Point Coprocessor

BC1T - Branch on FP condition True

BC1F - Branch on FP condition False

BC1TL - Branch on FP condition True Likely

BC1FL - Branch on FP condition False Likely

The four branch instructions are upward compatible extensions of the Branch on Floating point Coprocessor instructions of the MIPS instruction set. The BC1T and BC1F instructions are extensions of MIPS I. BC1TL and BC1FL are extensions of MIPS III. These instructions test one of eight floating point condition codes. This encoding is downward compatible with previous MIPS architectures.

The branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit offset, shifted left two bits and sign-extended to 64 bits. If the contents of the floating point condition code specified in the instruction are equal to the test value, the target address is branched to with a delay of one instruction. If the conditional branch is not taken and the nullify delay bit in the instruction is set, the instruction in the branch delay slot is nullified.

(5) Integer Conditional Moves

MOVT - Move conditional on condition code true

MOVF - Move conditional on condition code false

MOVN - Move conditional on register not equal to zero

MOVZ - Move conditional on register equal to zero

The four integer move instructions are exclusive to the MIPS IV instruction set and are used to test a condition code or a general register and then conditionally perform an integer move. The value of the floating point condition code specified in the instruction by the 3-bit condition code specifier, or the value of the register indicated by the 5-bit general register specifier, is compared to zero. If the result indicates that the move should be performed, the contents of the specified source register is copied into the specified destination register.

(6) Floating Point Multiply-Add

MADD - Floating Point Multiply-Add

MSUB - Floating Point Multiply-Subtract

NMADD - Floating Point Negative Multiply-Add

NMSUB - Floating Point Negative Multiply-Subtract

These four instructions are exclusive to the MIPS IV instruction set and accomplish two floating point operations with one instruction. Each of these four instrucitons performs intermediate rounding.

(7) Floating Point Compare

C.cond.fmt - Compare the contents of two FPU registers

The contents of the two FPU source registers specified in the instruction are interpreted and arithmetically compared. A result is determined based on the comparison and the conditions specified in the instruction.

(8) Floating Point Conditional Moves

MOVT.fmt - Floating Point Conditional Move on condition code true

MOVF.fmt - Floating Point Conditional Move on condition code false

MOVN.fmt - Floating Point Conditional Move on register not equal to zero

MOVZ.fmt - Floating Point Conditional Move on register equal to zero

The four floating point conditional move instructions are exclusive to the MIPS IV instruction set and are used to test a condition code or a general register and then conditionally perform a floating point move. The value of the floating point condition code specified by the 3-bit condition code specifier, or the value of the register indicated by the 5-bit general register specifier, is compared to zero. If the result indicates that the move should be performed, the contents of the specified source register is copied into the specified destination register. All of these conditional floating point move operations are non-arithmetic. Consequently, no IEEE 754 exceptions occur as a result of these instructions.

(9) Reciprocal's

RECIP.fmt - Reciprocal

RSQRT.fmt - Reciprocal Square Root

The reciprocal instruction performs a reciprocal on a floating point value. The reciprocal of the value in the floating point source register is placed in a destination register.

The reciprocal square root instruction performs a reciprocal square root on a floating point value. The reciprocal of the positive square root of a value in the floating point source register is placed in a destination register.

The V_R5000 meets full IEEE accuracy for the RECIP and RSQRT instructions.

On the V_R5000 microprocessor, the RECIP instruction has the same latency as a DIV instruction, but a RSQRT is faster than a SQRT followed by a RECIP.

3.3.2 Cycle Timing for Floating Point Instrucitons

Table 3-5 Floating Point Operations

Opcode	Latency	Repeat
ADD (sngl/dbl)	4	1
SUB (sngl/dbl)	4	1
MULT (sngl/dbl)	4/5	1/2
MADD (sngl/dbl)	4/5	1/2
MSUB (sngl/dbl)	4/5	1/2
NMADD (sngl/dbl)	4/5	1/2
NMSUB (sngl/dbl)	4/5	1/2
DIV (sngl/dbl)	21/36	19/34
SQRT (sngl/dbl)	21/36	19/34
RECIP (sngl/dbl)	21/36	19/34
RSQRT (sngl/dbl)	38/68	36/66
ROUND.W (sngl/dbl)	4/4	1/1
ROUND.L (sngl/dbl)	4/4	1/1
TRUNC.W (sngl/dbl)	4/4	1/1
TRUNC.L (sngl/dbl)	4/4	1/1
CEIL.W (sngl/dbl)	4/4	1/1
CEIL.L (sngl/dbl)	4/4	1/1
FLOOR.W (sngl/dbl)	4/4	1/1
FLOOR.L (sngl/dbl)	4/4	1/1
CVT.S.D	4	1
CVT.S.W	6	3
CVT.S.L	6	3
CVT.D.S	4	1
CVT.D.W	4	1
CVT.D.L	4	1
CVT.W (sngl/dbl)	4	1
CVT.L (sngl/dbl)	4	1
CMP (sngl/dbl)	1	1
MOV (sngl/dbl)	1	1
MOVC (sngl/dbl)	1	1
ABS (sngl/dbl)	1	1
NEG (sngl/dbl)	1	1
LWC1, LWXC1	2	1

Table 3-5 Floating Point Operations (Continued)

Opcode	Latency	Repeat
LDC1, LDXC1	2	1
SWC1, SWXC1	2	1
SDC1, SDXC1	2	1
MTC1, DMTC1	2	1
MFC1, DMFC1	2	1
CTC1	3	3
CFC1	2	2
BC1T, BC1TL	1	1
BC1F, BC1FL	1	1

3.4 The Cache Instruction

The CACHE instruction in the V_R5000 microprocessor is implemented as follows:

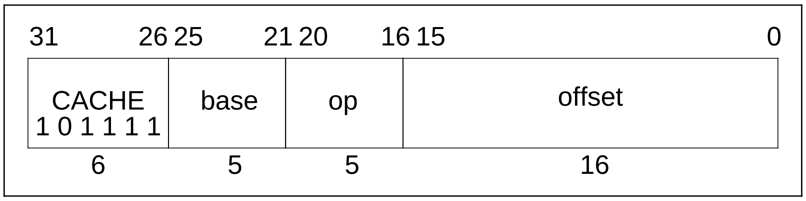


Figure 3-2 V_R5000 CACHE Instruction Format

Format:

CACHE op, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. The virtual address is translated to a physical address using the TLB, and the 5-bit sub-opcode specifies a cache operation for that address.

If CP0 is not usable (User or Supervisor mode) the CP0 enable bit in the *Status* register is clear, and a coprocessor unusable exception is taken. The operation of this instruction on any operation/cache combination not listed below, or on a secondary cache when none is present, is undefined. The operation of this instruction on uncached addresses is also undefined.

The Index operation uses part of the virtual address to specify a cache block.

For a primary cache of 32 KB with 32 bytes per tag, $vAddr_{13:5}$ specifies the block. In addition, $vAddr_{14}$ specifies which cache set to operate on.

For a secondary cache of $2^{CACHEBITS}$ bytes with $2^{LINEBITS}$ bytes per tag, $pAddr_{CACHEBITS ... LINEBITS}$ specifies the block.

Index Load Tag also uses $vAddr_{LINEBITS... 3}$ to select the doubleword for reading parity. When the *CE* bit of the *Status* register is set, Hit WriteBack, Hit WriteBack Invalidate, Index WriteBack Invalidate, and Fill also use $vAddr_{LINEBITS ... 3}$ to select the doubleword that has its parity modified. This operation is performed unconditionally.

The Hit operation accesses the specified cache as normal data references, and performs the specified operation if the cache block contains valid data with the specified physical address (a hit). If the cache block is invalid or contains a different address (a miss), no operation is performed.

Write back from a primary cache goes to the secondary cache and to memory. If no secondary cache is present, the data goes to memory. Data comes from the primary data cache, if present, and is modified (it is marked Dirty). Otherwise the data comes from the secondary cache. The address to be written is specified by the cache tag and not the translated physical address.

TLB Refill and TLB Invalid exceptions can occur on any operation. For Index operations (where the physical address is used to index the cache but need not match the cache tag) unmapped addresses may be used to avoid TLB exceptions. This operation never causes TLB Modified or Virtual Coherency exceptions.

Bits 17...16 of the instruction specify the cache as follows:

Code	Name	Cache
0	I	primary instruction
1	D	primary data
2	--	Reserved
3	SD	secondary cache

Bits 20...18 (this value is listed under the **Code** column) of the instruction specify the operation as follows:

Code	Caches	Name	Operation
0	I	Index Invalidate	Set the cache state of the cache block to Invalid.
0	D	Index Writeback Invalidate	Examine the cache state of the primary data cache block at the index specified by the virtual address. If the state is Dirty, write the block back to the secondary cache (if present) and to memory. The address to write is taken from the primary cache tag. Set the cache state of primary cache block to Invalid.
0	S	Flash Invalidate	Flash Invalidate the entire secondary cache in one operation for tag RAMs which support this function.
1	All	Index Load Tag	Read the tag for the cache block at the specified index and place it iinto the *TagLo* and *TagHi* CP0 registers, ignoring any parity errors.
2	I, D	Index Store Tag	Write the tag for the cache block at the specified index from the *TagLo* and *TagHi* CP0 registers.
2	S	Index Store Tag	Write the tag for the cache block at the specified index with the tag value from the effective address generated by the CACHE instruction and the valid bit from the TagLo CP0 register.
3	D	Create Dirty Exclusive	This operation is used to avoid loading data needlessly from secondary cache or memory when writing new contents into an entire cache block. If the cache block does not contain the specified address, and the block is dirty, write it back to the secondary cache (if present) and to memory. In all cases, set the cache block tag to the specified physical address, set the cache state to Dirty Exclusive.
4	I,D	Hit Invalidate	If the cache block contains the specified address, mark the cache block invalid.
5	D	Hit Writeback Invalidate	If the cache block contains the specified address, write the data back if it is dirty, and mark the cache block invalid.
5	S	Page Invalidate	The processor will generate a page invalidate by doing a burst of 128 line invalidates to the secondary cache at the page specified by the effective address generated by the CACHE instruction, which must be page-aligned. Interrupts are deferred during page invalidates.
5	I	Fill	Fill the primary instruction cache block from secondary cache or memory.
6	D	Hit Writeback	If the cache block contains the specified address, and its state is Dirty, write back the data and clear the state to not Dirty.
6	I	Hit Writeback	If the cache block contains the specified address, data is written back unconditionally.

Operation:

```
32, 64   T:   $vAddr \leftarrow ((offset_{15})^{48} \| offset_{15...0}) + GPR[base]$
              $(pAddr, uncached) \leftarrow AddressTranslation (vAddr, DATA)$
              CacheOp (op, vAddr, pAddr)
```

Exceptions:

Coprocessor unusable exception

3.5 Implementation Specific Instructions

Some of the V_R5000 instructions are implementation specific and therefore are not part of the MIPS IV Instruction Set. These are coprocessor instructions that perform operations in their respective coprocessors. Coprocessor loads and stores are I-type, and coprocessor computational instructions have coprocessor-dependent formats.

3.5.1 Implementation Specific CP0 Instructions

ERET Exception Return

31 26	25	24 6	5 0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	ERET 0 1 1 0 0 0
6	1	19	6

Format:

ERET

Description:

ERET is the V_R5000 instruction for returning from an interrupt, exception, or error trap. Unlike a branch or jump instruction, ERET does not execute the next instruction.

ERET must not itself be placed in a branch delay slot.

If the processor is servicing an error trap ($SR_2 = 1$), then load the PC from the ErrorEPC and clear the ERL bit of the Status register (SR_2). Otherwise ($SR_2 =0$), load the PC from the *EPC*, and clear the *EXL* bit of the Status register (SR_1).

An ERET executed between a LL and AC also causes the SC to fail.

Operation:

```
T:   if $SR_2$ = 1 then
            PC ← ErrorEPC
            SR ← $SR_{31..3}$ || 0 || $SR_{1..0}$
     else
         PC ← EPC
         SR ← $SR_{31..2}$  0  $SR_0$
       endif
       LLbit ← 0
```

Exceptions:

Coprocessor unusable exception.

TLBR Read Indexed TLB Entry

31 26	25	24 6	5 0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	TLBR 0 0 0 0 0 1
6	1	19	6

Format:

TLBR

Description:

The *EntryHi* and *EntryLo* registers are loaded with the contents of the TLB entry pointed at by the contents of the TLB *Index* register. The operation is invalid (and the results are unspecified) if the contents of the TLB *Index* register are greater than the number of TLB entries in the processor.

The *G* bit (which controls ASID matching) read from the TLB is written into both of the *EntryLo0* and *EntryLo1* registers.

Operation:

```
T:  PageMask  $\leftarrow$ TLB[$\text{Index}_{5..0}$]$_{255..192}$
    EntryHi   $\leftarrow$ TLB[$\text{Index}_{5..0}$]$_{191..128}$ and not TLB[$\text{Index}_{5..0}$]$_{255..192}$
    EntryLo1  $\leftarrow$ TLB[$\text{Index}_{5..0}$]$_{127..65}$ || TLB[$\text{Index}_{5..0}$]$_{140}$
    EntryLo0  $\leftarrow$ TLB[$\text{Index}_{5..0}$]$_{63..1}$ || TLB[$\text{Index}_{5..0}$]$_{140}$
```

Exceptions:

Coprocessor unusable exception.

TLBP Probe TLB For Matching Entry

31 26	25	24 6	5 0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	TLBP 0 0 1 0 0 0
6	1	19	6

Format:

TLBP

Description:

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 high-order bit of the *Index* register is set.

The architecture does not specify the operation of memory references associated with the instruction immediately after a TLBP instruction, nor is the operation specified if more than one TLB entry matches.

Operation:

```
T:   Index ← 1 || $0^{31}$
      For i in 0..TLBEntries - 1
          if (TLB[i]$_{167..141}$ and not ($0^{15}$ || TLB[i]$_{216..205}$))
          = (EntryHi$_{39..13}$ and not ($0^{15}$ || TLB[i]$_{216..205}$)) and
          (TLB[i]$_{140}$ or (TLB[i]$_{135..128}$ = EntryHi7..0)) then
                Index ← $0^{26}$ || i$_{5..0}$
          endif
       endfor
```

Exceptions:

Coprocessor unusable exception.

TLBWI Write Indexed TLB Entry

31 26	25	24 6	5 0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	TLBWI 0 0 0 0 1 0
6	1	19	6

Format:

TLBWI

Description:

The TLB entry pointed at by the contents of the TLB Index register is loaded with the contents of the EntryHi and EntryLo registers.

The *G* bit of the selected TLB entry is written with the logical AND of the *G* bits in the EntryLo0 and EntryLo1 registers.

The operation is invalid (and the results are unspecified) if the contents of theTLB Index register are greater than the number of TLB entries in the processor.

Operation:

```
T:    TLB[Index$_{5..0}$]  ←
                    EntryHi[39:25] || (EntryHi[24:13] and not PageMask) || EntryLo1
|| EntryLo0
```

Exceptions:

Coprocessor unusable exception.

TLBWR Write Random TLB Entry

31 26	25	24 6	5 0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	TLBWR 0 0 0 1 1 0
6	1	19	6

Format:

TLBWR

Description:

The TLB entry pointed to by the contents of the TLB *Random* register is loaded with the contents of the *EntryHi* and *EntryLo* registers.

The *G* bit of the selected TLB entry is written with the logical AND of the *G* bits in the *EntryLo0* and *EntryLo1* registers.

Operation:

```
T:    TLB[$\text{Random}_{5..0}$]  ←
                    EntryHi[39:25] || (EntryHi[25:13] and not PageMask) || EntryLo1
|| EntryLo0
```

Exceptions:

Coprocessor unusable exception.

DMTC0 Doubleword Move To System Control Coprocessor

31 26	25 21	20 16	15 11	10 0
COP0 0 1 0 0 0 0	DMT 0 0 1 0 1	rt	rd	0 0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

Format:

DMTC0 rt, rd

Description:

The contents of general register *rt* are loaded into coprocessor register *rd* of CP0.

This operation is defined in kernel mode regardless of the setting of the Status.KX bit. Execution of this instruction in supervisor mode with Status.SX = 0 or in user mode with UX = 0, causes a reserved instruction exception.

All 64-bits of the coprocessor 0 register are written from the general register source. The operation of DMTC0 on a 32-bit coprocessor 0 register is undefined.

Because the state of the virtual address translation system may be altered by this instruction, the operation of load instructions, store instructions, and TLB operations immediately prior to and after this instruction are undefined.

Operation:

```
T:      data ← GPR[rt]

T+1:    CPR[0,rd] ← data
```

Exceptions:

Coprocessor unusable exception.

Reserved instruction exception for supervisor mode with Status.SX = 0 or user mode with Status.UX = 0.

MTC0 — Move To System Control Coprocessor

31 26	25 21	20 16	15 11	10 0
COP0 0 1 0 0 0 0	MT 0 0 1 0 0	rt	rd	0 0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

Format:

MTC0 rt, rd

Description:

The contents of general register *rt* are loaded into coprocessor register *rd* of CP0.

Because the state of the virtual address translation system may be altered by this instruction, the operation of load instructions, store instructions, and TLB operations immediately prior to and after this instruction are undefined.

Operation:

```
T:       data ← GPR[rt]

T+1:    CPR[0,rd] ← data
```

Exceptions:

Coprocessor unusable exception.

DMFC0 Doubleword Move From System Control Coprocessor

31 26	25 21	20 16	15 11	10 0
COP0 0 1 0 0 0 0	DMF 0 0 0 0 1	rt	rd	0 0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

Format:

DMFC0 rt, rd

Description:

The contents of coprocessor register *rd* of the CP0 are loaded into general register *rt.*

This operation is defined in kernel mode regardless of the setting of the Status.KX bit. Execution of this instruction in supervisor mode with Status.SX = 0 or in user mode with UX = 0, causes a reserved instruction exception.

All 64-bits of the general register destination are written from the coprocessor register source. The operation of DMFC0 on a 32-bit coprocessor 0 register is undefined.

Operation:

```
T:      data ← GPR[0,rd]

T+1:    CPR[rt] ← data
```

Exceptions:

Coprocessor unusable exception.

Reserved instruction exception for supervisor mode with Status.SX = 0 or user mode with Status.UX = 0.

WAIT Enter Standby Mode

31 26	25	24 6	5 0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	WAIT 1 0 0 0 0 0
6	1	19	6

Format:

WAIT

Description:

The WAIT instruction is used to put the CPU into Standby Mode. In Standby Mode, most of the internal clocks are shut down which freezes the pipeline and reduces power consumption. See **Chapter 18 Standby Mode Operation** for more details.

Operation:

```
T:      if SysAD bus is idle then
            Enter Standby Mode
        endif
```

Exceptions:

Coprocessor unusable exception.

Chapter 4 V_R5000 Processor Pipeline

The V_R5000 processor has a five-stage instruction pipeline. Each stage takes one PCycle (one cycle of PClock, which runs at a multiple of the frequency of SysClock). Thus, the execution of each instruction takes at least five PCycles. An instruction can take longer—for example, if the required data is not in the cache, the data must be retrieved from main memory.

Once the pipeline has been filled, five instructions can be executed simultaneously. Figure 4-1 shows the five stages of the instruction pipeline.

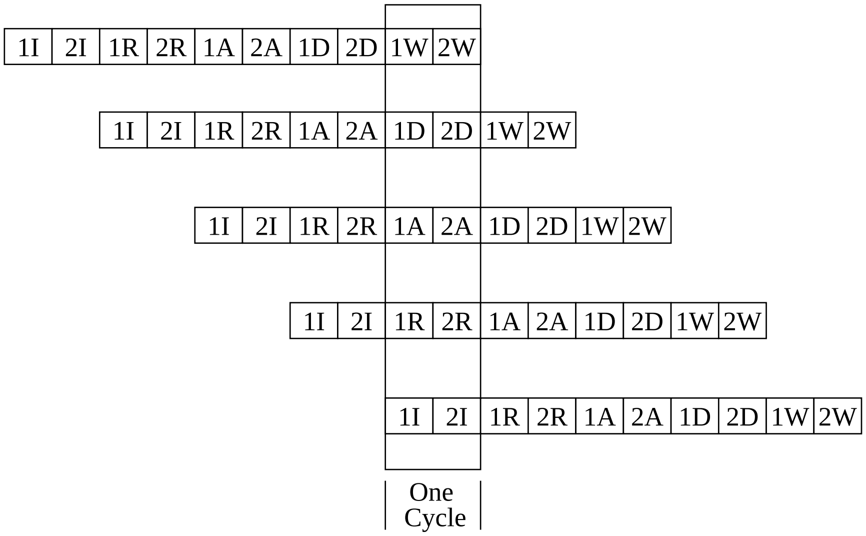

1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
One Cycle

Figure 4-1 Instruction Pipeline Stages

4.1 Instruction Pipeline Stages

- 1I - Instruction Fetch, Phase One
- 2I - Instruction Fetch, Phase Two
- 1R - Register Fetch, Phase One
- 2R - Register Fetch, Phase Two
- 1A - Execution, Phase One
- 2A - Execution, Phase Two
- 1D - Data Fetch, Phase One
- 2D - Data Fetch, Phase Two
- 1W - Write Back, Phase One
- 2W - Write Back, Phase Two

1I - Instruction Fetch, Phase One

During the 1I phase, the following occurs:

- Branch logic selects an instruction address and the instruction cache fetch begins.
- The instruction translation lookaside buffer (ITLB) begins the virtual-to-physical address translation.

2I - Instruction Fetch, Phase Two

The instruction cache fetch and the virtual-to-physical address translation continues.

1R - Register Fetch, Phase One

During the 1R phase, the following occurs:

- The instruction cache fetch is completed.
- The instruction cache tag is checked against the page frame number obtained from the ITLB

2R - Register Fetch, Phase Two

During the 2R phase, one of the following occurs:

- The instruction decoder decodes the instruction.
- Any required operands are fetched from the register file.
- Determine whether instruction is issued or delayed depending on interlock conditions.

1A - Execution - Phase One

During the 1A phase, one of the following occurs:

- Calculate branch address (if applicable).
- Any result from the A or D stages are bypassed
- The ALU starts an integer operation.
- The ALU calculates the data virtual address for load and store instructions.
- The ALU determines whether the branch condition is true.

2A - Execution - Phase Two

During the 2A phase, one of the following occurs:

- The integer operation begun in the 1A phase completes.

- Data cache address decode.
- Store data is shifted to the specified byte positions.
- The DTLB begins the data virtual to physical address translation.

1D - Data Fetch - Phase One

During the 1D phase, one of the following occurs:

- The DTLB data address translation completes.
- The JTLB virtual to physical address translation begins.
- Data cache access begins

2D - Data Fetch - Phase Two

- The data cache access completes. Data is shifted down and extended.
- The JTLB address translation completes.
- The data cache tag is checked against the PFN from the DTLB or JTLB for any data cache access.

1W - Write Back, Phase One

- This phase is used internally by the procesor to resolve all exceptions in preperation for the register write.

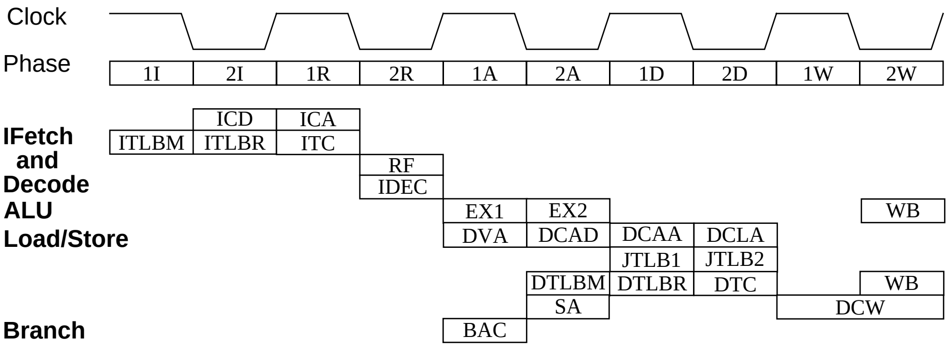
2W - Write Back, Phase Two

- For register-to-register and load instructions, the result is written back to the register file.

WB - Write Back

For register-to-register instructions, the instruction result is written back to the register file during the WB stage. Branch instructions perform no operation during this stage.

Figure 4-2 shows the activities occurring during each ALU pipeline stage, for load, store, and branch instructions.



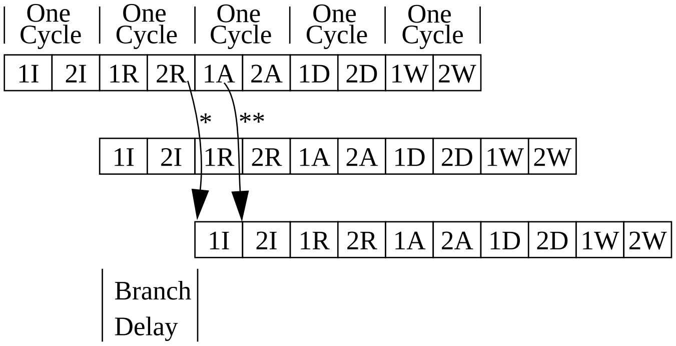
ICD	Instruction cache address decode	ICA	Instruction cache array access
ITLBM	Instruction address translation match	ITLBR	Instruction address translation read
ITC	Instruction tag check	RF	Register operand fetch
IDEC	Instruction address translation stage 2	EX1	Execute operation - phase 1
EX2	Execute operation - phase two	WB	Write back to register file
DVA	Data virtual address calculation	DCAD	Data cache address decode
DCAA	Data cache array access	DCLA	Data cache load align
JTLB1	JTLB address translation - phase 1	JTLB2	JTLB address translation - phase 2
DTLBM	Data address translation match	DTLBR	Data address translation read
DTC	Data tag check	SA	Store align
DCW	Data cache write	BAC	Branch address calculation

Figure 4-2 CPU Pipeline Activities

4.2 Branch Delay

The CPU pipeline has a branch delay of one cycle and a load delay of one cycles. The one-cycle branch delay is a result of the branch comparison logic operating during the 1A pipeline stage of the branch. This allows the branch target address calculated in the previous stage to be used for the instruction access in the following 1I phase.

Figure 4-3 illustrates the branch delay.


One Cycle
One Cycle
One Cycle
One Cycle
One Cycle
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
*
**
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
Branch Delay

* Branch and fall-through address calculated
** Address selection made

Figure 4-3 CPU Pipeline Branch Delay

4.3 Load Delay

The completion of a load at the end of the 2D pipeline stage produces an operand that is available for the 1A pipeline phase of the subsequent instruction following the load delay slot.

Figure 4-4 shows the load delay of two pipeline stages.

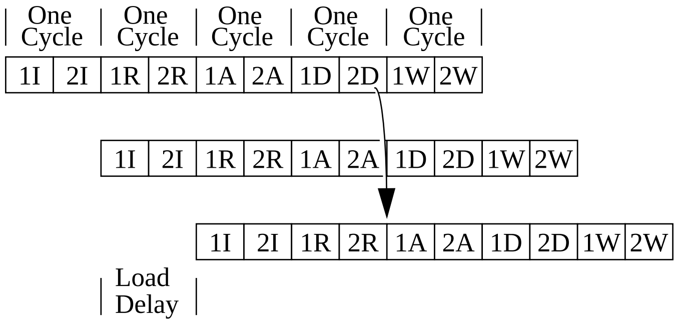


Figure 4-4 CPU Pipeline Load Delay

4.4 Interlock and Exception Handling

Smooth pipeline flow is interrupted when cache misses or exceptions occur, or when data dependencies are detected. Interruptions handled using hardware, such as cache misses, are referred to as *interlocks*, while those that are handled using software are called exceptions.

There are two types of interlocks:

- Stalls, which are resolved by halting the pipeline.
- Slips, which require one part of the pipeline to advance while another part of the pipeline is held static.

At each cycle, exception and interlock conditions are checked for all active instructions.

Because each exception or interlock condition corresponds to a particular pipeline stage, a condition can be traced back to the particular instruction in the exception/ interlock stage. For instance, a Reserved Instruction (RI) exception is raised in the execution (A) stage.

Table 4-1 Relationship of Pipeline Stage to Interlock Condition

State	Pipeline Stage				
	I	R	A	D	W
Stall	ITM	ICM		DCM	
				CPE	
Slip		LDI			
		MDSt			
		FCBusy			
Exceptions	ITLB	IBE	RI	DBE	
		IPErr	CUn	NMI	
			BP	Reset	
			SC	DPErr	
			DTLB	OVF	
			TLBMod	FPE	
			Intr		

Table 4-2 Pipeline Exceptions

Exception	Description
ITLB	Instruction Translation or Address Exception
Intr	External Interrupt
IBE	IBus Error
RI	Reserved Instruction
BP	Breakpoint
SC	System Call
CUn	Coprocessor Unusable
IPErr	Instruction Parity Error
OVF	Integer Overflow
FPE	FP Interrupt
DTLB	Data Translation or Address Exception
TLBMod	TLB Modified
DBE	Data Bus Error
DPErr	Data Parity Error
NMI	Non-maskable Interrupt
Reset	Reset

Table 4-3 Pipeline Interlocks

Interlock	Description
ITM	Instruction TLB Miss
ICM	Instruction Cache Miss
CPE	Coprocessor Possible Exception
DCM	Data Cache Miss
LDI	Load Interlock
MDSt	Multiply/Divide Start
FCBsy	FP Busy

4.4.1 Exception Conditions

When an exception condition occurs, the relevant instruction and all those that follow it in the pipeline are cancelled. Accordingly, any stall conditions and any later exception conditions that may have referenced this instruction are inhibited; there is no benefit in servicing stalls for a cancelled instruction. When this instruction reaches the W stage, three events occur;

- The exception flag causes the instruction to write various CP0 registers with the exception state,
- The current PC is changed to the appropriate exception vector address,
- The exception bits of earlier pipeline stages are cleared.

This implementation allows all instructions which occurred before the exception to complete, and all instructions which occurred after the instruction to be aborted. Hence the value of the EPC is such that execution can be restarted. In addition, all exceptions are guaranteed to be taken in order. Figure 4-5 illustrates the exception detection mechanism for a Reserved Instruction (RI) exception.

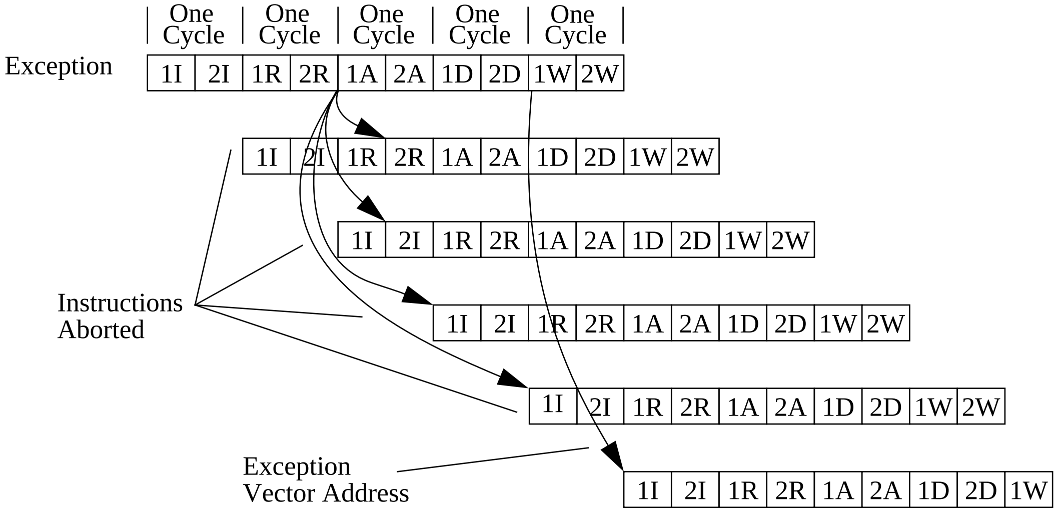
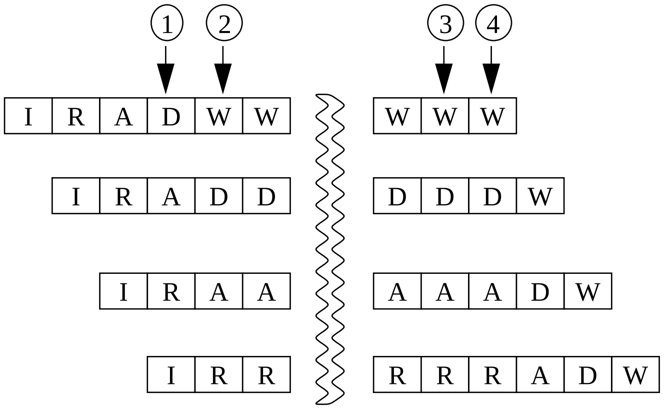

One Cycle
One Cycle
One Cycle
One Cycle
One Cycle
Exception
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
Instructions Aborted
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
Exception Vector Address
1I 2I 1R 2R 1A 2A 1D 2D 1W

Figure 4-5 Exception Detection Mechanism

4.4.2 Stall Conditions

A stall condition is used to suspend the pipline for conditions detected after the R pipeline stage. When a stall occurs, the processor resolves the condition and then restarts the pipeline. Once the interlock is removed, the restart sequence begins two cycles before the pipeline resumes execution. The restart sequence reverses the pipeline overrun by inserting the correct information into the pipeline. Figure 4-6 shows a data cache miss stall.



1 - Detect cache miss
2 - Start moving dirty cache line data to write buffer
3 - Fetch first doubleword into cache and restart pipeline
4 - Begin loading remainder of cache line into cache when Dcache is idle

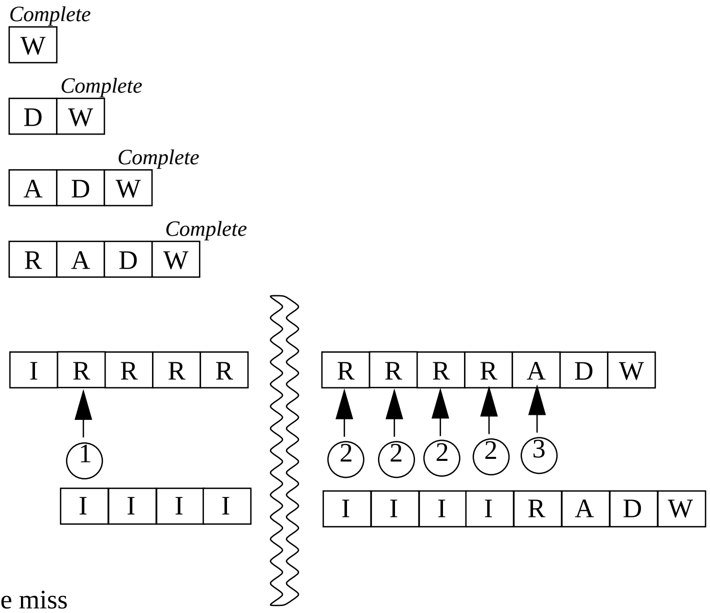
Figure 4-6 Servicing a Data Cache Miss

The data cache miss is detected in the D stage of the pipeline. If the cache line to be replaced is dirty, the W bit is set and data is moved to the internal write buffer in the next cycle. The squiggly line in Figure 4-6 indicates the memory access. Once the memory is accessed and the first doubleword of data is returned, the pipeline is restarted. The remainder of the cache line is returned in subsequent cycles. The dirty data in the write buffer is written out to memory after the cache line fill operations is completed.

4.4.3 Slip Conditions

During the 2R and 1A pipeline stages, internal logic determines whether it is possible to start the current instruction in this cycle. If all required source operands are available, as well as all hardware resources needed to complete the operation, then the instruction is issued. Otherwise, the instruction “slips”. Slipped instructions are retried on subsequent cycles until they are issued. Pipeline stages D and W advance normally during slips in an attempt to resolve the conflict. NOP’s are inserted into the bubbles which are created in the pipeline. Branch -likely instructions, ERET, nor exceptions do not cause slips.

Figure 4-7 shows how instructions can slip during an instruction cache miss.



1 - Detect cache miss
2 - Load cache line (4 doublewords) into Icache
3 - Restart pipeline

Figure 4-7 Slips During an Instruction Cache Miss

Instruction cache misses are detected in the R-stage of the pipeline. Slips are detected in the A stage. Instruction cache misses never require a writeback operation as writes are not allowed to the instruction cache. Unlike the data cache, early restart, where the pipeline is restarted after only a portion of the cache line fill has occurred, is not implemented for the instruction cache. The requested cache line is loaded into the instruction cache in its entirety before the pipeline is restarted.

4.5 Write Buffer

The V_R5000 processor contains a write buffer which improves the performance of write operations to external memory. All write cycles use the write buffer. The write buffer holds up to four 64-bit address and data pairs.

On a cache miss requiring a write-back, the entire buffer is used for the write-back data and allows the processor to proceed in parallel with the memory update. For uncached and write-through stores, the write buffer decouples the CPU from the write to memory. If the write buffer is full, additional stores are stalled until there is room for them in the write buffer.

Chapter 5 Superscalar Issue Mechanism

The V_R5000 processor incorporates a simple dual-issue mechanism which allows two instructions to be dispatched per cycle under certain conditions. A FPU ALU operation can be dispatched along with any other type of instruction, as long as the other instruction is not another FP ALU operation.

Figure 5-1 shows a simplfied diagram of the dual issue mechanism.

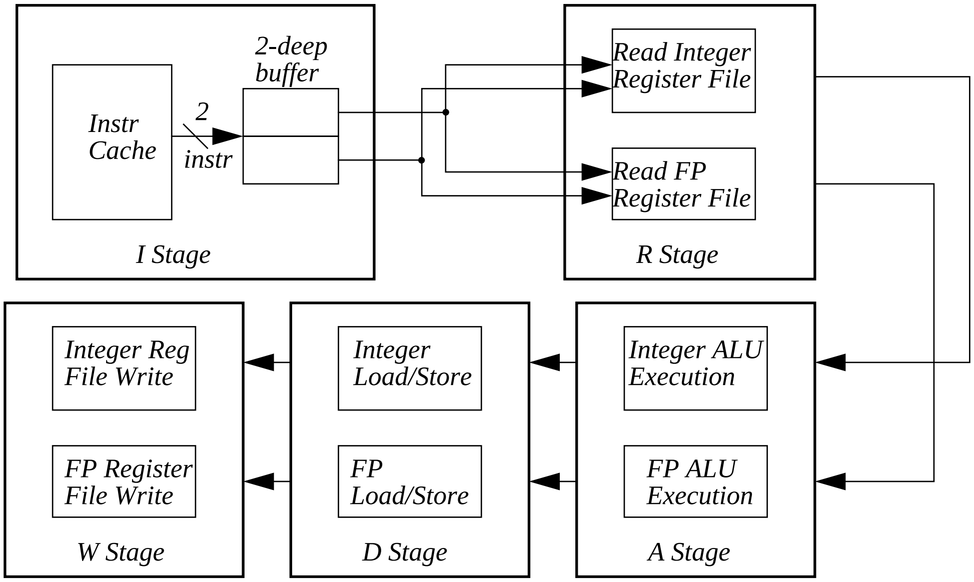


Figure 5-1 Dual Issue Mechanism

I - Stage

Two instructions are fetched from the instruction cache and placed in a 2-deep instruction buffer. Issue logic determines the type of instruction and which pipeline the instruction is routed to. Also, the instruction cache tag is checked against the page frame number (PFN) obtained from the ITLB.

R - Stage

Any required operands are fetched from the appropriate register file, and the decision is made to either proceed or slip the instruction based on any interlock conditions. For branch instruction, the branch address is calculated.

A - Stage

The appropriate ALU begins the arithmetic, logical, or shift operation. The data virtual address is calculated for any load or store instructions. The appropriate ALU determines whether the branch condition is true. The data cache access is started.

D - Stage

The data cache access is completed. Data is shifted down and extended. Data address translation in the DTLB completes. The virtual to physical address translation in the JTLB is performed. The data cache tag is checked against the PFN from the DTLB or JTLB for any data cache access.

W - Stage

The processor resolves all exceptions. For register-to-register and load instructions, the result is written back to the appropriate register file.

Chapter 6 Memory Management Unit

The V_R5000 processor provides a full-featured memory management unit (MMU) which uses an on-chip translation lookaside buffer (TLB) to translate virtual addresses into physical addresses.

This chapter describes the processor virtual and physical address spaces, the virtual-to-physical address translation, the operation of the TLB in making these translations, and those System Control Coprocessor (CP0) registers that provide the software interface to the TLB.

6.1 Translation Lookaside Buffer (TLB)

Mapped virtual addresses are translated into physical addresses using an on-chip TLB.[†] The TLB is a fully associative memory that holds 48 entries, which provide mapping to 48 odd/even page pairs (96 pages). When address mapping is indicated, each TLB entry is checked simultaneously for a match with the virtual address that is extended with an ASID stored in the *EntryHi* register.

The address mapped to a page ranges in size from 4 KB to 16 MB, in multiples of 4—that is, 4K, 16K, 64K, 256K, 1M, 4M, 16M.

6.1.1 Hits and Misses

If there is a virtual address match, or hit, in the TLB, the physical page number is extracted from the TLB and concatenated with the offset to form the physical address (see Figure 6-1).

If no match occurs (TLB miss), an exception is taken and software refills the TLB from the page table resident in memory. Software can write over a selected TLB entry or use a hardware mechanism to write into a random entry.

6.1.2 Multiple Matches

The V_R5000 processor does not provide any detection or shutdown mechanism for multiple matches in the TLB. Unlike earlier designs, multiple matches do not physically damage the TLB. Therefore, multiple match detection is not needed. The result of this condition is undefined, and software is expected to never allow this to occur.

6.2 Processor Modes

The V_R5000 has three processor operating modes, an instruction set mode, and an addressing mode. All are described in this section.

[†] There are virtual-to-physical address translations that occur outside of the TLB. For example, addresses in *the kseg0* and *kseg1* spaces are unmapped translations. In these spaces the physical address is 0x000 0000 0 11 VA[28:0].

6.2.1 Processor Operating Modes

The three operating modes are listed in order of decreasing system privilege:

- **Kernel Mode** (Highest system privilege): can access and change any register. The innermost core of the operating system runs in kernel mode.
- **Supervisor Mode:** has fewer privileges and is used for less critical sections of the operating system.
- **User Mode** (lowest system privilege): prevents users from interfering with one another.

User mode is the processor's base operating mode. The processor is forced to Kernel mode when the processor is handling an error (ERL bit is set) or an exception (EXL bit is set).

The processor's operating mode is set by the *Status* register's *KSU* field, together with the *ERL, EXL, KX, SX, UX* and *XX* bits. Table 6-1 lists the *Status* register settings for the three operating modes, as well as error and exception level settings; the blanks in the table indicate *don't cares*.

Table 6-1 Processor Modes

XX **31**	**KX** **7**	**SX** **6**	**UX** **5**	**KSU** **2**	**ERL** **2**	**EXL** **1**	**IE** **0**	**Description**	**ISA** **III**	**ISA** **IV**	**Addressing** **Mode** **32-Bit/64-Bit**
			0	10	0	0		User mode	0	0	32
0			1	10	0	0			1	0	64
1			1	10	0	0			1	1	64
		0		01	0	0		Supervisor mode	0	1	32
		1		01	0	0			1	1	64
	0			00	0	0		Kernel mode	1	1	32
	1			00	0	0			1	1	64
	0				0	1		Exception level	1	1	32
	1				0	1			1	1	64
	0				1			Error level	1	1	32
	1				1				1	1	64
					0	0	1	Interrupts are enabled			

6.2.2 Instruction Set Mode

The processor's *instruction set mode* determines which instruction set is enabled. By default, the processor implements the MIPS IV Instruction Set Architecture (ISA). For compatibility with earlier machines, however, it can be limited to the MIPS III ISA or the MIPS I/II ISAs.

6.2.3 Addressing Modes

The processor's *addressing mode* determines whether it generates 32-bit or 64-bit memory addresses.

Refer to Table 6-1 for the following addressing mode encodings:

- In Kernel mode the *KX* bit enables 64-bit addressing; all instructions are always valid.
- In Supervsor mode, the *SX* bit enables 64-bit addressing and the MIPS III instructions.
- In User mode, the *UX* bit enables 64-bit addressing and the MIPS III instructions; the *XX* bit enables the new MIPS IV instructions.

6.3 Address Spaces

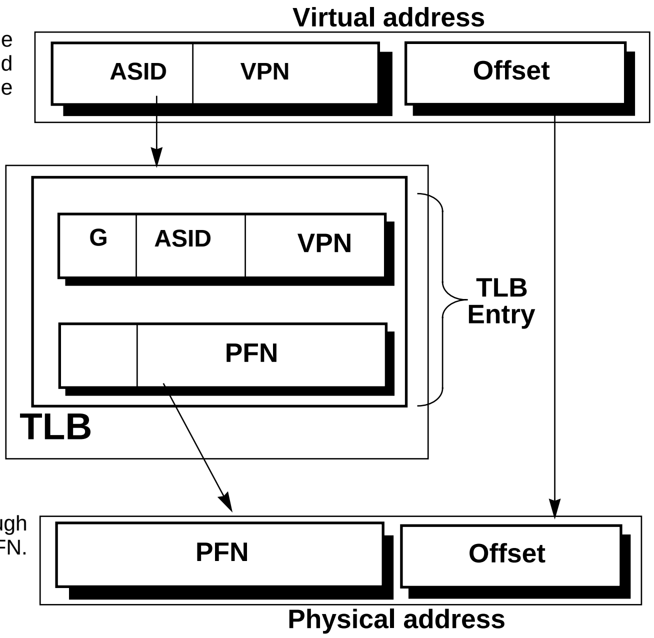
This section describes the virtual and physical address spaces and the manner in which virtual addresses are converted or "translated" into physical addresses in the TLB.

6.3.1 Virtual Address Space

The processor has three address spaces*:* kernel, supervisor, and user. Each space can be independently configured to be a 32-bit or 64-bit space by the KX, SX, and UX bits in the Status register.

- If UX=0 (extended address bit = 0), user addresses are 32 bits wide. The maximum user process size is 2 GB (2^{31}).
- If UX=1 (extended address bit = 1), user addresses are 64 bits wide. The maximum user process size is 1 TB (2^{40}).

Figure 6-1 shows the translation of a virtual address into a physical address.



1. Virtual address (VA) represented by the virtual page number (VPN) is compared with tag in TLB. The ASID portion of the VA is held in EnHI Register.

2. If there is a match, the page frame number (PFN) representing the upper bits of the physical address (PA) is output from the TLB.

3. The Offset, which does not pass through the TLB, is then concatenated to the PFN.

Figure 6-1 Overview of a Virtual-to-Physical Address Translation

As shown in Figure 6-1, the virtual address is extended with an 8-bit address space identifier (ASID), which reduces the frequency of TLB flushing when switching contexts. This 8-bit ASID is in the CP0 *EntryHi* register. The *Global* bit (*G*) is in each TLB entry.

6.3.2 Physical Address Space

Using a 36-bit address, the processor physical address space encompasses 64 GB.

6.3.3 Virtual-to-Physical Address Translation

Converting a virtual address to a physical address begins by comparing the virtual address from the processor with the virtual addresses in the TLB; there is a match when the virtual page number (VPN) of the address is the same as the VPN field of the entry, and either:

- the Global (*G*) bit of the TLB entry is set, or
- the ASID field of the virtual address is the same as the ASID field of the TLB entry.

This match is referred to as a *TLB hit*. If there is no match, a TLB Miss exception is taken by the processor and software is allowed to refill the TLB from a page table of virtual/physical addresses in memory.

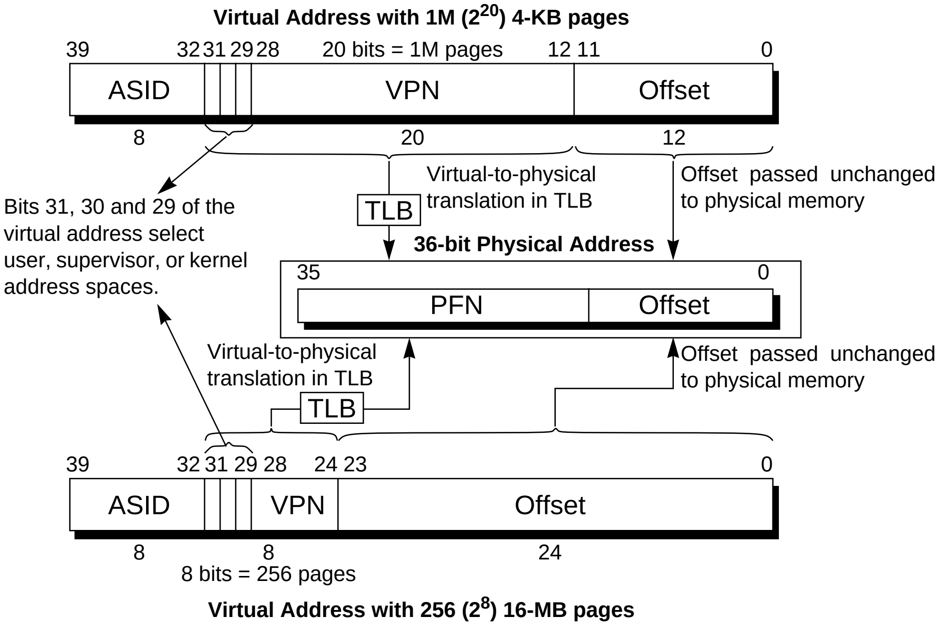
If there is a virtual address match in the TLB, the physical address is output from the TLB and concatenated with the *Offset*, which represents an address within the page frame space. The *Offset* does not pass through the TLB.

The next sections describe the 32-bit and 64-bit address translations.

6.3.4 32-bit Mode Virtual Address Translation

Figure 6-2 shows the virtual-to-physical-address translation of a 32-bit mode address.

- The top portion of Figure 6-2 shows a virtual address with a 12-bit, or 4-KB, page size, labelled *Offset*. The remaining 20 bits of the address represent the VPN, and index the 1M-entry page table.
- The bottom portion of Figure 6-2 shows a virtual address with a 24-bit, or 16-MB, page size, labelled *Offset*. The remaining 8 bits of the address represent the VPN, and index the 256-entry page table.



Virtual Address with 1M (2^20) 4-KB pages
39
32
31
29
28
20 bits = 1M pages
12
11
0
ASID
VPN
Offset
8
20
12
Virtual-to-physical translation in TLB
Offset passed unchanged to physical memory
Bits 31, 30 and 29 of the virtual address select user, supervisor, or kernel address spaces.
TLB
36-bit Physical Address
35
0
PFN
Offset
Virtual-to-physical translation in TLB
Offset passed unchanged to physical memory
TLB
39
32
31
29
28
24
23
0
ASID
VPN
Offset
8
8
24
8 bits = 256 pages
Virtual Address with 256 (2^8) 16-MB pages

Figure 6-2 32-bit Mode Virtual Address Translation

6.3.5 64-bit Mode Virtual Address Translation

Figure 6-3 shows the virtual-to-physical-address translation. This figure illustrates the two extremes in the range of possible page sizes: a 4-KB page (12 bits) and a 16-MB page (24 bits).

- The top portion of Figure 6-3 shows a virtual address with a 12-bit, or 4-KB, page size, labelled *Offset*. The remaining 28 bits of the address represent the VPN, and index the 256M-entry page table.
- The bottom portion of Figure 6-3 shows a virtual address with a 24-bit, or 16-MB, page size, labelled *Offset*. The remaining 16 bits of the address represent the VPN, and index the 64K-entry page table.

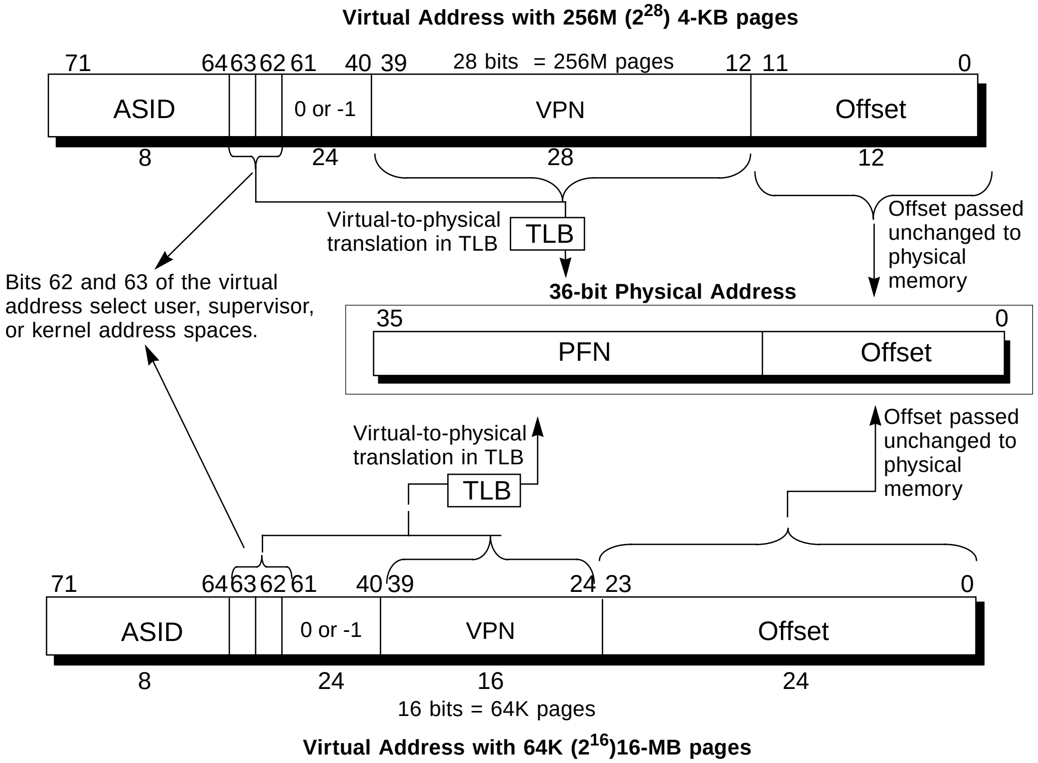

Virtual Address with 256M (2^{28}) 4-KB pages
71
64
63
62
61
40
39
28 bits = 256M pages
12
11
0
ASID
0 or -1
VPN
Offset
8
24
28
12
Virtual-to-physical translation in TLB
TLB
Offset passed unchanged to physical memory
Bits 62 and 63 of the virtual address select user, supervisor, or kernel address spaces.
36-bit Physical Address
35
0
PFN
Offset
Virtual-to-physical translation in TLB
TLB
Offset passed unchanged to physical memory
71
64
63
62
61
40
39
24
23
0
ASID
0 or -1
VPN
Offset
8
24
16
24
16 bits = 64K pages
Virtual Address with 64K (2^{16})16-MB pages

Figure 6-3 64-bit Mode Virtual Address Translation

6.3.6 Address Spaces

The processor has three address spaces.

- User address space
- Supervisor address space
- Kernel address space

Each space can be independently configured as either 32- or 64-bit.

6.3.7 User Address Space

In User address space, a single, uniform virtual address space—labelled User segment (*useg*), is available; its size is:

- 2 GB (2^{31} bytes) if UX = 0 (*useg*)
- 1 TB (2^{40} bytes) if UX = 1 (*xuseg*)

Figure 6-4 shows the range of User virtual address space.

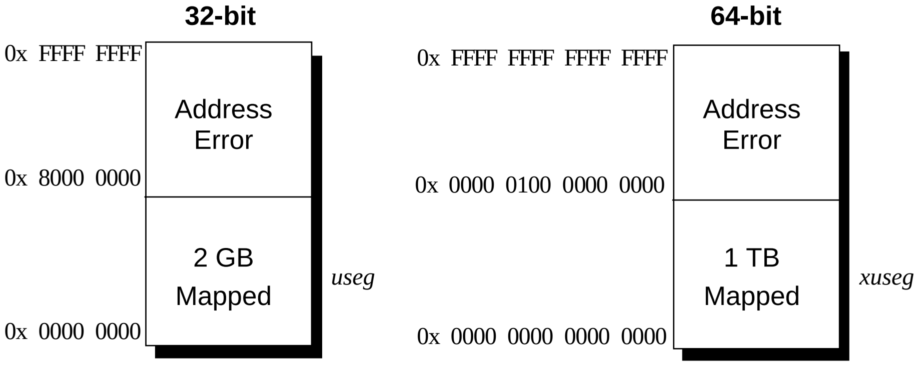


Figure 6-4 UserVirtual Address Space as Viewed from User Mode

User space can be accessed from user, supervisor, and kernel modes.

The User segment starts at address 0 and the current active user process resides in either useg (in 32-bit mode) or xuseg (in 64-bit mode). The TLB identically maps all references to useg/xuseg from all modes, and controls cache accessibility.

The processor operates in User mode when the *Status* register contains the following bit-values:

- *KSU* bits = 10_2
- *EXL* = 0
- *ERL* = 0

The *UX* bit in the *Status* register selects between 32- or 64-bit User address spaces as follows:

- when *UX* = 0, 32-bit *useg* space is selected.
- when *UX* = 1, 64-bit *xuseg* space is selected.

Table 6-2 lists the characteristics of the two user address spaces, *useg* and *xuseg*.

Table 6-2 32-bit and 64-bit User Address Space Segments

Address Bit Values	Status Register Bit Values				Segment Name	Address Range	Segment Size
	KSU	EXL	ERL	UX			
32-bit A(31) = 0	any		0	0	*useg*	0x0000 0000 through 0x7FFF FFFF	2 GB (2^{31} bytes)
64-bit A(63:40) = 0			0	1	*xuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 TB (2^{40} bytes)

(1) 32-bit User Space (useg)

In 32-bit User space, when *UX* = 0 in the *Status* register, all valid addresses have their most-significant bit cleared to 0; any attempt to reference an address with the most-significant bit set while in User mode causes an Address Error exception.

The system maps all references to *useg* through the TLB, and bit settings within the TLB entry for the page determine the cacheability of a reference. TLB misses on addresses in 32-bit User space (*useg*) use the TLB refill vector.

(2) 64-bit User Space (*xuseg*)

In 64-bit User space, when *UX* =1 in the *Status* register, addressing is extended to 64-bits. When UX=1, the processor provides a single, uniform address space of 2^{40} bytes, labelled *xuseg*.

All valid User mode virtual addresses have bits 63:40 equal to 0; an attempt to reference an address with bits 63:40 not equal to 0 causes an Address Error exception. TLB misses on addresses in 64-bit User (*xuseg*) space use the XTLB refill vector.

6.3.8 Supervisor Space

Supervisor address space is designed for layered operating systems in which a true kernel runs in Kernel mode, and the rest of the operating system runs in Supervisor mode. The Supervisor address space provides code and data addresses for supervisor mode.

Supervisor space can be accessed from supervisor mode and kernel mode.

The processor operates in Supervisor mode when the *Status* register contains the following bit-values:

- *KSU* = 01_2
- *EXL* = 0
- *ERL* = 0

The *SX* bit in the *Status* register select between 32- or 64-bit Supervisor space addressing:

- when *SX* = 0, 32-bit supervisor space is selected and TLB misses on supervisor space addresses are handled by the 32-bit TLB refill exception handler
- when *SX* = 1, 64-bit supervisor space is selected and TLB misses on supervisor space addresses are handled by the 64-bit XTLB refill exception handler. Figure 6-5 shows Supervisor address mapping. Table 6-3 lists the characteristics of the supervisor space segments; descriptions of the address spaces follow.

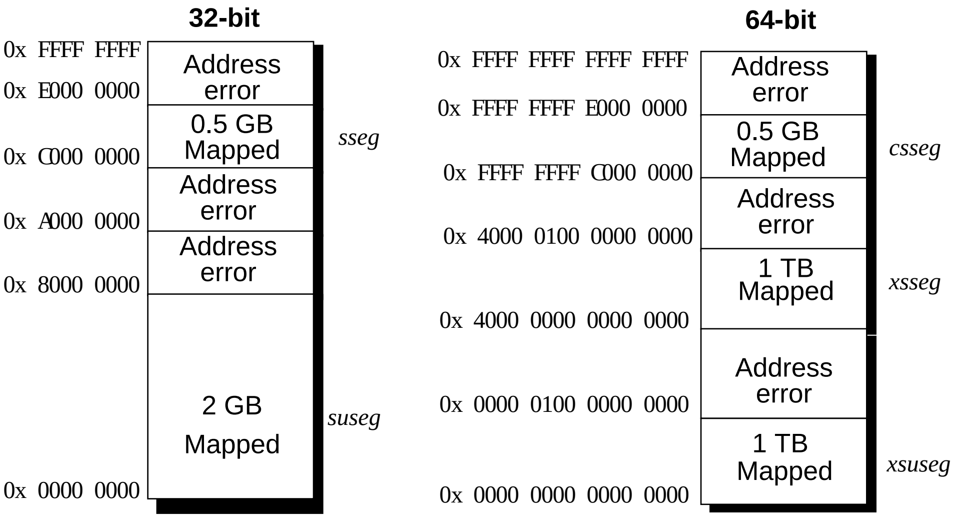


Figure 6-5 User and Supervisor Address Spaces as Viewed from Supervisor Mode

Table 6-3 Supervisor Mode Addressing

A(63:62)	SX	UX	Segment Name	Address Range	Segment Size
00_2	X	0	*suseg*	0x0000 0000 0000 0000 through 0x0000 0000 7FFF FFFF	2 GB (2^{31} bytes)
00_2	X	1	*xsuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 TB (2^{40} bytes)
01_2	1	X	*xsseg*	0x4000 0000 0000 0000 through 0x4000 00FF FFFF FFFF	1 TB (2^{40} bytes)
11_2	X	X	*sseg* *or* *csseg*	0xFFFF FFFF C000 0000 through 0xFFFF FFFF DFFF FFFF	512 MB (2^{29} bytes)

(1) 32-bit Supervisor, User Space (*suseg*)

In Supervisor space, when *SX* = 0 in the *Status* register and the most-significant bit of the 32-bit virtual address is set to 0, the *suseg* virtual address space is selected; it covers the full 2^{31} bytes (2 GB) of the current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

This mapped space starts at virtual address 0x0000 0000 and runs through 0x7FFF FFFF.

(2) 32-bit Supervisor, Supervisor Space (*sseg*)

In Supervisor space, when *SX* = 0 in the *Status* register and the three most-significant bits of the 32-bit virtual address are 110_2, the *sseg* virtual address space is selected; it covers 2^{29}-bytes (512 MB) of the current supervisor address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

This mapped space begins at virtual address 0xC000 0000 and runs through 0xDFFF FFFF.

(3) 64-bit Supervisor, User Space (*xsuseg*)

In Supervisor space, when *SX* = 1 in the *Status* register and bits 63:62 of the virtual address are set to 00_2, the *xsuseg* virtual address space is selected; it covers the full 2^{40} bytes (1 TB) of the current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

This mapped space starts at virtual address 0x0000 0000 0000 0000 and runs through 0x0000 00FF FFFF FFFF.

(4) 64-bit Supervisor, Current Supervisor Space (*xsseg*)

In Supervisor space, when *SX* = 1 in the *Status* register and bits 63:62 of the virtual address are set to 01_2, the *xsseg* current supervisor virtual address space is selected. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

This mapped space begins at virtual address 0x4000 0000 0000 0000 and runs through 0x4000 00FF FFFF FFFF.

(5) 64-bit Supervisor, Separate Supervisor Space (*csseg*)

In Supervisor space, when *SX* = 1 in the *Status* register and bits 63:62 of the virtual address are set to 11_2, the *csseg* separate supervisor virtual address space is selected. Addressing of the *csseg* is compatible with addressing *sseg* in 32-bit mode. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

This mapped space begins at virtual address 0xFFFF FFFF C000 0000 and runs through 0xFFFF FFFF DFFF FFFF.

6.3.9 Kernel Space

The processor operates in Kernel mode when the *Status* register contains one of the following values:

- *KSU* = 00_2
- *EXL* = 1
- *ERL* = 1

The *KX* bit in the *Status* register selects between 32- or 64-bit Kernel space addressing:

- when *KX* = 0, 32-bit kernel space is selected.
- when *KX* = 1, 64-bit kernel space is selected.

The processor enters Kernel mode whenever an exception is detected and it remains there until an Exception Return (ERET) instruction is executed or EXL is cleared. The ERET instruction restores the processor to the address space existing prior to the exception.

Kernel virtual address space is divided into regions differentiated by the high-order bits of the virtual address, as shown in Figure 6-6. Table 6-4 lists the characteristics of the kernel mode segments.

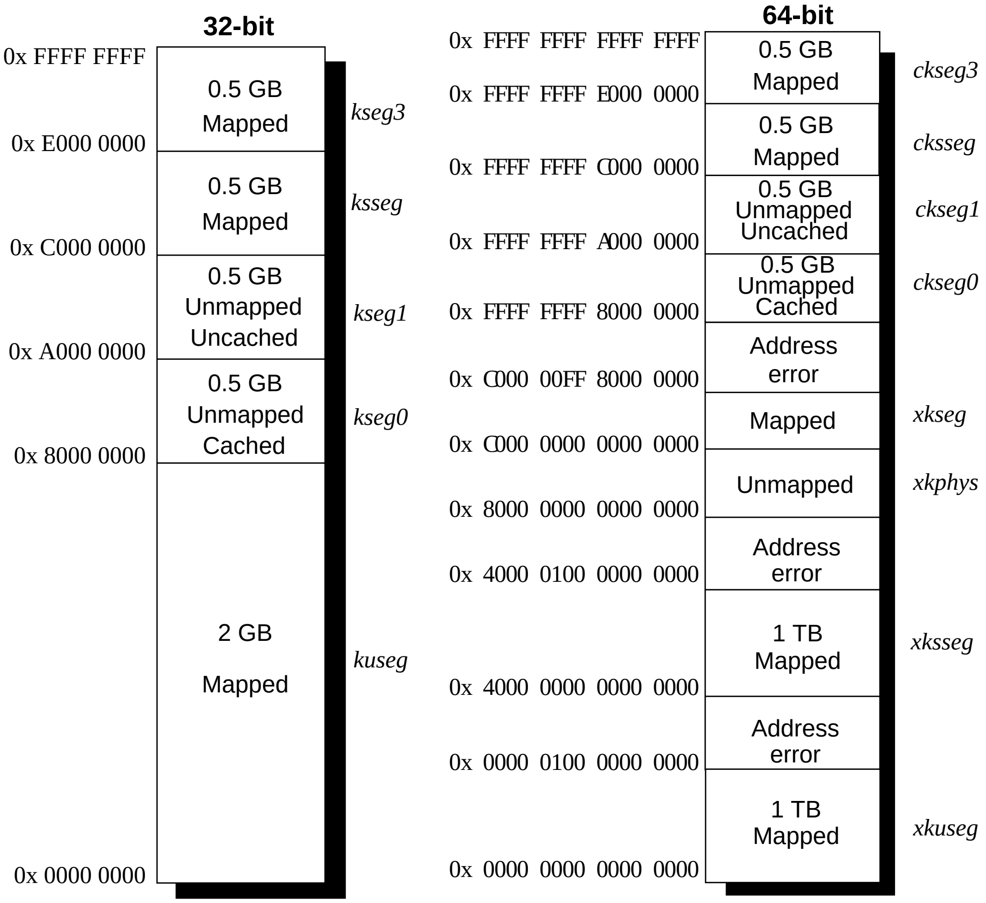


Figure 6-6 User, Supervisor, and Kernel Address Spaces as Viewed from Kernel Mode

Table 6-4 Kernel Mode Addressing

A(63:62)				Segment Name	Address Range	Segment Size
	KX	SX	UX			
00_2	X	X	0	*kuseg*	0x0000 0000 0000 0000 through 0x0000 0000 7FFF FFFF	2 GB (2^{31} bytes)
00_2	X	X	1	*xkuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 TB (2^{40} bytes)
01_2	X	1	X	*xksseg*	0x4000 0000 0000 0000 through 0x4000 00FF FFFF FFFF	1 TB (2^{40} bytes)
10_2	1	X	X	*xkphys*	0x8000 0000 0000 0000 through 0x8000 000F FFFF FFFF etc.	8× 64 GB (2^{36} bytes)
11_2	1	X	X	*xkseg*	0xC000 0000 0000 0000 through 0xC000 00FF 7FFF FFFF	(2^{40}–2^{31}) bytes
11_2	X	X	X	*kseg0*	0xFFFF FFFF 8000 0000 through 0xFFFF FFFF 9FFF FFFF	512 MB (2^{29} bytes)
11_2	X	X	X	*kseg1*	0xFFFF FFFF A000 0000 through 0xFFFF FFFF BFFF FFFF	512 MB (2^{29} bytes)
11_2	X	X	X	*ksseg*	0xFFFF FFFF C000 0000 through 0xFFFF FFFF DFFF FFFF	512 MB (2^{29} bytes)
11_2	X	X	X	*kseg3*	0xFFFF FFFF E000 0000 through 0xFFFF FFFF FFFF FFFF	512 MB (2^{29} bytes)

(1) 32-bit Kernel, User Space (*kuseg*)

In Kernel space, when *KX* = 0 in the *Status* register, and the most-significant bit of the virtual address, A31, is cleared, the 32-bit *kuseg* virtual address space is selected; it covers the full 2^{31} bytes (2 GB) of the current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

(2) 32-bit Kernel, Kernel Space 0 (*kseg0*)

In Kernel space, when *KX* = 0 in the *Status* register and the most-significant three bits of the virtual address are 100_2, 32-bit *kseg0* virtual address space is selected; it is the 2^{29}-byte (512-MB) kernel physical space. References to *kseg0* are not mapped through the TLB; the physical address selected is defined by subtracting 0x8000 0000 from the virtual address. The *K0* field of the *Config* register, described in this chapter, controls cacheability and coherency.

(3) 32-bit Kernel, Kernel Space 1 (*kseg1*)

In Kernel mode, when *KX* = 0 in the *Status* register and the most-significant three bits of the 32-bit virtual address are 101_2, 32-bit *kseg1* virtual address space is selected; it is the 2^{29}-byte (512-MB) kernel physical space.

References to *kseg1* are not mapped through the TLB; the physical address selected is defined by subtracting 0xA000 0000 from the virtual address.

Caches are disabled for accesses to these addresses, and physical memory (or memory-mapped I/O device registers) are accessed directly.

(4) 32-bit Kernel, Supervisor Space (*ksseg*)

In Kernel space, when *KX* = 0 in the *Status* register and the most-significant three bits of the 32-bit virtual address are 110_2, the *ksseg* virtual address space is selected; it is the current 2^{29}-byte (512-MB) supervisor virtual space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

(5) 32-bit Kernel, Kernel Space 3 (*kseg3*)

In Kernel space, when *KX* = 0 in the *Status* register and the most-significant three bits of the 32-bit virtual address are 111_2, the *kseg3* virtual address space is selected; it is the current 2^{29}-byte (512-MB) kernel virtual space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

(6) 64-bit Kernel, User Space (*xkuseg*)

In Kernel space, when *KX* = 1 in the *Status* register and bits 63:62 of the 64-bit virtual address are 00_2, the *xkuseg* virtual address space is selected; it covers the current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

When ERL = 1 in the *Status* register, the user address region becomes a 2^{31}-byte unmapped (that is, mapped directly to physical addresses) uncached address space.

(7) 64-bit Kernel, Current Supervisor Space (*xksseg*)

In Kernel space, when *KX* = 1 in the *Status* register and bits 63:62 of the 64-bit virtual address are 01_2, the *xksseg* virtual address space is selected; it is the current supervisor virtual space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

(8) 64-bit Kernel, Physical Spaces (*xkphys*)

In Kernel space, when *KX* = 1 in the *Status* register and bits 63:62 of the 64-bit virtual address are 10_2, the *xkphys* virtual address space is selected; it is a set of eight 2^{36}-byte kernel physical spaces. Accesses with address bits 58:36 not equal to 0 cause an address error.

References to this space are not mapped; the physical address selected is taken from bits 35:0 of the virtual address. Bits 61:59 of the virtual address specify the cacheability and coherency attributes, as shown in Table 6-5.

Table 6-5 Cacheability and Coherency Attributes

Value (61:59)	Cacheability and Coherency Attributes	Starting Address
0	Cacheable, noncoherent, write-through, no write allocate	0x8000 0000 0000 0000
1	Cacheable, noncoherent, write-through, write allocate	0x8800 0000 0000 0000
2	Uncached	0x9000 0000 0000 0000
3	Cacheable, noncoherent	0x9800 0000 0000 0000
4-7	Reserved	0xA000 0000 0000 0000

(9) 64-bit Kernel, Kernel Space (*xkseg*)

In Kernel space, when *KX* = 1 in the *Status* register and bits 63:62 of the 64-bit virtual address are 11_2, the address space selected is one of the following:

- kernel virtual space, *xkseg*, the current kernel virtual space*;* the virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address
- one of the four 32-bit kernel compatibility spaces, as described in the next section.

(10) 64-bit Kernel, Compatibility Spaces

In Kernel space, when *KX* = 1 in the *Status* register, bits 63:62 of the 64-bit virtual address are 11_2, and bits 61:31 of the virtual address equal –1. The lower two bytes of address, as shown in Figure 6-6, select one of the following 512-MB compatibility spaces.

- *ckseg0*. This 64-bit virtual address space is an unmapped region, compatible with the 32-bit address model *kseg0*. The *K0* field of the *Config* register controls cacheability and coherency.
- *ckseg1*. This 64-bit virtual address space is an unmapped and uncached region, compatible with the 32-bit address model *kseg1*.
- *cksseg*. This 64-bit virtual address space is the current supervisor virtual space, compatible with the 32-bit address model *kss*eg.
- *ckseg3*. This 64-bit virtual address space is kernel virtual space, compatible with the 32-bit address model *kseg3*.

6.4 System Control Coprocessor

The System Control Coprocessor (CP0) is implemented as an integral part of the CPU, and supports memory management, address translation, exception handling, and other privileged operations. CP0 contains the registers shown in Figure 6-7 plus a 48-entry TLB. The sections that follow describe how the processor uses the memory management-related registers.

Each CP0 register has a unique number that identifies it; this number is referred to as the *register number*. For instance, the *Page Mask* register is register number 5.

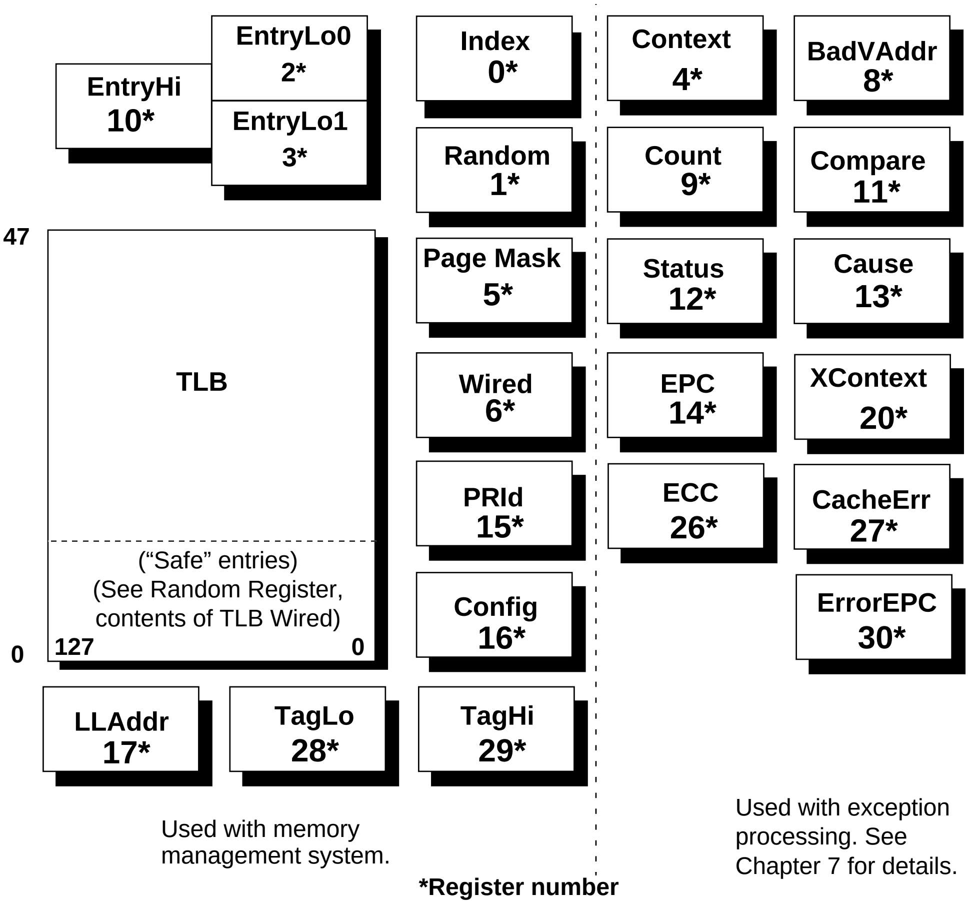
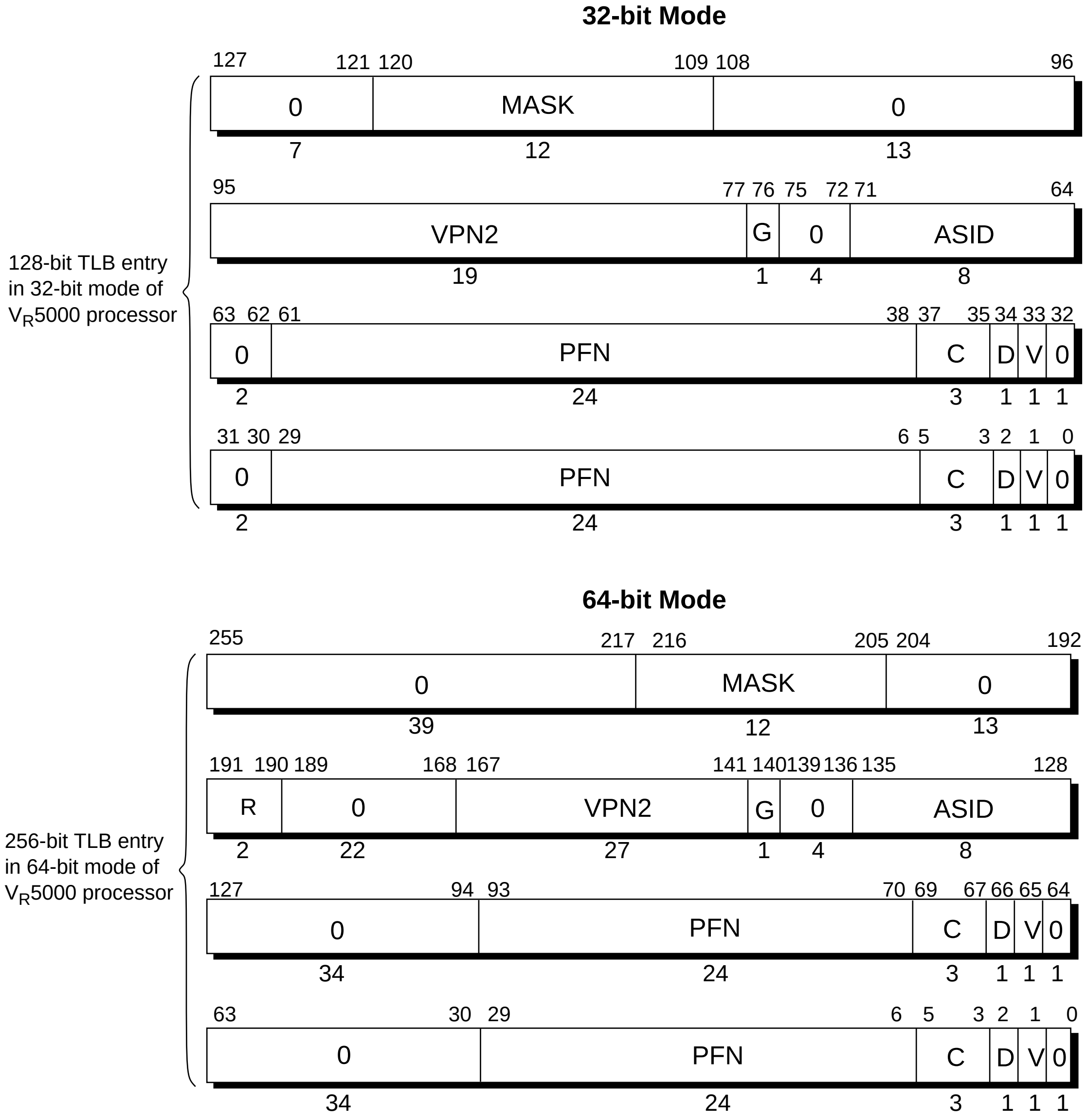


Figure 6-7 CP0 Registers and the TLB

6.4.1 Format of a TLB Entry

Figure 6-8 shows the TLB entry formats for both 32- and 64-bit modes. Each field of an entry has a corresponding field in the *EntryHi*, *EntryLo0*, *EntryLo1*, or *PageMask* registers.

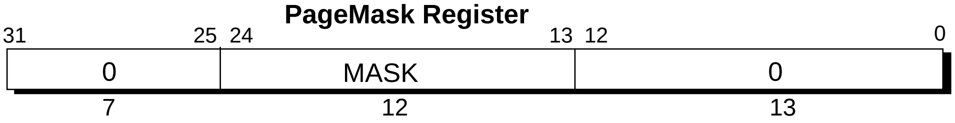

32-bit Mode

64-bit Mode

Figure 6-8 Format of a TLB Entry

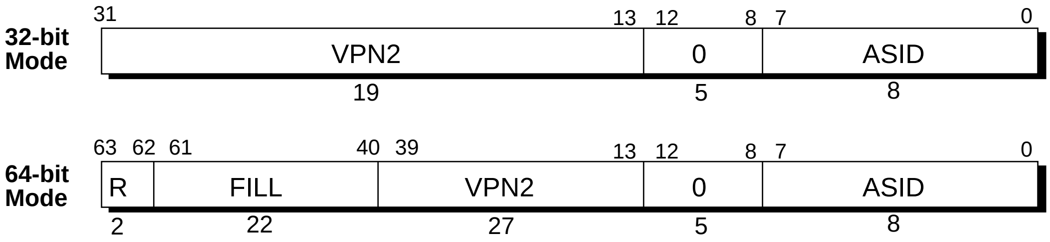
The format of the *EntryHi*, *EntryLo0*, *EntryLo1*, and *PageMask* registers are nearly the same as the TLB entry. The one exception is the *Global* field (*G* bit), which is used in the TLB, but is reserved in the *EntryHi* register. Figure 6-9 and Figure 6-10 describe the TLB entry fields shown in Figure 6-8.

PageMask Register



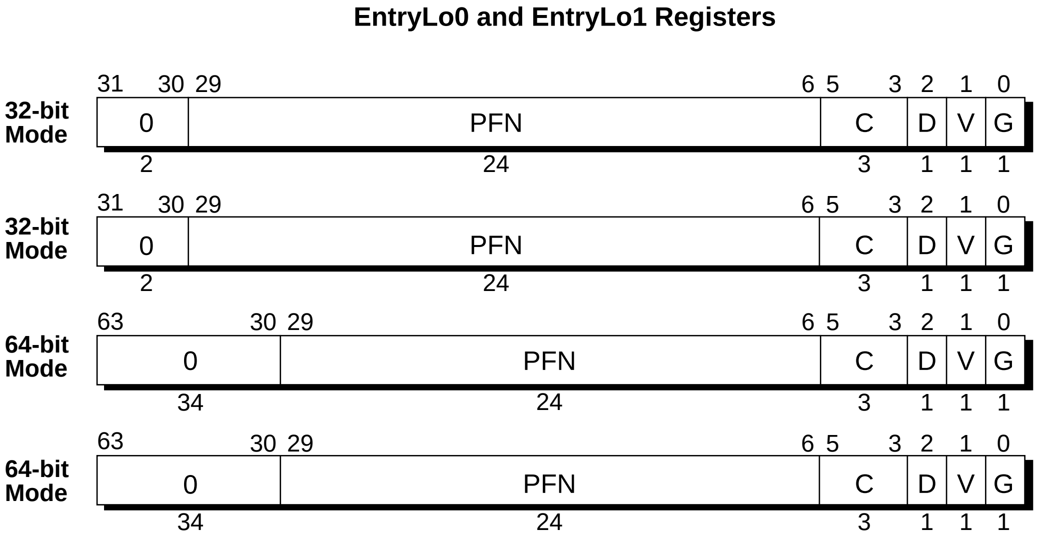
Mask..... Page comparison mask.
0........... Reserved. Must be written as zeroes, and returns zeroes when read.

EntryHi Register



VPN2 ... Virtual page number divided by two (maps to two pages).
ASID .... Address space ID field. An 8-bit field that lets multiple processes share the TLB; each process has a distinct mapping of otherwise identical virtual page numbers.
R .... Region. (00 → user, 01 → supervisor, 11 → kernel) used to match $\mathrm{vAddr}_{63...62}$
Fill .... Reserved. 0 on read; ignored on write.
0........... Reserved. Must be written as zeroes, and returns zeroes when read.

Figure 6-9 Fields of the PageMask and EntryHi Registers



PFN...... Page frame number; the upper bits of the physical address.
C .... Specifies the TLB page coherency attribute; see Table 6-6.
D .... Dirty. If this bit is set, the page is marked as dirty and, therefore, writable. This bit is actually a write-protect bit that software can use to prevent alteration of data.
V .... Valid. If this bit is set, it indicates that the TLB entry is valid; otherwise, a TLBL or TLBS miss occurs.
G .... Global. If this bit is set in both Lo0 and Lo1, then the processor ignores the ASID during TLB lookup.
0........... Reserved. Must be written as zeroes, and returns zeroes when read.

Figure 6-10 Fields of the EntryLo0 and EntryLo1 Registers

The TLB page coherency attribute (*C*) bits specify whether references to the page should be cached; if cached, the algorithm selects between several coherency attributes. Table 6-6 shows the coherency attributes selected by the *C* bits.

Table 6-6   TLB Page Coherency (C) Bit Values

***C*(5:3) Value**	**Page Coherency Attribute**
0	Cacheable, noncoherent, write-through, no write allocate
1	Cacheable, noncoherent, write-through, write allocate
2	Uncached
3	Cacheable noncoherent (noncoherent)
4	Reserved
5	Reserved
6	Reserved
7	Reserved

6.5 CP0 Registers

The following sections describe the CP0 registers that are assigned specifically as a software interface with memory management (each register is followed by its register number in parentheses).

- *Index* register (CP0 register number 0)
- *Random* register (1)
- *EntryLo0* (2) and *EntryLo1* (3) registers
- *PageMask* register (5)
- *Wired* register (6)
- *EntryHi* register (10)
- *PRId* register (15)
- *Config* register (16)
- *LLAddr* register (17)
- *TagLo* (28) and *TagHi* (29) registers

6.5.1 Index Register (0)

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

The *Index* register also specifies the TLB entry affected by TLB Read (TLBR) or TLB Write Index (TLBWI) instructions.

Figure 6-11 shows the format of the *Index* register; Table 6-7 describes the *Index* register fields.

Index Register

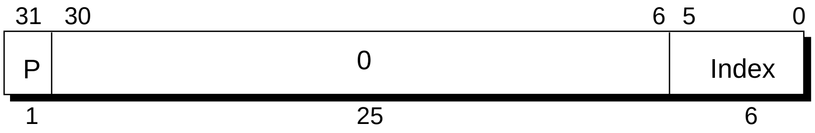


Figure 6-11 Index Register

Table 6-7 Index Register Field Descriptions

Field	Description
P	Probe failure. Set to 1 when the previous TLBProbe (TLBP) instruction was unsuccessful.
Index	Index to the TLB entry affected by the TLBRead and TLBWrite instructions
0	Reserved. Must be written as zeroes, and returns zeroes when read.

6.5.2 Random Register (1)

The *Random* register is a read-only register of which six bits index an entry in the TLB. This register decrements as each instruction executes, and its values range between an upper and a lower bound, as follows:

- A lower bound is set by the number of TLB entries reserved for exclusive use by the operating system (the contents of the *Wired* register).
- An upper bound is set by the total number of TLB entries (47 maximum).

The *Random* register specifies the entry in the TLB that is affected by the TLB Write Random instruction. The register does not need to be read for this purpose; however, the register is readable to verify proper operation of the processor.

To simplify testing, the *Random* register is set to the value of the upper bound upon system reset. This register is also set to the upper bound when the *Wired* register is written.

Figure 6-12 shows the format of the *Random* register. Table 6-8 describes the *Random* register fields.

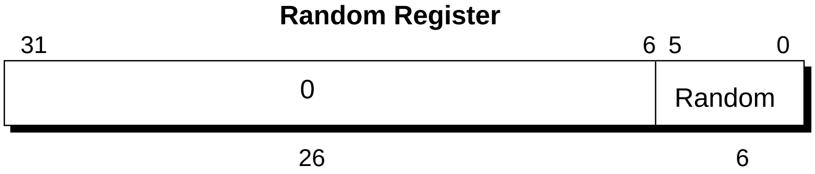


Figure 6-12   Random Register

Table 6-8   Random Register Field Descriptions

Field	Description
Random	TLB Random index
0	Reserved. Must be written as zeroes, and returns zeroes when read.

6.5.3 EntryLo0 (2), and EntryLo1 (3) Registers

The *EntryLo* register consists of two registers that have identical formats:

- *EntryLo0* is used for even virtual pages.
- *EntryLo1* is used for odd virtual pages.

The *EntryLo0* and *EntryLo1* registers are read/write registers. They hold the physical page frame number (PFN) of the TLB entry for even and odd pages, respectively, when performing TLB read and write operations. Figure 6-10 shows the format of these registers.

6.5.4 PageMask Register (5)

The *PageMask* register is a read/write register used for reading from or writing to the TLB; it holds a comparison mask that sets the variable page size for each TLB entry.

TLB read and write operations use this register as either a source or a destination; when virtual addresses are presented for translation into physical address, the corresponding bits in the TLB identify which virtual address bits among bits 24:13 are used in the comparison. When the *Mask* field is not one of the values shown in Table 6-9, the operation of the TLB is undefined.

Table 6-9 Mask Field Values for Page Sizes

Page Size	**Bit**											
	24	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**
4 KB	0	0	0	0	0	0	0	0	0	0	0	0
16 KB	0	0	0	0	0	0	0	0	0	0	1	1
64 KB	0	0	0	0	0	0	0	0	1	1	1	1
256 KB	0	0	0	0	0	0	1	1	1	1	1	1
1 MB	0	0	0	0	1	1	1	1	1	1	1	1
4 MB	0	0	1	1	1	1	1	1	1	1	1	1
16 MB	1	1	1	1	1	1	1	1	1	1	1	1

6.5.5 Wired Register (6)

The *Wired* register is a read/write register that specifies the boundary between the *wired* and *random* entries of the TLB as shown in Figure 6-13. Wired entries are fixed, nonreplaceable entries, which cannot be overwritten by a TLB write operation. Random entries can be overwritten.

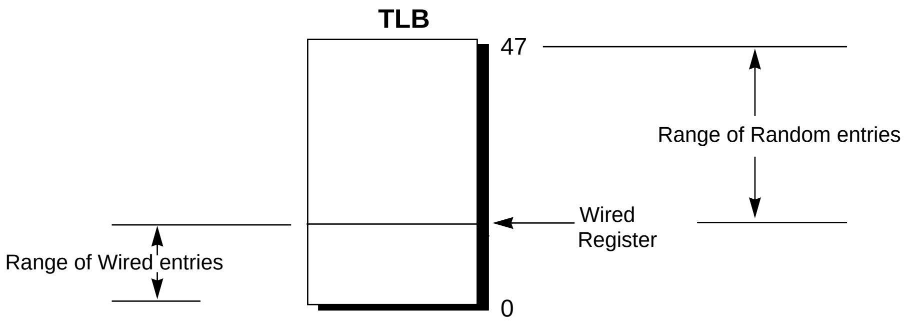


Figure 6-13 Wired Register Boundary

The *Wired* register is set to 0 upon system reset. Writing this register also sets the *Random* register to the value of its upper bound (see *Random* register, above). Figure 6-14 shows the format of the *Wired* register; Table 6-10 describes the register fields.

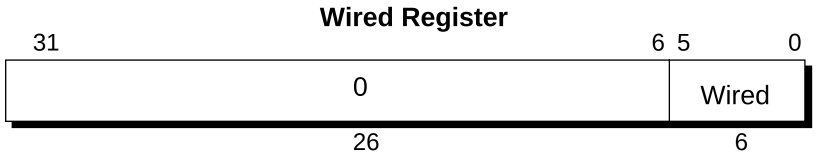


Figure 6-14 Wired Register

Table 6-10 Wired Register Field Descriptions

Field	Description
Wired	TLB Wired boundary
0	Reserved. Must be written as zeroes, and returns zeroes when read.

6.5.6 EntryHi Register (10)

The *EntryHi* register holds the high-order bits of a TLB entry for TLB read and write operations.

The *EntryHi* register is accessed by the TLB Probe, TLB Write Random, TLB Write Indexed, and TLB Read Indexed instructions.

When either a TLB refill, TLB invalid, or TLB modified exception occurs, the *EntryHi* register is loaded with the virtual page number (VPN2) and the ASID of the virtual address that did not have a matching TLB entry.

6.5.7 Processor Revision Identifier (PRId) Register (15)

The 32-bit, read-only *Processor Revision Identifier* (*PRId*) register contains information identifying the implementation and revision level of the CPU and CP0. Figure 6-15 shows the format of the *PRId* register; Table 6-11 describes the *PRId* register fields.

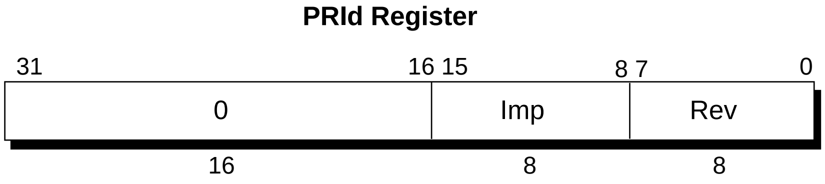


Figure 6-15 Processor Revision Identifier Register Format

Table 6-11 PRId Register Fields

Field	Description
Imp	Implementation number
Rev	Revision number
0	Reserved. Must be written as zeroes, and returns zeroes when read.

The low-order byte (bits 7:0) of the *PRId* register is interpreted as a revision number, and the high-order byte (bits 15:8) is interpreted as an implementation number. The implementation number of the V_R5000 processor is 0x23. The content of the high-order halfword (bits 31:16) of the register are reserved.

The revision number is stored as a value in 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.

The revision number can distinguish some chip revisions, however there is no guarantee that changes to the chip will necessarily be reflected in the *PRId* register, or that changes to the revision number necessarily reflect real chip 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 chip.

6.5.8 Config Register (16)

The *Config* register specifies various configuration options which can be selected.

Some configuration options, as defined by *Config* bits 31:13,11:3 are set by the hardware during reset and are included in the *Config* register as read-only status bits for the software to access. Other configuration options are read/write (as indicated by *Config* register bits 12 and 3:0) and controlled by software; on reset these fields are undefined.

Certain configurations have restrictions. The *Config* register should be initialized by software before caches are used. Caches should be written back to memory before line sizes are changed, and caches should be reinitialized after any change is made.

Figure 6-16 shows the format of the *Config* register; Table 6-12 describes the *Config* register fields.

Config Register

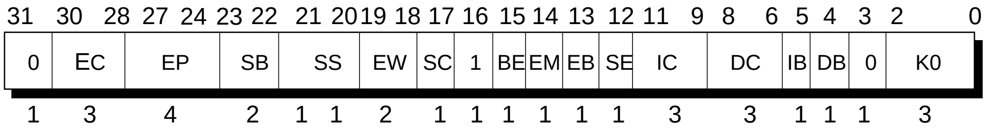


Figure 6-16   Config Register Format

Table 6-12   Config Register Fields

Field	Description
EC	System clock ratio: 0 → processor clock frequency divided by 2 1 → processor clock frequency divided by 3 2 → processor clock frequency divided by 4 3 → processor clock frequency divided by 5 4 → processor clock frequency divided by 6 5 → processor clock frequency divided by 7 6 → processor clock frequency divided by 8 7 → Reserved
EP	Transmit data pattern (pattern for write-back data): 0 → D Doubleword every cycle 1 → DDxDDx 2 Doublewords every 3 cycles 2 → DDxxDDxx 2 Doublewords every 4 cycles 3 → DxDxDxDx 2 Doublewords every 4 cycles 4 → DDxxxDDxxx 2 Doublewords every 5 cycles 5 → DDxxxxDDxxxx 2 Doublewords every 6 cycles 6 → DxxDxxDxxDxx 2 Doublewords every 6 cycles 7 → DDxxxxxxDDxxxxxx 2 Doublewords every 8 cycles 8 → DxxxDxxxDxxxDxxx 2 Doublewords every 8 cycles
SB	Secondary Cache block size. On the V$_R$5000 this is set to 8 words. 1 → 8 words 00, 10, 11 → Reserved
SS	Secondary Cache Size 00 → 512 KB 01 → 1 MB 10 → 2 MB 11 → None
EW	SysAD bus width. On the V$_R$5000 this is set to 64-bit. 00 → 64-bit 01, 10, 11 → Reserved

Field	Description
SC	Secondary Cache present. 0 → Secondary cache present 1 → Secondary cache not present
BE	Big Endian Mode: 0 → Little Endian 1 → Big Endian
EM	ECC mode enable. On the V_R5000 this must be set to parity. 0 → ECC mode 1 → Parity mode
EB	Block ordering. On the V_R5000 this must be set to sub-block. 0 → Sequential 1 → Sub-block
SE	Secondary Cache Enable (software writeable) 0 → Disabled 1 → Enabled
IC	Primary I-cache Size (I-cache size = 2^{12+IC} bytes). In the V_R5000 processor, this must be set to 32 KB.
DC	Primary D-cache Size (D-cache size = 2^{12+DC} bytes). In the V_R5000 processor, this must be set to 32 KB.
IB	Primary I-cache line size. In the V_R5000 processor, this must be set to 32 bytes. 0 → 16 bytes 1 → 32 bytes
DB	Primary D-cache line size. In the V_R5000 processor, this must be set to 32 bytes. 0 → 16 bytes 1 → 32 bytes
K0	*kseg0* coherency algorithm (see *EntryLo0* and *EntryLo1* registers and the *C* field of Table 6-6) (software writeable)

6.5.9 Load Linked Address (LLAddr) Register (17)

The read/write *Load Linked Address* (*LLAddr*) register contains the physical address read by the most recent Load Linked instruction.

This register is for diagnostic purposes only, and serves no function during normal operation.

Figure 6-17 shows the format of the *LLAddr* register; *PAddr* represents bits of the physical address, PA(35:4).

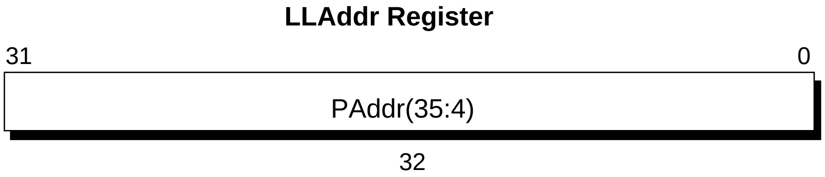


Figure 6-17 LLAddr Register Format

6.5.10 Cache Tag Registers [TagLo (28) and TagHi (29)]

The *TagLo* and *TagHi* registers are 32-bit read/write registers that hold either the primary cache tag and parity, or the secondary cache tag and ECC during cache initialization, cache diagnostics, or cache error processing. The *Tag* registers are written by the CACHE and MTC0 instructions.

The *P* and *ECC* fields of these registers are ignored on Index Store Tag operations. Parity and ECC are computed by the store operation.

Figure 6-18 shows the format of these registers for primary cache operations. Figure 6-19 shows the format of these registers for secondary cache operations.

Table 6-13 lists the field definitions of the *TagLo* and *TagHi* registers.

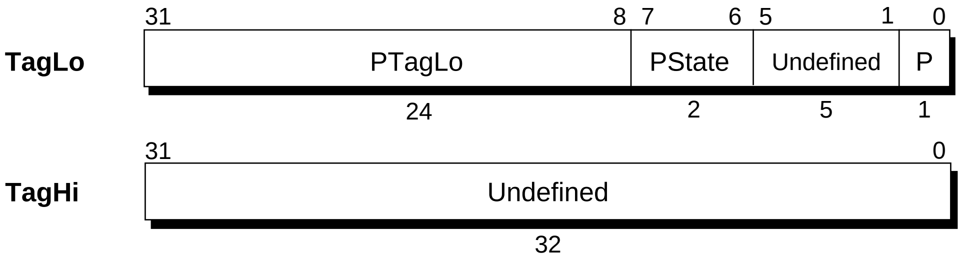


Figure 6-18   TagLo and TagHi Register (P-cache) Formats

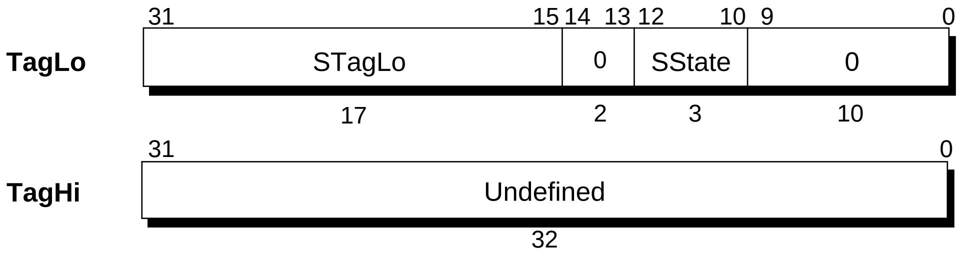


Figure 6-19   TagLo and TagHi Register (S-cache) Formats

Table 6-13 Cache Tag Register Fields

Field	Description
PTagLo	Specifies the physical address bits 35:12
PState	Specifies the primary cache state 0 → Invalid 1 → Reserved 2 → Reserved 3 → Valid
P	Specifies the primary tag even parity bit
STagLo	Specifies the physical address bits 35:19
SState	Specifies the secondary cache state 0 → Invalid 1 → Reserved 2 → Reserved 3 → Reserved 4 → Valid 5 → Reserved 6 → Reserved 7 → Reserved
0	Reserved. Must be written as zeroes, and returns zeroes when read.
Undefined	These fields should not be used.

6.6 Virtual-to-Physical Address Translation Process

During virtual-to-physical address translation, the CPU compares the 8-bit ASID (if the Global bit, *G*, is not set) of the virtual address to the ASID of the TLB entry to see if there is a match. One of the following comparisons are also made:

- In 32-bit mode, the highest 7-to-19 bits (depending upon the page size) of the virtual address are compared to the contents of the TLB virtual page number.
- In 64-bit mode, the highest 15-to-27 bits (depending upon the page size) of the virtual address are compared to the contents of the TLB virtual page number.

If a TLB entry matches, the physical address and access control bits (*C*, *D*, and *V*) are retrieved from the matching TLB entry. While the *V* bit of the entry must be set for a valid translation to take place, it is not involved in the determination of a matching TLB entry.

Figure 6-20 illustrates the TLB address translation process.

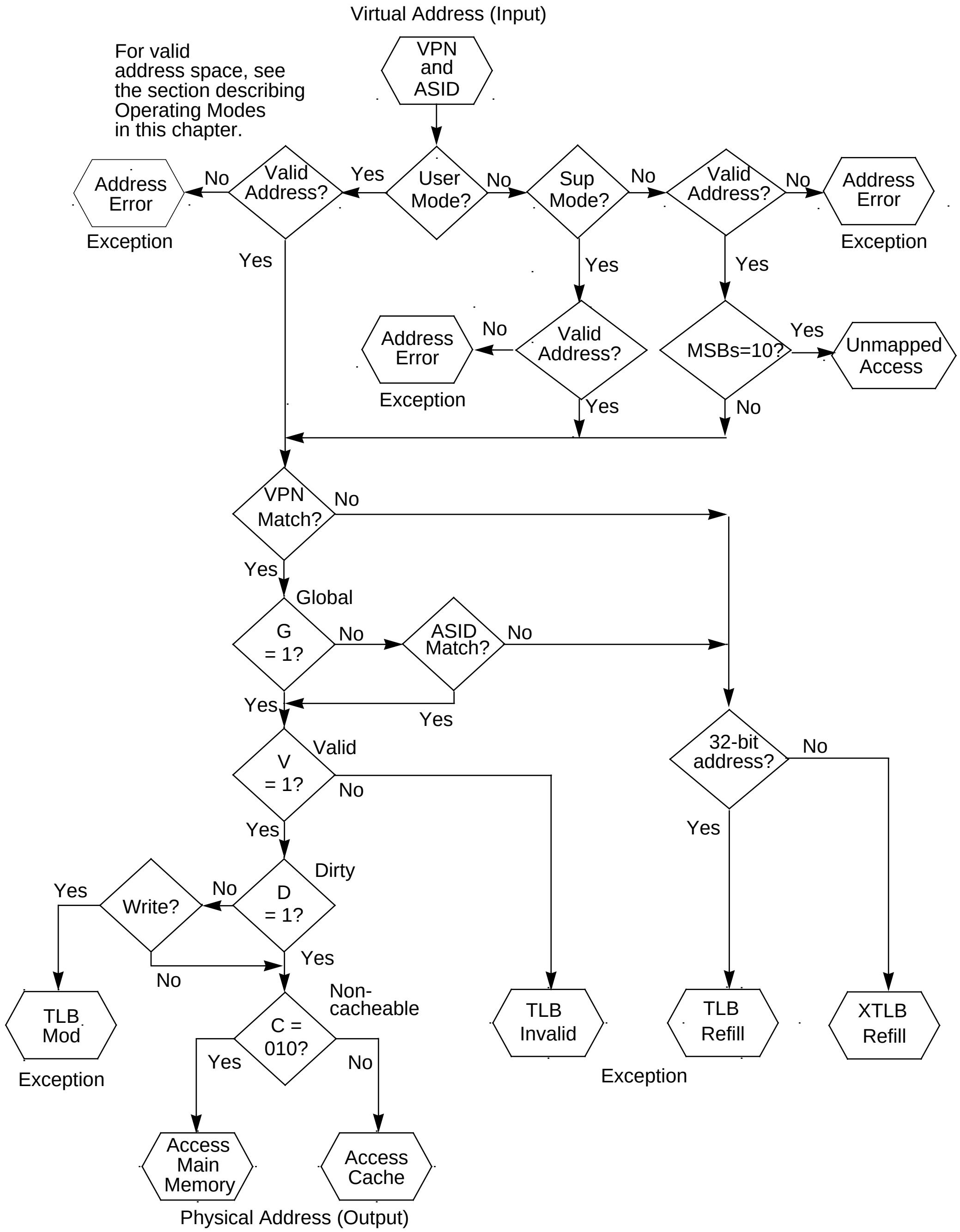


Figure 6-20 TLB Address Translation

6.7 TLB Exceptions

If there is no TLB entry that matches the virtual address, 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 exception occurs. If the *C* bits equal 010_2, the physical address that is retrieved accesses main memory, bypassing the cache.

6.8 TLB Instructions

Table 6-14 lists the instructions that the CPU provides for working with the TLB.

Table 6-14   TLB Instructions

Op Code	Description of Instruction
TLBP	Translation Lookaside Buffer Probe
TLBR	Translation Lookaside Buffer Read
TLBWI	Translation Lookaside Buffer Write Index
TLBWR	Translation Lookaside Buffer Write Random

Chapter 7 CPU Exception Processing

This chapter describes the CPU exception processing, including an explanation of exception processing, followed by the format and use of each CPU exception register.

7.1 Overview of Exception Processing

The processor receives exceptions from a number of sources, including translation lookaside buffer (TLB) misses, arithmetic overflows, I/O interrupts, and system calls. When the CPU detects one of these exceptions, the normal sequence of instruction execution is suspended and the processor enters Kernel mode.

The processor then disables interrupts and forces execution of a software exception processor (called a *handler*) located at a fixed address. The handler saves the context of the processor, including the contents of the program counter, the current operating mode (User or Supervisor), and the status of the interrupts (enabled or disabled). This context is saved so it can be restored when the exception has been serviced.

When an exception occurs, the CPU loads the *Exception Program Counter* (*EPC*) register with a location where execution can restart after the exception has been serviced. The restart location in the *EPC* register 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 registers described later in the section assist in this exception processing by retaining address, cause and status information.

7.2 Exception Processing Registers

This section describes the CP0 registers that are used in exception processing. Table 7-1 lists these registers, along with their number—each register has a unique identification number that is referred to as its *register number*. For instance, the *ECC* register is register number 26. The remaining CP0 registers are used in memory management.

Software examines the CP0 registers during exception processing to determine the cause of the exception and the state of the CPU at the time the exception occurred. The registers in Table 7-1 are used in exception processing, and are described in the sections that follow.

Table 7-1 CP0 Exception Processing Registers

Register Name	**Reg. No.**
Context	4
BadVAddr (Bad Virtual Address)	8
Count	9
Compare register	11
Status	12
Cause	13
EPC (Exception Program Counter)	14
XContext	20
ECC	26
CacheErr (Cache Error and Status)	27
ErrorEPC (Error Exception Program Counter)	30

CPU general registers are interlocked and the result of an instruction can normally be used by the next instruction; if the result is not available right away, the processor stalls until it is available. CP0 registers and the TLB are not interlocked, however; there may be some delay before a value written by one instruction is available to following instructions.

7.2.1 Context Register (4)

The *Context* register is a read/write register containing the 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, the operating system loads the TLB with the missing translation from the PTE array. Normally, the operating system uses the *Context* register to address the current page map which resides in the kernel-mapped segment, *kseg3*. The *Context* register duplicates some of the information provided in the *BadVAddr* register, but the information is arranged in a form that is more useful for a software TLB exception handler. Figure 7-1 shows the format of the *Context* register; Table7-2 describes the *Context* register fields.

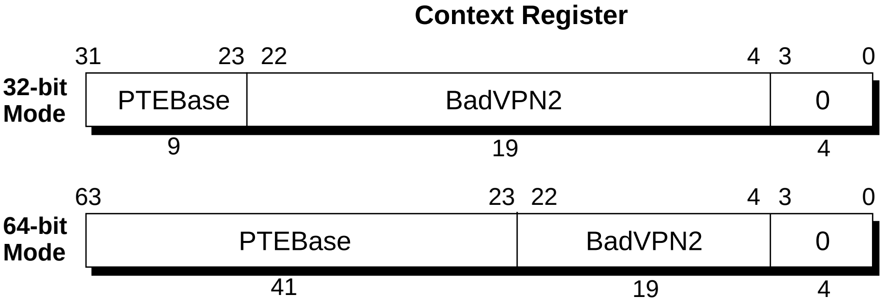


Figure 7-1 Context Register Format

Table 7-2 Context Register Fields

Field	Description
BadVPN2	This field is written by hardware on a miss. It contains the virtual page number (VPN) of the most recent virtual address that did not have a valid translation.
PTEBase	This field is a read/write field for use by the operating system. It is normally written with a value that allows the operating system to use the *Context* register as a pointer into the current PTE array in memory.

The 19-bit *BadVPN2* field contains bits 31:13 of the virtual address that caused the TLB miss; bit 12 is excluded because a single TLB entry maps to an even-odd page pair. For a 4-KB page size, this format can directly address the pair-table of 8-byte PTEs. For other page and PTE sizes, shifting and masking this value produces the appropriate address.

7.2.2 Bad Virtual Address Register (BadVAddr) (8)

The Bad Virtual Address register (*BadVAddr*) is a read-only register that displays the most recent virtual address that caused one of the following exceptions: TLB Invalid, TLB Modified, TLB Refill, or Address Error.

Figure 7-2 shows the format of the *BadVAddr* register.

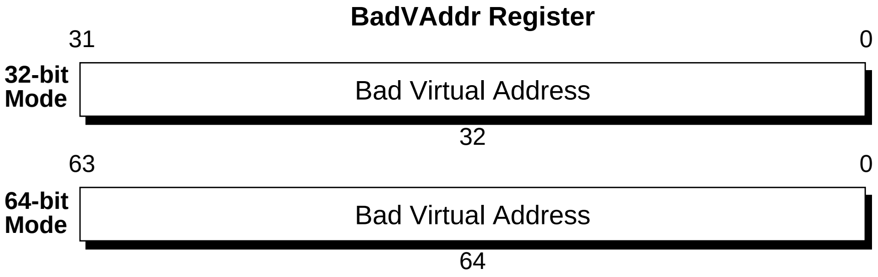


Figure 7-2 BadVAddr Register Format

Note: The *BadVAddr* register does not save any information for bus errors, since bus errors are not addressing errors.

7.2.3 Count Register (9)

The *Count* register acts as a timer incrementing at a constant rate whether or not an instruction is executed, retired, or any forward progress is made through the pipeline. On the V_R5000 the count register can be configured at reset time to count either half the maximum issue rate or at the maximum issue rate. The default behavior is to count at half the maximum issue rate.

This register can be read or written. It can be written for diagnostic purposes or system initialization; for example, to synchronize processors.

Figure 7-3 shows the format of the *Count* register.

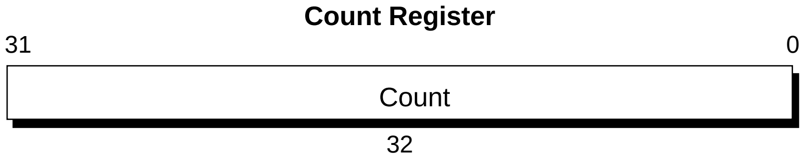


Figure 7-3 Count Register Format

7.2.4 Compare Register (11)

The *Compare* register acts as a timer (see also the *Count* register); it maintains a stable value that does not change on its own.

When the value of the *Count* register equals the value of the *Compare* register, interrupt bit *IP(7)* in the *Cause* register is set. This causes an interrupt as soon as the interrupt is enabled.

Writing a value to the *Compare* register, as a side effect, clears the timer interrupt.

For diagnostic purposes, the *Compare* register is a read/write register. In normal use however, the *Compare* register is write-only. Figure 7-4 shows the format of the *Compare* register.

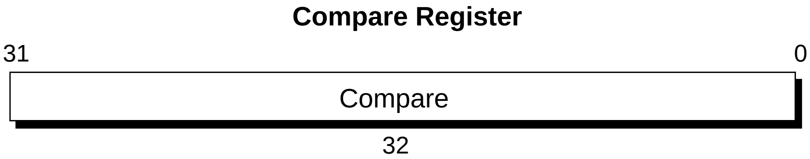


Figure 7-4 Compare Register Format

7.2.5 Status Register (12)

The *Status* register (SR) is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. The following list describes the more important *Status* register fields.

- The 8-bit *Interrupt Mask* (*IM*) field controls the enabling of eight interrupt conditions. Interrupts must be enabled before they can be asserted, and the corresponding bits are set in both the *Interrupt Mask* field of the *Status* register and the *Interrupt Pending* field of the *Cause* register. IM[1:0] are software interrupt masks, while IM[7:2] correspond to Int[5:0].
- The 3-bit *Coprocessor Usability* (*CU*) field controls the usability of 3 possible coprocessors. Regardless of the *CU0* bit setting, CP0 is always usable in Kernel mode. For all other cases, an access to an unusable coprocessor causes an exception.
- The 9-bit *Diagnostic Status* (*DS*) field is used for self-testing, and checks the cache and virtual memory system.

- The *Reverse-Endian (RE)* bit, bit 25, reverses the endianness of the machine. The processor can be configured as either little-endian or big-endian at system reset; reverse-endian selection is used in Kernel and Supervisor modes, and in the User mode when the *RE* bit is 0. Setting the *RE* bit to 1 inverts the User mode endianness.

(1) Status Register Format

Figure 7-5 shows the format of the *Status* register. Table 7-3 describes the *Status* register fields. Figure 7-6 and Table 7-4 provide additional information on the ★ *Diagnostic Status* (*DS*) field. All bits in the *DS* field are readable and writable.

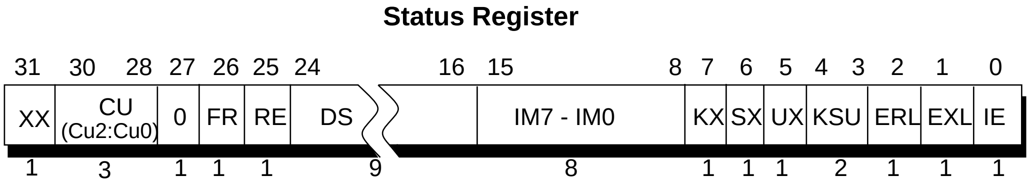


Figure 7-5 Status Register

Table 7-3 Status Register Fields

Field	Description
XX	Enables execution of MIPS IV instructions in user-mode 1 → MIPS IV instructions usable 0 → MIPS IV instructions unusable
CU	Controls the usability of each of the four coprocessor unit numbers. CP0 is always usable when in Kernel mode, regardless of the setting of the CU_0 bit. Setting CU_3 enables the MIPS IV instruction set, 1 → usable 0 → unusable
0	Reserved. Set to 0.
FR	Enables additional floating-point registers 0 → 16 registers 1 → 32 registers
RE	*Reverse-Endian* bit, valid in User mode.
DS	*Diagnostic Status* field (see Figure 7-6).
IM	*Interrupt Mask*: controls the enabling of each of the external, internal, and software interrupts. An interrupt is taken if interrupts are enabled, and the corresponding bits are set in both the *Interrupt Mask* field of the *Status* register and the *Interrupt Pending* field of the *Cause* register. 0 → disabled 1→ enabled
KX	Enables 64-bit addressing in Kernel mode. The extended-addressing TLB refill exception is used for TLB misses on kernel addresses. 0 → 32–bit 1 → 64–bit
SX	Enables 64-bit addressing and operations in Supervisor mode. The extended-addressing TLB refill exception is used for TLB misses on supervisor addresses. 0 → 32–bit 1 → 64–bit
UX	Enables 64-bit addressing and operations in User mode. The extended-addressing TLB refill exception is used for TLB misses on user addresses. 0 → 32–bit 1 → 64–bit

Field	Description
KSU	Mode bits 10_2 → User 01_2 → Supervisor 00_2 → Kernel
ERL	Error Level; set by the processor when Reset, Soft Reset, NMI, or Cache Error exception are taken. 0 → normal 1 → error When ERL is set: Interrupts are disabled. The ERET instruction will use the return address held in ErrorEPC instead of EPC. Kuseg and xkuseg are treated as unmapped and uncached regions.This allows main memory to be accessed in the presence of cache errors.
EXL	Exception Level; set by the processor when any exception other than Reset, Soft Reset, NMI, or Cache Error exception are taken. 0 → normal 1 → exception When EXL is set: Interrupts are disabled. TLB refill exceptions will use the general exception vector instead of the TLB refill vector. EPC will not be updated if another exception is taken.
IE	Interrupt Enable 0 → disable interrupts 1 → enables interrupts

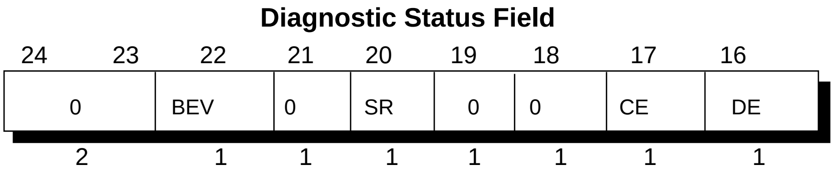


Figure 7-6   Status Register DS Field

Table 7-4   Status Register Diagnostic Status Bits

Bit	Description
BEV	Controls the location of TLB refill and general exception vectors. 0 → normal 1→ bootstrap
0	Reserved. Must be written as zeroes. Returns zeroes when read.
SR	1→ Indicates that a Soft Reset or NMI has occurred.
CE	Contents of the ECC register set or modify the check bits of the caches when CE = 1; see description of the *ECC* register.
DE	Specifies that cache parity or ECC errors cannot cause exceptions. 0 → parity/ECC remain enabled 1 → disables parity/ECC
0	Reserved. Must be written as zeroes, and returns zeroes when read.

(2) Status Register Modes and Access States

Fields of the *Status* register set the modes and access states described in the sections that follow.

Interrupt Enable: Interrupts are enabled when all of the following conditions are true:

- *IE* = 1
- *EXL* = 0
- *ERL* = 0

If these conditions are met, the settings of the *IM* bits enable the interrupt.

Operating Modes: The following CPU *Status* register bit settings are required for User, Kernel, and Supervisor modes.

- The processor is in User mode when *KSU* = 10_2, *EXL* = 0, and *ERL* = 0.
- The processor is in Supervisor mode when *KSU* = 01_2, *EXL* = 0, and *ERL* = 0.

- The processor is in Kernel mode when *KSU* = 00_2, or *EXL* = 1, or *ERL* = 1.

32- and 64-bit Modes: The following CPU *Status* register bit settings select 32- or 64-bit operation for User, Kernel, and Supervisor operating modes. Enabling 64-bit operation permits the execution of 64-bit opcodes and translation of 64-bit addresses. 64-bit operation for User, Kernel and Supervisor modes can be set independently.

- 64-bit addressing for Kernel mode is enabled when *KX* = 1. 64-bit operations are always valid in Kernel mode.
- 64-bit addressing and operations are enabled for Supervisor mode when *SX* = 1.
- 64-bit addressing and operations are enabled for User mode when *UX* = 1.

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

Supervisor Address Space Accesses: Access to the supervisor address space is allowed when the processor is in Kernel or Supervisor mode, as described above in the section titled, Operating Modes.

User Address Space Accesses: Access to the user address space is allowed in any of the three operating modes.

(3) Status Register Reset

The contents of the *Status* register are undefined at reset, except for the following bits in the *Diagnostic Status* field:

- *ERL* and *BEV* = 1

The *SR* bit distinguishes between the Reset exception and the Soft Reset exception (caused either by **Reset*** or Nonmaskable Interrupt [NMI]).

7.2.6 Cause Register (13)

The 32-bit read/write *Cause* register describes the cause of the most recent exception.

Figure 7-7 shows the fields of this register. Table 7-5 describes the *Cause* register fields.

All bits in th*e Cause* register, with the exception of the *IP(1:0)* bits, are read-only; *IP(1:0)* are used for software interrupts.

Table 7-5 Cause Register Fields

Field	Description
BD	Indicates whether the last exception taken occurred in a branch delay slot. 1 → delay slot 0 → normal
CE	Coprocessor unit number referenced when a Coprocessor Unusable exception is taken.
IP	Indicates an interrupt is pending. 1 → interrupt pending 0 → no interrupt
ExcCode	Exception code field (see Table 7-6)
0	Reserved. Must be written as zeroes, and returns zeroes when read.

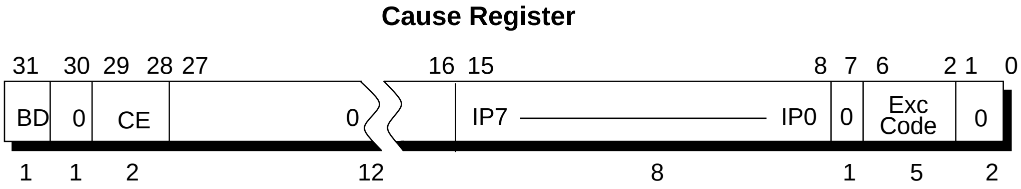


Figure 7-7 Cause Register Format

Table 7-6 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	IBE	Bus error exception (instruction fetch)
7	DBE	Bus error exception (data reference: load or store)
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

7.2.7 Exception Program Counter (EPC) Register (14)

The Exception Program Counter (*EPC*) is a read/write register that contains the address at which 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, or
- 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).

The processor does not write to the *EPC* register when the *EXL* bit in the *Status* register is set to a 1.

Figure 7-8 shows the format of the *EPC* register.

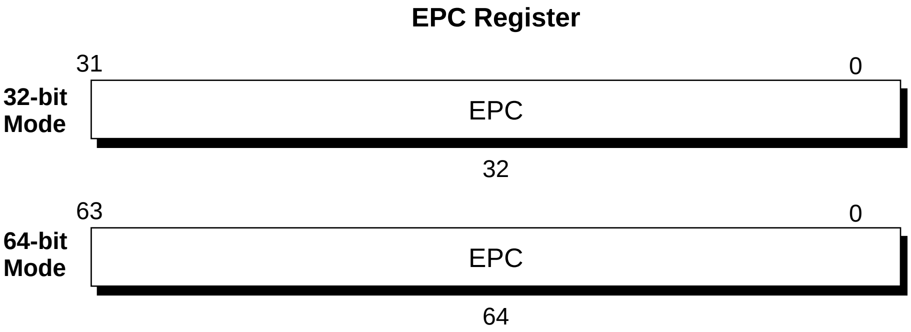


Figure 7-8 EPC Register Format

7.2.8 XContext Register (20)

The read/write *XContext* register contains a pointer to an entry in the page table entry (PTE) array, an operating system data structure that stores virtual-to-physical address translations. When there is a TLB miss, the operating system software loads the TLB with the missing translation from the PTE array. The *XContext* register duplicates some of the information provided in the *BadVAddr* register, and puts it in a form useful for a software TLB exception handler. The *XContext* register is for use with the XTLB refill handler, which loads TLB entries for references to a 64-bit address space, and is included solely for operating system use. The operating system sets the PTE base field in the register, as needed. Normally, the operating system uses the *Context* register to address the current page map, which resides in the kernel-mapped segment *kseg3*. Figure 7-9 shows the format of the *XContext* register; Table 7-7 describes the *XContext* register fields.

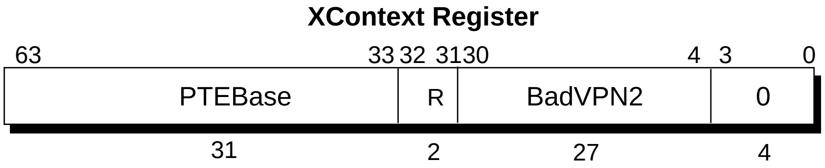


Figure 7-9 XContext Register Format

The 27-bit *BadVPN2* field has bits 39:13 of the virtual address that caused the TLB miss; bit 12 is excluded because a single TLB entry maps to an even-odd page pair. For a 4-KB page size, this format may be used directly to address the pair-table of 8-byte PTEs. For other page and PTE sizes, shifting and masking this value produces the appropriate address.

Table 7-7 XContext Register Fields

Field	Description
BadVPN2	The *Bad Virtual Page Number*/2 field is written by hardware on a miss. It contains the VPN of the most recent invalidly translated virtual address.
R	The *Region* field contains bits 63:62 of the virtual address. 00_2 = user 01_2 = supervisor 11_2 = kernel.
PTEBase	The *Page Table Entry Base* read/write field is normally written with a value that allows the operating system to use the *Context* register as a pointer into the current PTE array in memory.

7.2.9 Error Checking and Correcting (ECC) Register (26)

The 8-bit *Error Checking and Correcting* (*ECC*) register reads or writes primary-cache data parity bits for cache initialization, cache diagnostics, or cache error processing. (Tag ECC and parity are loaded from and stored to the *TagLo* register.)

The *ECC* register is loaded by the Index Load Tag CACHE operation. Content of the ECC register is:

- written into the primary data cache on store instructions (instead of the computed parity) when the *CE* bit of the *Status* register is set.
- substituted for the computed instruction parity for the CACHE operation Fill.

Figure 7-10 shows the format of the *ECC* register; Table 7-8 describes the register fields.

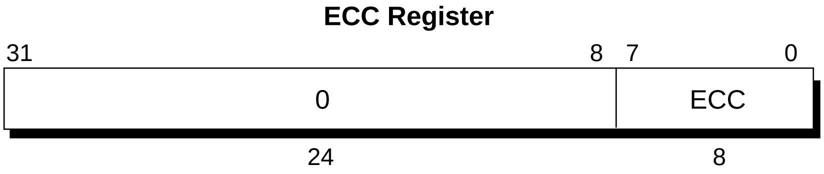


Figure 7-10 ECC Register Format

Table 7-8 ECC Register Fields

Field	Description
ECC	An 8-bit field specifying the parity bits read from or written to a primary cache. ECC field values for Index_Store_Tag_D, Index_Load_Tag_D cache operations: ECC[0] Even parity for least significant byte of requested doubleword ECC[1] Even parity for 2nd least significant byte ECC[2] Even parity for 3rd least significant byte ECC[3] Even parity for 4th least significant byte ECC[4] Even parity for 4th most significant byte ECC[5] Even parity for 3rd most significant byte ECC[6] Even parity for 2nd most signficant byte ECC[7] Even parity for most significant byte of requested doubleword ECC field values for Index_Store_Tag_I, Index_Load_Tag_I cache operations: ECC[0] Even parity for least significant word of requested doubleword ECC[1] Even parity for most significant word of requested doubleword
0	Reserved. Must be written as zeroes, and returns zeroes when read.

7.2.10 Cache Error (CacheErr) Register (27)

The 32-bit read-only *CacheErr* register processes ECC errors in the secondary cache and parity errors in the primary cache. Parity errors cannot be corrected.

The *CacheErr* register holds cache index and status bits that indicate the source and nature of the error; it is loaded when a Cache Error exception is asserted.

Figure 7-11 shows the format of the *CacheErr* register and Table 7-9 describes the *CacheErr* register fields.

CacheErr Register

31	30	29	28	27	26	25	24	23	22	21 … 3	2	1 0
ER	EC	ED	ET	0	EE	EB	EI	0	0	SIDX	0	PIDX
1	1	1	1	1	1	1	1	1	1	19	1	2

Figure 7-11 CacheErr Register Format

Table 7-9 CacheErr Register Fields

Field	Description
ER	Type of reference 0 → instruction 1 → data
EC	Cache level of the error 0 → primary 1 → reserved
ED	Indicates if a data field error occurred 0 → no error 1 → error
ET	Indicates if a tag field error occurred 0 → no error 1 → error
EE	This bit is set if the error occurred on the SysAD bus.
EB	This bit is set if a data error occurred in addition to the instruction error (indicated by the remainder of the bits). If so, this requires flushing the data cache after fixing the instruction error.
EI	This bit is set if the error occured on filling primary on store miss.
SIDX	Physical address [21:3] of the reference that encountered the error
PIDX	Virtual address [13:12] of the double word in error. (used with SIDX to construct a virtual index for the primary caches)
0	Reserved. Must be written as zeroes, and returns zeroes when read.

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

The *ErrorEPC* register is similar to the *EPC* register, except that *ErrorEPC* is used on parity error exceptions. It is also used to store the program counter (PC) on Reset, Soft Reset, and nonmaskable interrupt (NMI) exceptions.

The read/write *ErrorEPC* register contains the virtual address at which instruction processing can resume after servicing an error. This address can be:

- the virtual address of the instruction that caused the exception
- the virtual address of the immediately preceding branch or jump instruction, when this address is in a branch delay slot.

There is no branch delay slot indication for the *ErrorEPC* register.

Figure 7-12 shows the format of the *ErrorEPC* register.

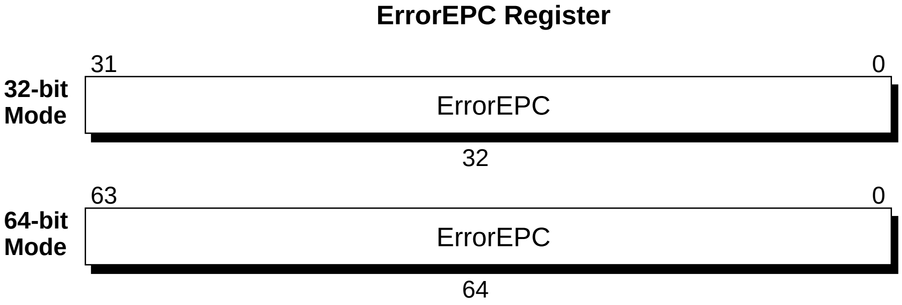


Figure 7-12 ErrorEPC Register Format

7.3 Processor Exceptions

This section describes the processor exceptions—it describes the cause of each exception, its processing by the hardware, and servicing by a handler (software). The types of exception, with exception processing operations, are described in the next section.

7.3.1 Exception Types

This section gives sample exception handler operations for the following exception types:

- reset
- soft reset
- nonmaskable interrupt (NMI)
- cache error
- remaining processor exceptions

When the *EXL* bit in the *Status* register is 0, either User, Supervisor, or Kernel operating mode is specified by the *KSU* bits in the *Status* register. When the *EXL* bit is a 1, the processor is in Kernel mode.

When the processor takes an exception, the *EXL* bit is set to 1, which means the system is in Kernel mode. After saving the appropriate state, the exception handler typically changes *KSU* to Kernel mode and resets the *EXL* bit back to 0. When restoring the state and restarting, the handler restores the previous value of the *KSU* field and sets the *EXL* bit back to 1.

Returning from an exception, also resets the *EXL* bit to 0.

In the following sections, sample hardware processes for various exceptions are shown, together with the servicing required by the handler (software).

(1) Reset Exception Process

Figure 7-13 shows the Reset exception process.

```
T:  undefined
    Random ← TLBENTRIES–1
    Wired ← 0
    Config ← 0 || EC || EP || 00000000 || BE || 110 || 010 || 1 || 1 || 0 || undefined
                || DC || undefined[6]
    ErrorEPC ← PC
    SR ← $SR_{31:23}$ || 1 || 0 || 0 || $SR_{19:3}$ || 1 || $SR_{1:0}$
    PC ← 0xFFFF FFFF BFC0 0000
```

Figure 7-13 Reset Exception Processing

(2) Cache Error Exception Process

Figure 7-14 shows the Cache Error exception process.

```
T:  ErrorEPC ← PC
    CacheErr ← ER || EC || ED || ET || ES || EE || ED || $0^{25}$
    SR ← $SR_{31:3}$ || 1 ||$SR_{1:0}$
    if $SR_{22}$ = 1 then      /*What is the BEV bit setting*/
       PC ← 0xFFFF FFFF BFC0 0200 + 0x100  /*Access boot-PROM area*/
    else
       PC ← 0xFFFF FFFF A000 0000 + 0x100  /*Access main memory area*/
    endif
```

Figure 7-14 Cache Error Exception Processing

(3) Soft Reset and NMI Exception Process

Figure 7-15 shows the Soft Reset and NMI exception process.

```
T:  ErrorEPC ← PC
    SR ← $SR_{31:23}$ || 1 || 0 || 1 || $SR_{19:3}$ || 1 || $SR_{1:0}$
    PC ← 0xFFFF FFFF BFC0 0000
```

Figure 7-15 Soft Reset and NMI Exception Processing

(4) General Exception Process

Figure 7-16 shows the process used for exceptions other than Reset, Soft Reset, NMI, and Cache Error.

```
T:   Cause ← BD || 0 || CE || $0^{12}$ || $Cause_{15:8}$ || ExcCode || $0^{2}$
if $SR_1$ = 0 then/* System is in User or Supervisor mode with no current exception */
               EPC ← PC
    endif
    SR ← $SR_{31:2}$ || 1 || $SR_0$
    if $SR_{22}$ = 1 then
       PC ← 0xFFFF FFFF BFC0 0200 + vector  /*access to uncached space*/
    else
       PC ← 0xFFFF FFFF 8000 0000 + vector  /*access to cached space*/
    endif
```

Figure 7-16 General Exception Processing

7.3.2 Exception Vector Locations

The Reset, Soft Reset, and NMI exceptions are always vectored to location 0xFFFF FFFF BFC0 0000. Addresses for all other exceptions are a combination of a *vector offset* and a *base address*.

The base addres is determined by the BEV bit of the *Status* register.

Table 7-10 shows the 64-bit-mode vector base address for all exceptions; the 32-bit mode address is the low-order 32 bits (for instance, the base address for NMI in 32-bit mode is 0xBFC0 0000).

Table 7-11 shows the vector offset added to the base address to create the exception address.

Table 7-10 Exception Vector Base Addresses

BEV Bit	V_R5000 Processor Vector Base Address
0	0xFFFF FFFF 8000 0000
1	0xFFFF FFFF BFC0 0200

Table 7-11 Exception Vector Offsets

Exception	V_R5000 Processor Vector Offset
TLB refill, EXL = 0	0x000
XTLB refill, EXL = 0 (X = 64-bit TLB)	0x080
Cache Error	0x100
Others	0x180

When **BEV = 0**, the vector base address for the cache error exception changes from *kseg0* (0xFFFF FFFF 8000 0000) to *kseg1* (0xFFFF FFFF A000 0000). This change indicates that the caches are initialized and that the vector can be cached. When **BEV = 1**, the vector base for the cache error exception is 0xFFFF FFFF BFC0 0200. This is an uncached and unmapped space, allowing the exception to bypass the cache and the TLB.

(1) TLB Refill Vector Selection

In all present implementations of the MIPS III ISA, there are two TLB refill exception vectors:

- one for references to 32-bit address space (TLB Refill)
- one for references to 64-bit address space (XTLB Refill)

The TLB refill vector selection is based on the address space of the address (*user*, *supervisor*, or *kernel*) that caused the TLB miss, and the value of the corresponding extended addressing bit in the *Status* register (*UX*, *SX*, or *KX*). The current operating mode of the processor is not important except that it plays a part in specifying in which address space an address resides. The *Context* and *XContext* registers are entirely separate page-table-pointer registers that point to and refill from two separate page tables. For all TLB exceptions (Refill, Invalid, TLBL or TLBS), the *BadVPN2* fields of both registers are loaded as they were in the V_R4000.

In contrast to the V_R5000, the V_R4000 processor selects the vector based on the current operating mode of the processor (*user*, *supervisor*, or *kernel*) and the value of the corresponding extended addressing bit in the *Status* register (*UX*, *SX* or *KX*). In addition, the *Context* and *XContext* registers are not implemented as entirely separate registers; the *PTEbase* fields are shared. A miss to a particular address goes through either TLB Refill or XTLB Refill, depending on the source of the reference. There can be only a single page table unless the refill handlers execute address-deciphering and page table selection in software.

Note: Refills for the 0.5 GB supervisor mapped region, *sseg/ksseg*, are controlled by the value of *KX* rather than *SX*. This simplifies control of the procesor when supervisor mode is not being used.

Table 7-12 lists the TLB refill vector locations, based on the adress that caused the TLB miss and its correspoinding mode bit.

Table 7-12 TLB Refill Vectors

Space	Address Range	Regions	Exception Vector
Kernel	0xFFFF FFFF E000 0000 to 0xFFFF FFFF FFFF FFFF	*kseg3*	Refill (KX=0) or XRefill (KX=1)
Supervisor	0xFFFF FFFF C000 0000 to 0xFFFF FFFF DFFF FFFF	*sseg, ksseg*	Refill (SX=0) or XRefill (SX=1)
Kernel	0xC000 0000 0000 0000 to 0xC000 0FFE FFFF FFFF	*xkseg*	XRefill (KX=1)
Supervisor	0x4000 0000 0000 0000 to 0x4000 0FFF FFFF FFFF	*xsseg, xksseg*	XRefill (SX=1)
User	0x0000 0000 8000 0000 to 0x0000 0FFF FFFF FFFF	*xsuseg, xuseg, xkuseg*	XRefill (UX=1)
User	0x0000 0000 0000 0000 to 0x0000 0000 7FFF FFFF	*useg, xuseg, suseg,* *xsuseg, kuseg, xkuseg*	Refill (UX=0) or XRefill (UX=1)

7.3.3 Priority of Exceptions

Table 7-13 describes exceptions in the order of highest to lowest priority. While more than one exception can occur for a single instruction, only the exception with the highest priority is reported.

Table 7-13 Exception Priority Order

Reset *(highest priority)*
Soft Reset
Nonmaskable Interrupt (NMI)
Address error — Instruction fetch
TLB refill — Instruction fetch
TLB invalid — Instruction fetch
Cache error — Instruction fetch
Bus error — Instruction fetch
Integer overflow, Trap, System Call, Breakpoint, Reserved Instruction, Coprocessor Unusable, or Floating-Point Exception
Address error — Data access
TLB refill — Data access
TLB invalid — Data access
TLB modified — Data write
Cache error — Data access
Bus error — Data access
Interrupt *(lowest priority)*

Generally speaking, the exceptions described in the following sections are handled (“processed”) by hardware; these exceptions are then serviced by software.

7.3.4 Reset Exception

Cause

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

Processing

The CPU provides a special interrupt vector for this exception:

- location 0xFFFF FFFF BFC0 0000 in 64-bit mode

The Reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to process this exception. It also means 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 this exception occurs, except for the following register fields:

- In the *Status* register, *SR* is cleared to 0, and *ERL* and *BEV* are set to 1. All other bits are undefined.
- Some *Config* register are initialized from the boot-time mode stream.
- The *Random* register is initialized to the value of its upper bound.
- The *Wired* register is initialized to 0.

Servicing

The Reset exception is serviced by:

- initializing all processor registers, coprocessor registers, caches, and the memory system
- performing diagnostic tests
- bootstrapping the operating system

7.3.5 Soft Reset Exception

Cause

The Soft Reset exception occurs in response to assertion of the **Reset*** input Execution begins at the Reset vector when the **Reset*** signal is negated.

The Soft Reset exception is not maskable.

Processing

The Reset vector is used for this exception. The Reset vector is located within uncached and unmapped address space. Hence the cache and TLB need not be initialized in order to process the exception. Regardless of the cause, when this exception occurs the *SR* bit of the *Status* register is set, distinguishing this exception from a Reset exception.

The primary purpose of the Soft Reset exception is to reinitialize the processor after a fatal error during normal operation. Unlike an NMI, all cache and bus state machines are reset by this exception.

When the Soft Reset exception occurs, all register contents are preserved with the following exceptions:

- *ErrorEPC* register, which contains the restart PC.
- *ERL, BEV,* and *SR* bits of the Status Register, each of which is set to 1.

Because the Soft Reset can abort cache and bus operations, the cache and memory states are undefined when the Soft Reset exception occurs.

Servicing

The Soft Reset exception is serviced by saving the current processor state for diagnostic purposes, and reinitializing for the Reset exception.

7.3.6 Non Maskable Interrupt (NMI) Exception

Cause

The Non Maskable Interrupt exception occurs in response to falling edge of the NMI signal, or an external write to the **Int*[6]** bit of the *Interrupt* Register. The NMI interrupt is not maskable and occurs regardless of the settings of the *EXL, ERL,* and *IE* bits in the *Status* Register.

Processing

The Reset vector is used for this exception. The Reset vector is located within uncached and unmapped address space. Hence the cache and TLB need not be initialized in order to process the exception. Regardless of the cause, when this exception occurs the *SR* bit of the *Status* register is set, distinguishing this exception from a Reset exception.

Because the NMI can occur in the midst of another exception, it is typically not possible to continue program execution after servicing an NMI. An NMI exception is taken only at instruction boundaries. The state of the caches and memory system are preserved.

When the NMI exception occurs, all register contents are preserved with the following exceptions:

- ErrorEPC register, which contains the restart PC.
- ERL, BEV, and SR bits of the Status Register, each of which is set to 1.

Servicing

The NMI exception is serviced by saving the current processor state for diagnostic purposes, and reinitializing for the Reset exception.

Caution If a pipeline cancelling logic (e.g. cache error, bus error) occurs after the V$_R$5000 detects an NMI by the V$_R$5000 starts the NMI handling, the NMI will be cancelled and only the pipeline cancelling logic will be handled.
If an NMI cancellation occurred, make NMI* inactive once and then make it active again after the NMI cancellation.

7.3.7 Address Error Exception

Cause

The Address Error exception occurs when an attempt is made to execute one of the following:

- load or store a doubleword that is not aligned on a doubleword boundary
- 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 or Supervisor mode
- reference the supervisor address space from User mode

This exception is not maskable.

Processing

The common exception vector is used for this exception. The *AdEL* or *AdES* code in the *Cause* register is set, indicating whether the instruction caused the exception with an instruction reference, load operation, or store operation shown by the *EPC* register and *BD* bit in the *Cause* register.

When this 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 contains the address of the instruction that caused the exception, unless this instruction is in a branch delay slot. If it is in a branch delay slot, the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set as indication.

Servicing

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

Restriction

An address error exception will erroneously occur on a branch instruction that is the second to last instruction of a segment (e.g., USEG0).

7.3.8 TLB Exceptions

Three types of TLB exceptions can occur:

- TLB Refill occurs when there is no TLB entry that matches an attempted reference to a mapped address space.
- TLB Invalid occurs when a virtual address reference matches a TLB entry that is marked invalid.
- TLB Modified occurs when a store operation virtual address reference to memory matches a TLB entry which is marked valid but is not dirty (the entry is not writable).

The following three sections describe these TLB exceptions.

TLB Refill Exception

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.

Processing

There are two special exception vectors for this exception; one for references to 32-bit address spaces, and one for references to 64-bit address spaces. The *UX*, *SX*, and *KX* bits of the *Status* register determine whether the user, supervisor or kernel address spaces referenced are 32-bit or 64-bit spaces. All references use these vectors when the *EXL* bit is set to 0 in the *Status* register. This exception sets the *TLBL* or *TLBS* code in the *ExcCode* field of the *Cause* register. This code indicates whether the instruction, as shown by the *EPC* register and the *BD* bit in the *Cause* register, caused the miss by an instruction reference, load operation, or store operation.

When this exception occurs, the *BadVAddr*, *Context*, *XContext* and *EntryHi* registers hold the virtual address that failed address 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 contains the address of the instruction that caused the exception, unless this instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing

To service this exception, the contents of the *Context* or *XContext* register are used as a virtual address to fetch memory locations containing the physical page frame and access control bits for a pair of TLB entries. The two entries are placed into the E*ntryLo0/EntryLo1* register; the *EntryHi* and *EntryLo* registers are 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. This condition is processed by allowing a TLB refill exception in the TLB refill handler. This second exception goes to the common exception vector because the *EXL* bit of the *Status* register is set.

TLB Invalid Exception

Cause

The TLB invalid exception occurs when a virtual address reference matches a TLB entry that is marked invalid (TLB valid bit cleared). This exception is not maskable.

Processing

The common exception vector is used for this exception. The *TLBL* or *TLBS* code in the *ExcCode* field of the *Cause* register is set. This indicates whether the instruction, as shown by the *EPC* register and *BD* bit in the *Cause* register, caused the miss by an instruction reference, load operation, or store operation.

When this exception occurs, the *BadVAddr*, *Context*, *XContext* and *EntryHi* registers contain the virtual address that failed address 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 put the replacement TLB entry. The contents of the *EntryLo* register is undefined.

The *EPC* register contains the address of the instruction that caused the exception unless this instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing

A TLB entry is typically marked invalid when one of the following is true:

- 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 a TLB Invalid exception, the TLB entry is located with TLBP (TLB Probe), and replaced by an entry with that entry’s *Valid* bit set.

TLB Modified Exception

Cause

The TLB modified exception occurs when a store operation virtual address reference to memory matches a TLB entry that is marked valid but is not dirty and therefore is not writable. This exception is not maskable.

Processing

The common exception vector is used for this exception, and the *Mod* code in the *Cause* register is set.

When this exception occurs, the *BadVAddr*, *Context*, *XContext* and *EntryHi* registers contain the virtual address that failed address translation. The *EntryHi* register also contains the ASID from which the translation fault occurred. The contents of the *EntryLo* register is undefined.

The *EPC* register contains the address of the instruction that caused the exception unless that instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing

The kernel uses the failed virtual address or virtual page number to identify the corresponding access control information. The page identified may or may not permit write accesses; if writes are not permitted, a write protection violation occurs.

If write accesses are permitted, the page frame is marked dirty/writable by the kernel in its own data structures. The TLBP instruction places the index of the TLB entry that must be altered into the *Index* register. The *EntryLo* register is loaded with a word containing the physical page frame and access control bits (with the *D* bit set), and the *EntryHi* and *EntryLo* registers are written into the TLB.

7.3.9 Cache Error Exception

Cause

The Cache Error exception occurs when either a primary or secondary cache parity error is detected. This exception is maskable by the *DE* bit in the Status Register.

Processing

The processor sets the *ERL* bit in the *Status* register, saves the exception restart address in the *ErrorEPC* register, and then transfers the information to a special vector in uncached space;

If BEV = 0, the vector is 0xFFFF FFFF A000 0100.

If BEV = 1, the vector is 0xFFFF FFFF BFC0 0300.

Servicing

All errors should be logged. To correct parity errors the system uses the CACHE instruction to invalidate the cache block, overwrite the old data through a cache miss, and resumes execution with an ERET. Other errors are not correctable and are likely to be fatal to the current process.

7.3.10 Bus Error Exception

Cause

A Bus Error exception is raised by board-level circuitry for events such as bus time-out, backplane bus parity errors, and invalid physical memory addresses or access types. This exception is not maskable.

A Bus Error exception occurs when a cache miss refill, uncached reference, or an unbuffered write occurs synchronously; a Bus Error exception resulting from a buffered write transaction must be reported using the general interrupt mechanism.

Processing

The common interrupt vector is used for a Bus Error exception. The *IBE* or *DBE* code in the *ExcCode* field of the *Cause* register is set, signifying whether the instruction (as indicated by the *EPC* register and *BD* bit in the *Cause* register) caused the exception by an instruction reference, load operation, or store operation.

The *EPC* register contains the address of the instruction that caused the exception, unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing

The physical address at which the fault occurred can be computed from information available in the CP0 registers.

- If the *IBE* code in the *Cause* register is set (indicating an instruction fetch reference), the virtual address is contained in the *EPC* register.
- If the *DBE* code is set (indicating a load or store reference), the instruction that caused the exception is located at the virtual address contained in the *EPC* register (or 4+ the contents of the *EPC* register if the *BD* bit of the *Cause* register is set).

The virtual address of the load and store reference can then be obtained by interpreting the instruction. The physical address can be obtained by using the TLBP instruction and reading the *EntryLo* register to compute the physical page number. The process executing at the time of this exception is handed a bus error signal, which is usually fatal.

7.3.11 Integer Overflow Exception

Cause

An Integer Overflow exception occurs when an ADD, ADDI, SUB, DADD, DADDI or DSUB instruction results in a 2’s complement overflow. This exception is not maskable.

Processing

The common exception vector is used for this exception, and the *OV* code in the *Cause* register is set.

The *EPC* register contains the address of the instruction that caused the exception unless the instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing

The process executing at the time of the exception is handed a floating-point exception/integer overflow signal. This error is usually fatal to the current process.

7.3.12 Trap Exception

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.

Processing

The common exception vector is used for this exception, and the *Tr* code in the *Cause* register is set.

The *EPC* register contains the address of the instruction causing the exception unless the instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing

The process executing at the time of a Trap exception is handed a floating-point exception/integer overflow signal. This error is usually fatal.

7.3.13 System Call Exception

Cause

A System Call exception occurs during an attempt to execute the SYSCALL instruction. This exception is not maskable.

Processing

The common exception vector is used for this exception, and the *Sys* code in the *Cause* register is set.

The *EPC* register contains the address of the SYSCALL instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction.

If the SYSCALL instruction is in a branch delay slot, the *BD* bit of the *Status* register is set; otherwise this bit is cleared.

Servicing

When this exception occurs, control is transferred to the applicable system routine.

To resume execution, the *EPC* register must be altered so that the SYSCALL instruction does not re-execute; this is accomplished by adding a value of 4 to the *EPC* register (*EPC* register + 4) before returning.

If a SYSCALL instruction is in a branch delay slot, a more complicated algorithm, beyond the scope of this description, may be required.

7.3.14 Breakpoint Exception

Cause

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

Processing

The common exception vector is used for this exception, and the *BP* code in the *Cause* register is set.

The *EPC* register contains the address of the BREAK instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction.

If the BREAK instruction is in a branch delay slot, the *BD* bit of the *Status* register is set, otherwise the bit is cleared.

Servicing

When the Breakpoint exception occurs, control is transferred to the applicable system routine. Additional distinctions can be made by analyzing the unused bits of the BREAK instruction (bits 25:6), and loading the contents of the instruction whose address the *EPC* register contains. A value of 4 must be added to the contents of the *EPC* register *(EPC* register + 4) to locate the instruction if it resides in a branch delay slot.

To resume execution, the *EPC* register must be altered so that the BREAK instruction does not re-execute; this is accomplished by adding a value of 4 to the *EPC* register (*EPC* register + 4) before returning.

If a BREAK instruction is in a branch delay slot, interpretation of the branch instruction is required to resume execution.

7.3.15 Reserved Instruction Exception

Cause

The Reserved Instruction exception occurs when one of the following conditions occurs:

- an attempt is made to execute an instruction with an undefined major opcode (bits 31:26)
- an attempt is made to execute a SPECIAL instruction with an undefined minor opcode (bits 5:0)
- an attempt is made to execute a REGIMM instruction with an undefined minor opcode (bits 20:16)
- an attempt is made to execute 64-bit operations in 32-bit mode when in User or Supervisor modes

64-bit operations are always valid in Kernel mode regardless of the value of the *KX* bit in the *Status* register.

This exception is not maskable.

Processing

The common exception vector is used for this exception, and the *RI* code in the *Cause* register is set.

The *EPC* register contains the address of the reserved instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction.

Servicing

No instructions in the MIPS ISA are currently interpreted. The process executing at the time of this exception is handed an illegal instruction/reserved operand fault signal. This error is usually fatal.

7.3.16 Coprocessor Unusable Exception

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
- CP0 instructions, when the unit has not been marked usable and the process executes in either User or Supervisor mode.

This exception is not maskable.

Processing

The common exception vector is used for this exception, and the *CPU* code in the *Cause* register is set. The contents of the *Coprocessor Usage Error* field of the coprocessor *Control* register indicate which of the four coprocessors was referenced. The *EPC* register contains the address of the unusable coprocessor instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction.

Servicing

The coprocessor unit to which an attempted reference was made is identified by the Coprocessor Usage Error field, which results in one of the following situations:

- If the process is entitled access to the coprocessor, the coprocessor is marked usable and the corresponding user state is restored to the coprocessor.
- If the process is entitled access to the coprocessor, but the coprocessor does not exist or has failed, interpretation of the coprocessor instruction is possible.

- If the *BD* bit is set in the *Cause* register, the branch instruction must be interpreted; then the coprocessor instruction can be emulated and execution resumed with the *EPC* register advanced past the coprocessor instruction.
- If the process is not entitled access to the coprocessor, the process executing at the time is handed an illegal instruction/privileged instruction fault signal. This error is usually fatal.

7.3.17 Floating-Point Exception

Cause

The Floating-Point exception is used by the floating-point coprocessor. This exception is not maskable.

Processing

The common exception vector is used for this exception, and the *FPE* code in the *Cause* register is set.

The contents of the *Floating-Point Control/Status* register indicate the cause of this exception.

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 program that caused the exception.

7.3.18 Interrupt Exception

Cause

The Interrupt exception occurs when one of the eight interrupt conditions is asserted. The significance of these interrupts is dependent upon 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, and all of the eight interrupts can be masked at once by clearing the *IE* bit of the *Status* register.

Processing

The common exception vector is used for this exception, and the *Int* code in the *Cause* register is set.

The *IP* field of the *Cause* register indicates current interrupt requests. It is possible that more than one of the bits can be simultaneously set (or even *no* bits may be set) if the interrupt is asserted and then deasserted before this register is read.

Servicing

If the interrupt is caused by one of the two software-generated exceptions (*SW1* or *SW0*), 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.

Due to the on-chip write buffer, a store to an external device may not occur until after other instructions in the pipeline finish. Hence, the user must ensure that the store will occur before the *return from exception* instruction (ERET) is executed. Otherwise the interrupt may be serviced again even though there is no actual interrupt pending.

7.4 Exception Handling and Servicing Flowcharts

The remainder of this section contains flowcharts for the following exceptions and guidelines for their handlers:

- general exceptions and their exception handler
- TLB/XTLB miss exception and their exception handler
- cache error exception and its handler
- reset, soft reset and NMI exceptions, and a guideline to their handler.

Generally speaking, the exceptions are handled by hardware (HW); the exceptions are then serviced by software (SW).

Exceptions other than Reset, Soft Reset, NMI, CacheError or first-level TLB miss
Note: Interrupts can be masked by IE or IMs

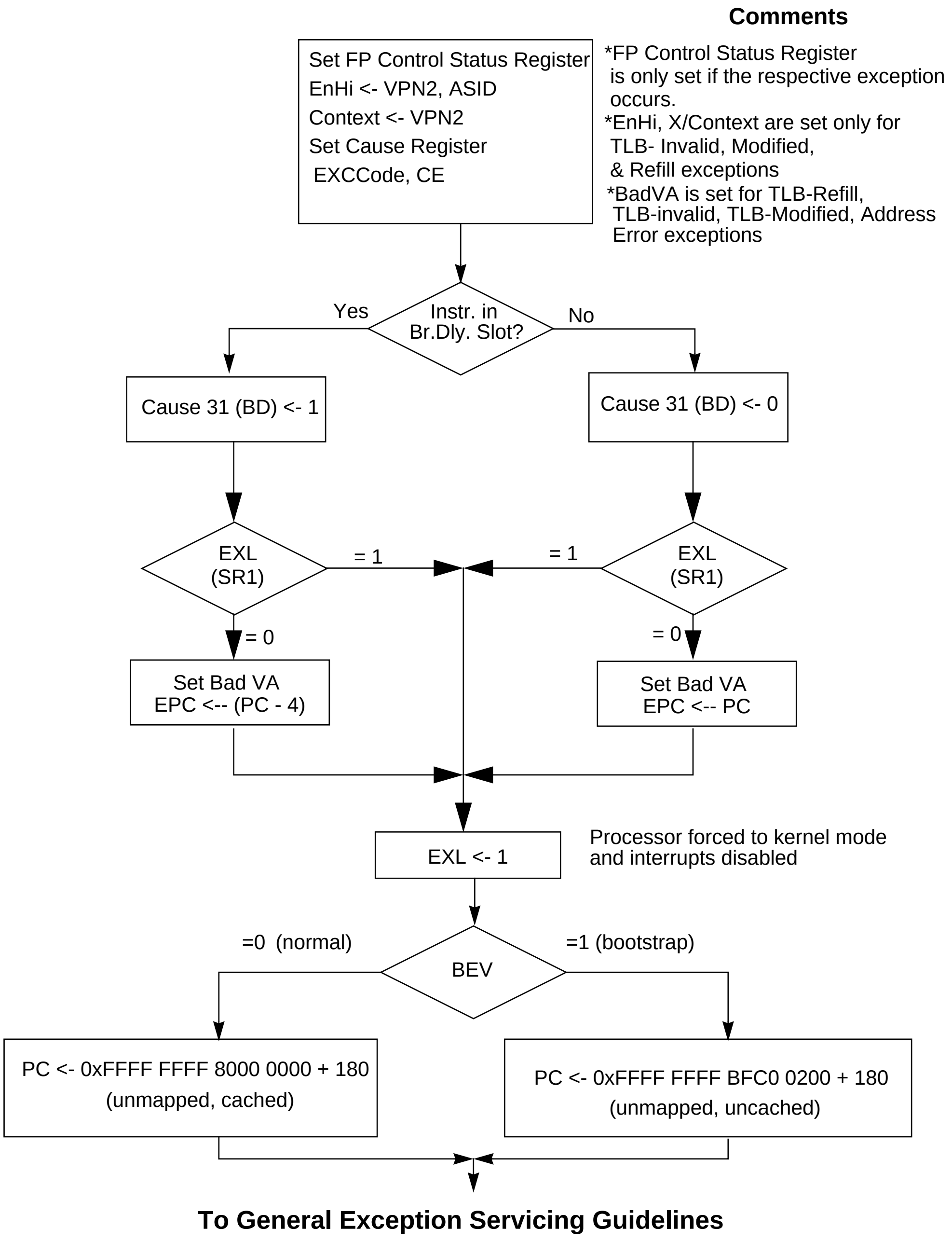

Comments
Set FP Control Status Register
EnHi <- VPN2, ASID
Context <- VPN2
Set Cause Register
EXCCode, CE
*FP Control Status Register is only set if the respective exception occurs.
*EnHi, X/Context are set only for TLB- Invalid, Modified, & Refill exceptions
*BadVA is set for TLB-Refill, TLB-invalid, TLB-Modified, Address Error exceptions
Yes
Instr. in Br.Dly. Slot?
No
Cause 31 (BD) <- 1
Cause 31 (BD) <- 0
EXL (SR1)
= 1
= 1
EXL (SR1)
= 0
= 0
Set Bad VA
EPC <-- (PC - 4)
Set Bad VA
EPC <-- PC
EXL <- 1
Processor forced to kernel mode and interrupts disabled
=0 (normal)
BEV
=1 (bootstrap)
PC <- 0xFFFF FFFF 8000 0000 + 180
(unmapped, cached)
PC <- 0xFFFF FFFF BFC0 0200 + 180
(unmapped, uncached)
To General Exception Servicing Guidelines

Figure 7-17 General Exception Handler (HW)

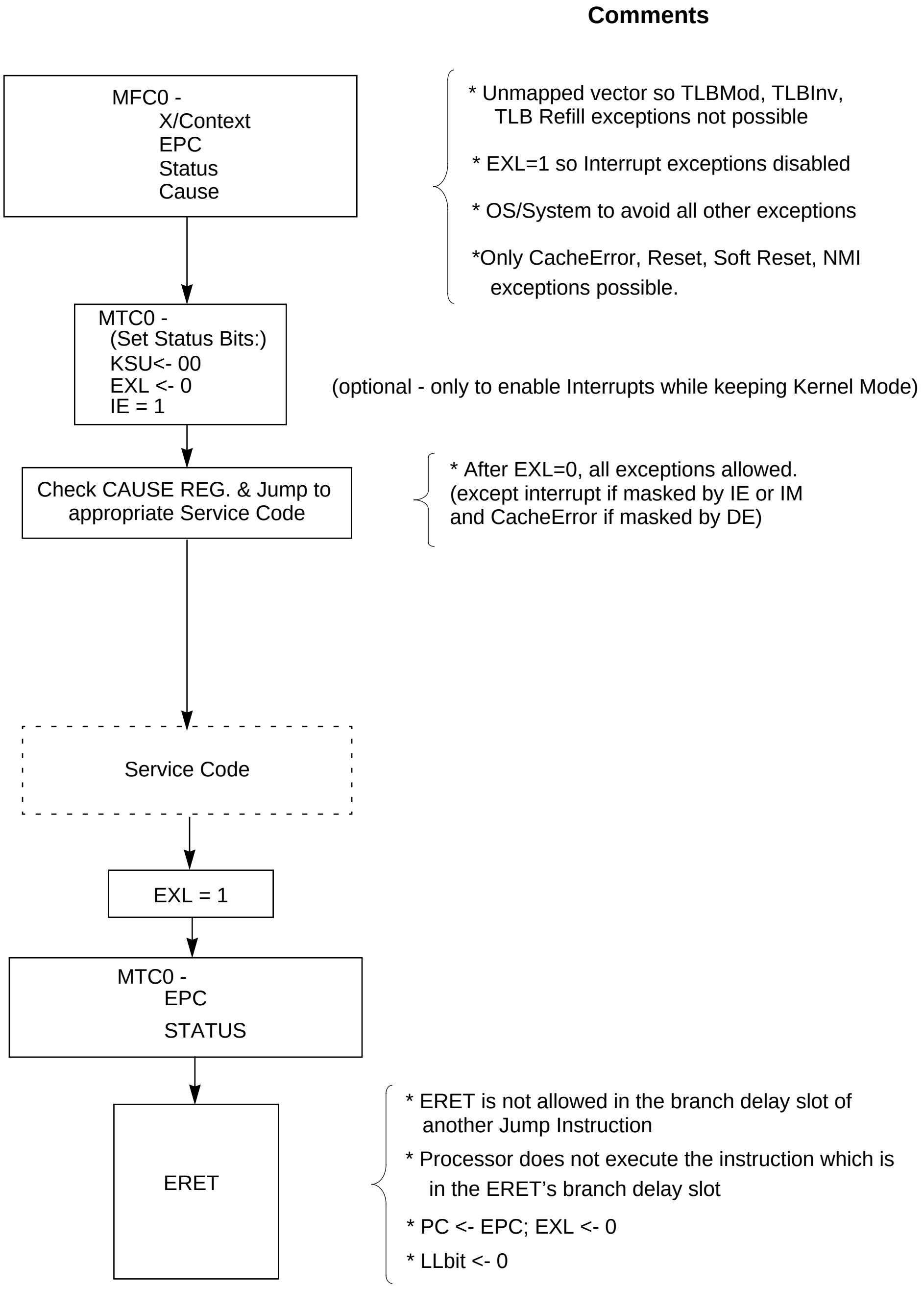

Comments
MFC0 -
X/Context
EPC
Status
Cause
* Unmapped vector so TLBMod, TLBInv, TLB Refill exceptions not possible
* EXL=1 so Interrupt exceptions disabled
* OS/System to avoid all other exceptions
*Only CacheError, Reset, Soft Reset, NMI exceptions possible.
MTC0 -
(Set Status Bits:)
KSU<- 00
EXL <- 0
IE = 1
(optional - only to enable Interrupts while keeping Kernel Mode)
Check CAUSE REG. & Jump to appropriate Service Code
* After EXL=0, all exceptions allowed.
(except interrupt if masked by IE or IM
and CacheError if masked by DE)
Service Code
EXL = 1
MTC0 -
EPC
STATUS
ERET
* ERET is not allowed in the branch delay slot of another Jump Instruction
* Processor does not execute the instruction which is in the ERET's branch delay slot
* PC <- EPC; EXL <- 0
* LLbit <- 0

Figure 7-18   General Exception Servicing Guidelines (SW)

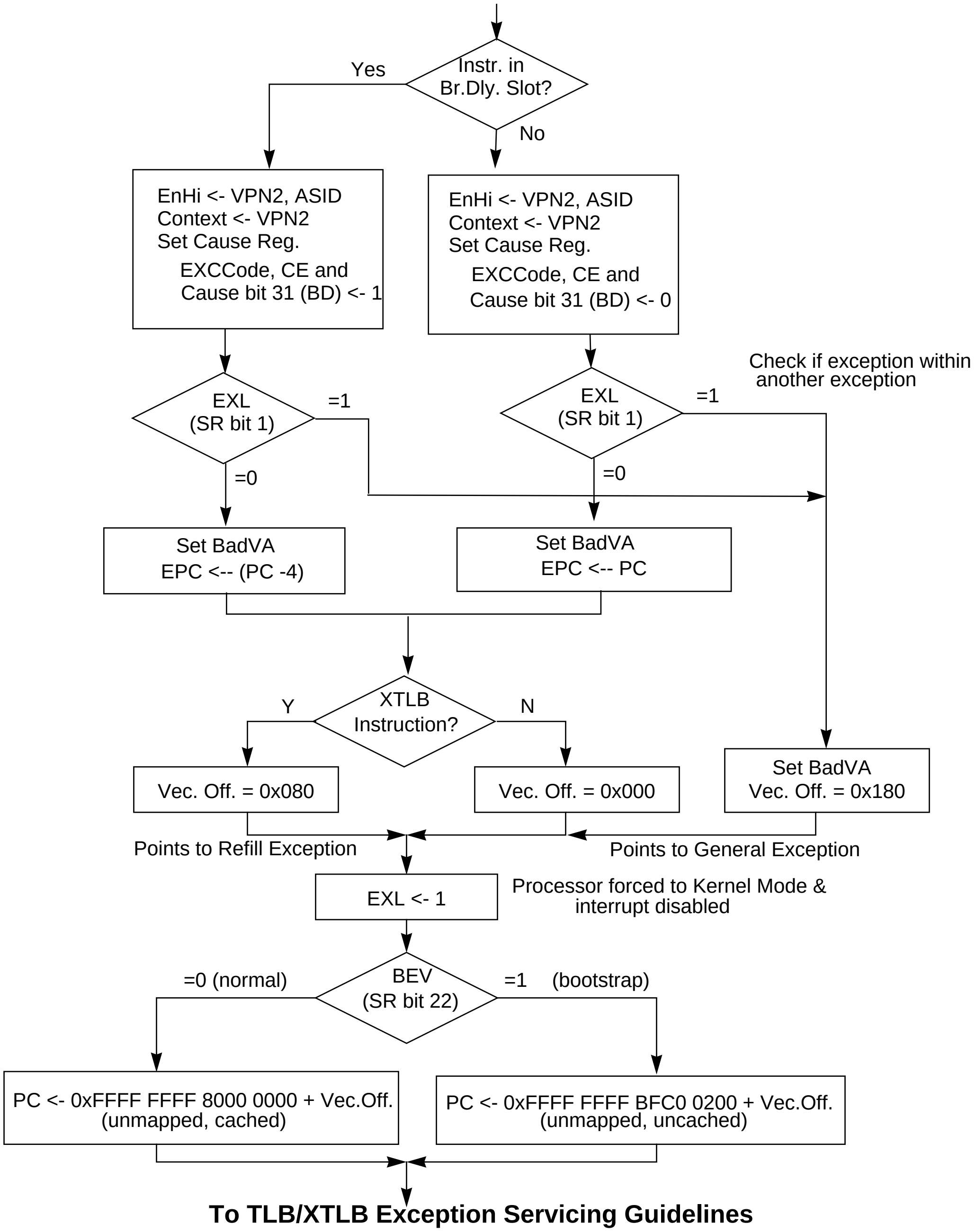
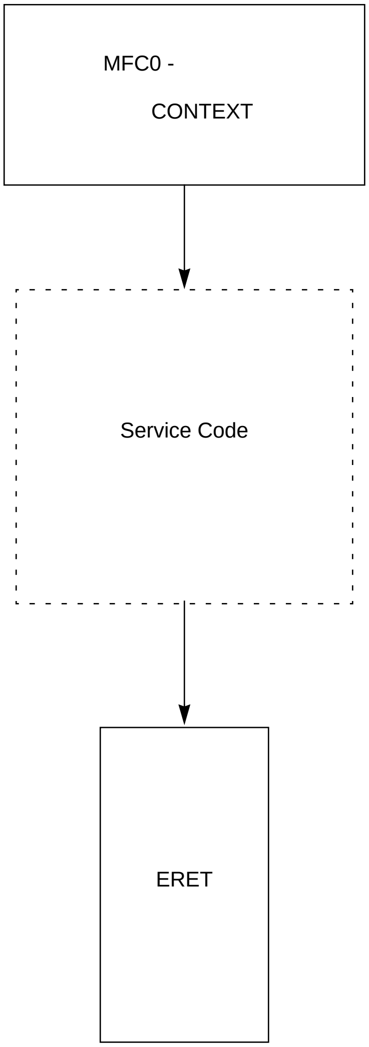


Figure 7-19   TLB/XTLB Miss Exception Handler (HW)



Comments

* Unmapped vector so TLBMod, TLBInv, TLB Refill or VCEP exceptions not possible

* EXL=1 so Interrupt exceptions disabled

* OS/System to avoid all other exceptions

*Only CacheError, Reset, Soft Reset, NMI exceptions possible.

* Load the mapping of the virtual address in Context Reg. Move it to ENLO and Write into the TLB

* There could be a TLB miss again during the mapping of the data or instruction address. The processor will jump to the general exception vector since the EXL is 1. (Option to complete the first level refill in the general exception handler or ERET to the original instruction and take the exception again)

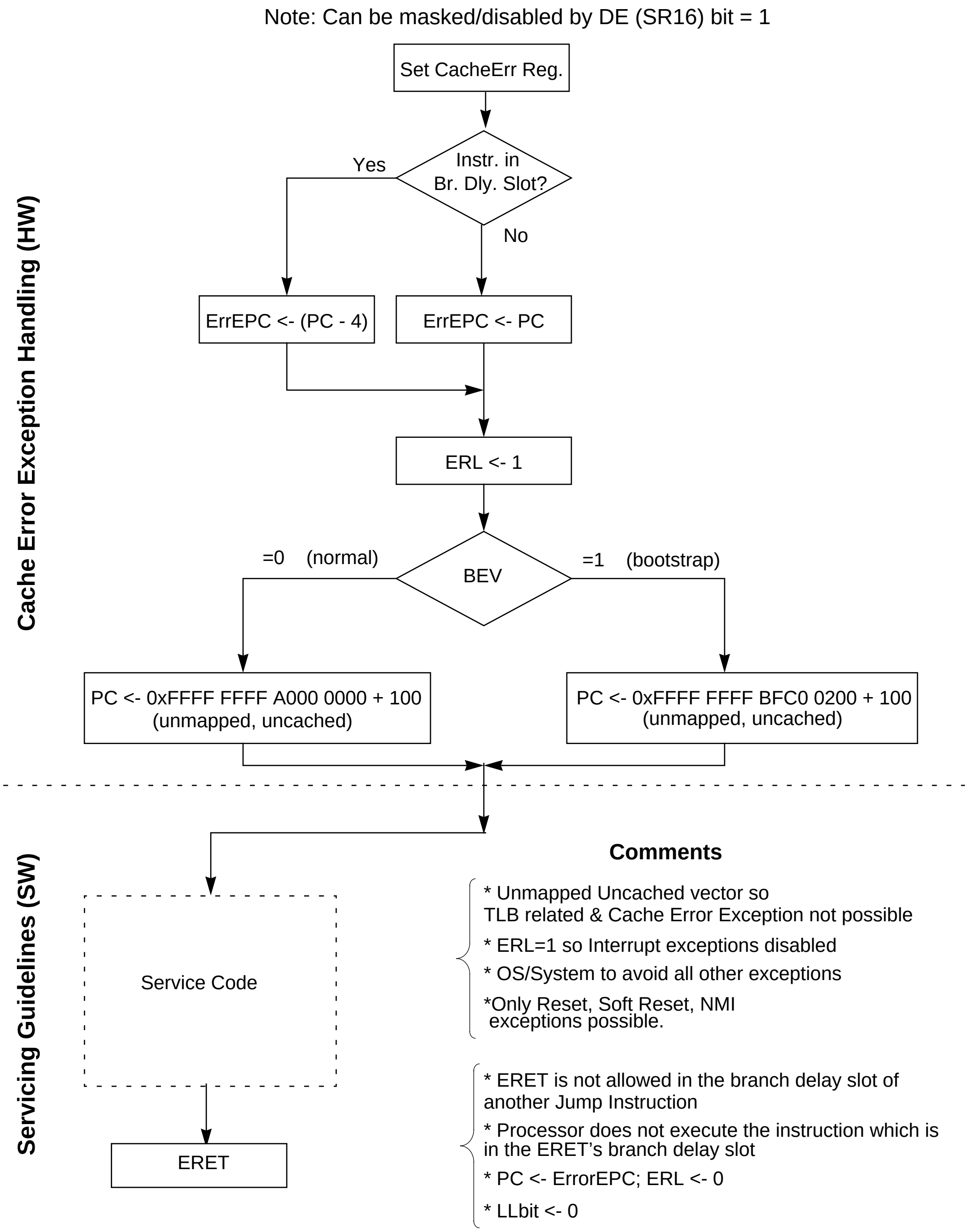
* ERET is not allowed in the branch delay slot of another Jump Instruction

* Processor does not execute the instruction which is in the ERET’s branch delay slot

* PC <- EPC; EXL <- 0

* LLbit <- 0

Figure 7-20   TLB/XTLB Exception Servicing Guidelines (SW)



Note: Can be masked/disabled by DE (SR16) bit = 1

Cache Error Exception Handling (HW)
Set CacheErr Reg.
Instr. in Br. Dly. Slot?
Yes
No
ErrEPC <- (PC - 4)
ErrEPC <- PC
ERL <- 1
BEV
=0 (normal)
=1 (bootstrap)
PC <- 0xFFFF FFFF A000 0000 + 100 (unmapped, uncached)
PC <- 0xFFFF FFFF BFC0 0200 + 100 (unmapped, uncached)
Servicing Guidelines (SW)
Service Code
ERET

Comments

* Unmapped Uncached vector so
TLB related & Cache Error Exception not possible

* ERL=1 so Interrupt exceptions disabled

* OS/System to avoid all other exceptions

*Only Reset, Soft Reset, NMI
exceptions possible.

* ERET is not allowed in the branch delay slot of another Jump Instruction

* Processor does not execute the instruction which is in the ERET's branch delay slot

* PC <- ErrorEPC; ERL <- 0

* LLbit <- 0

Figure 7-21 Cache Error Exception Handling (HW) and Servicing Guidelines

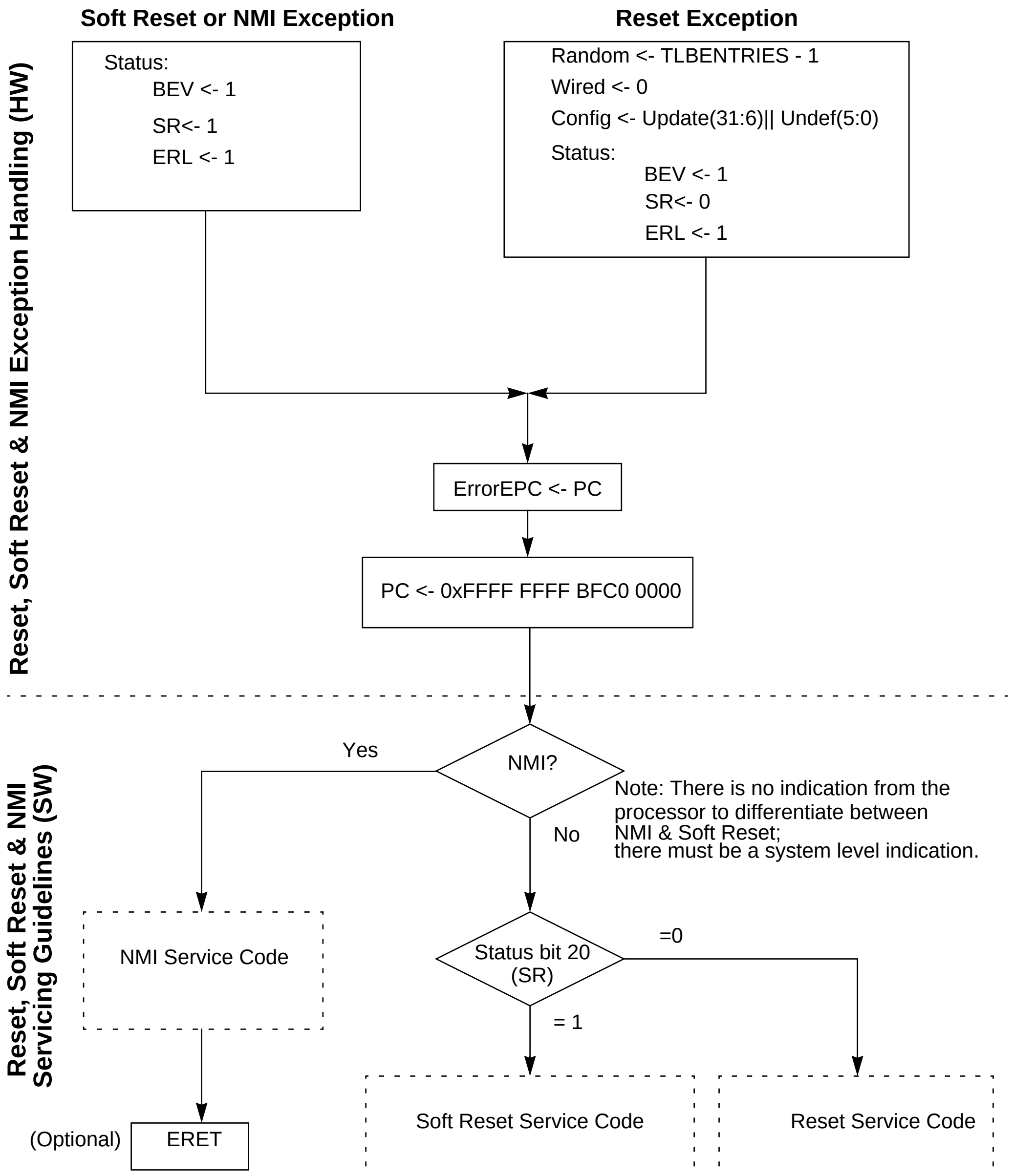

Soft Reset or NMI Exception
Status:
BEV <- 1
SR<- 1
ERL <- 1
Reset Exception
Random <- TLBENTRIES - 1
Wired <- 0
Config <- Update(31:6)|| Undef(5:0)
Status:
BEV <- 1
SR<- 0
ERL <- 1
ErrorEPC <- PC
PC <- 0xFFFF FFFF BFC0 0000
Reset, Soft Reset & NMI Exception Handling (HW)
Yes
NMI?
No
Note: There is no indication from the processor to differentiate between NMI & Soft Reset; there must be a system level indication.
NMI Service Code
Status bit 20 (SR)
=0
= 1
(Optional)
ERET
Soft Reset Service Code
Reset Service Code
Reset, Soft Reset & NMI Servicing Guidelines (SW)

Figure 7-22   Reset, Soft Reset & NMI Exception Handling

Chapter 8 Floating Point Unit

This chapter describes the floating-point unit (FPU) of the V_R5000 processor, including the programming model, instruction set and formats, and the pipeline.

The FPU, with associated system software, fully conforms to the requirements of ANSI/IEEE Standard 754–1985, *IEEE Standard for Binary Floating-Point Arithmetic*. In addition, the MIPS architecture fully supports the recommendations of the standard and precise exceptions.

8.1 Overview

The FPU operates as a coprocessor for the CPU (it is assigned coprocessor label *CP1*), and extends the CPU instruction set to perform arithmetic operations on floating-point values.

Figure 8-1 illustrates the functional organization of the FPU.

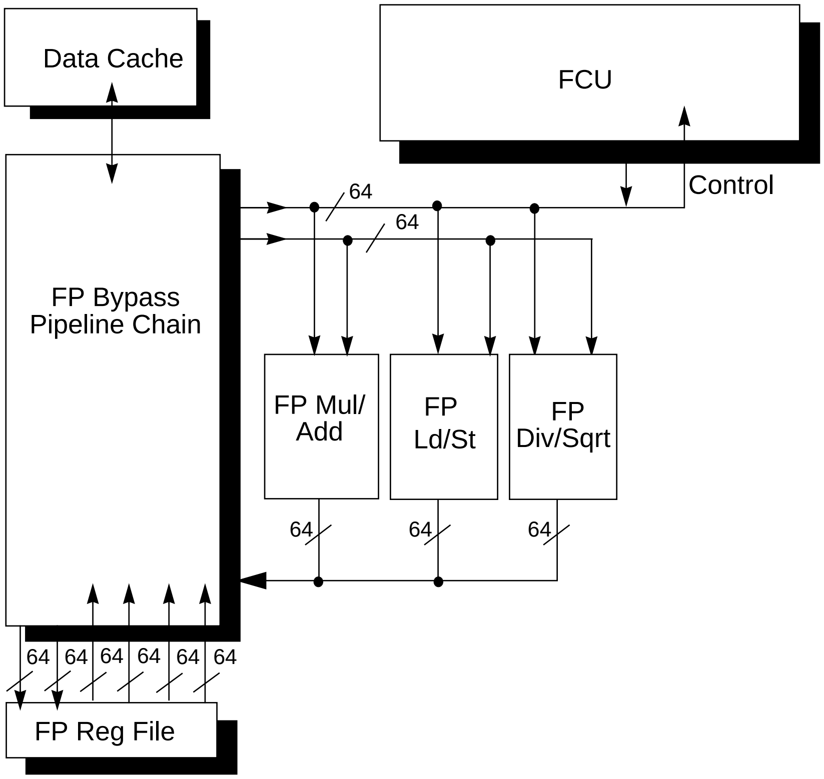


Figure 8-1 FPU Functional Block Diagram

8.2 FPU Features

This section briefly describes the operating model, the load/store instruction set, and the coprocessor interface in the FPU. A more detailed description is given in the sections that follow.

- **Full 64-bit Operation**. When the *FR* bit in the CPU *Status* register equals 0, the FPU is in 32-bit mode and contains thirty-two 32-bit registers that hold single- or, when used in pairs, double-precision values. When the *FR* bit in the CPU *Status* register equals 1, the FPU is in 64-bit mode and the registers are expanded to 64 bits wide. Each register can hold single- or double-precision values. The FPU also includes a 32-bit *Control/Status* register that provides access to all IEEE-Standard exception handling capabilities.
- **Load and Store Instruction Set**. Like the CPU, the FPU uses a load- and store-oriented instruction set, with single-cycle load and store operations.
- **Tightly Coupled Coprocessor Interface**. The FPU resides on-chip to form a tightly coupled unit with a seamless integration of floating-point and fixed-point instruction sets. Since each unit receives and executes instructions in parallel, some floating-point instructions can execute at the same single-cycle-per-instruction rate as fixed-point instructions.

8.3 FPU Programming Model

This section describes the set of FPU registers and their data organization. The FPU registers include *Floating-Point General Purpose* registers *(FGR*s) and two control registers: *Control/Status* and *Implementation/Revision*.

8.4 Floating-Point General Registers (FGRs)

The FPU has a set of *Floating-Point General Purpose* registers *(FGR*s) that can be accessed in the following ways:

- As 32 general purpose registers (32 FGRs), each of which is 32 bits wide when the *FR* bit in the CPU *Status* register equals 0; or as 32 general purpose registers (32 FGRs), each of which is 64-bits wide when *FR* equals 1. The CPU accesses these registers through move, load, and store instructions.
- As 16 floating-point registers (see the next section for a description of FPRs), each of which is 64-bits wide, when the *FR* bit in the CPU *Status* register equals 0. The FPRs hold values in either single- or double-precision floating-point format. Each FPR corresponds to adjacently numbered FGRs as shown in Figure 8-2.
- As 32 floating-point registers (see the next section for a description of FPRs), each of which is 64-bits wide, when the *FR* bit in the CPU *Status* register equals 1. The FPRs hold values in either single- or double-precision floating-point format. Each FPR corresponds to an FGR as shown in Figure 8-2.

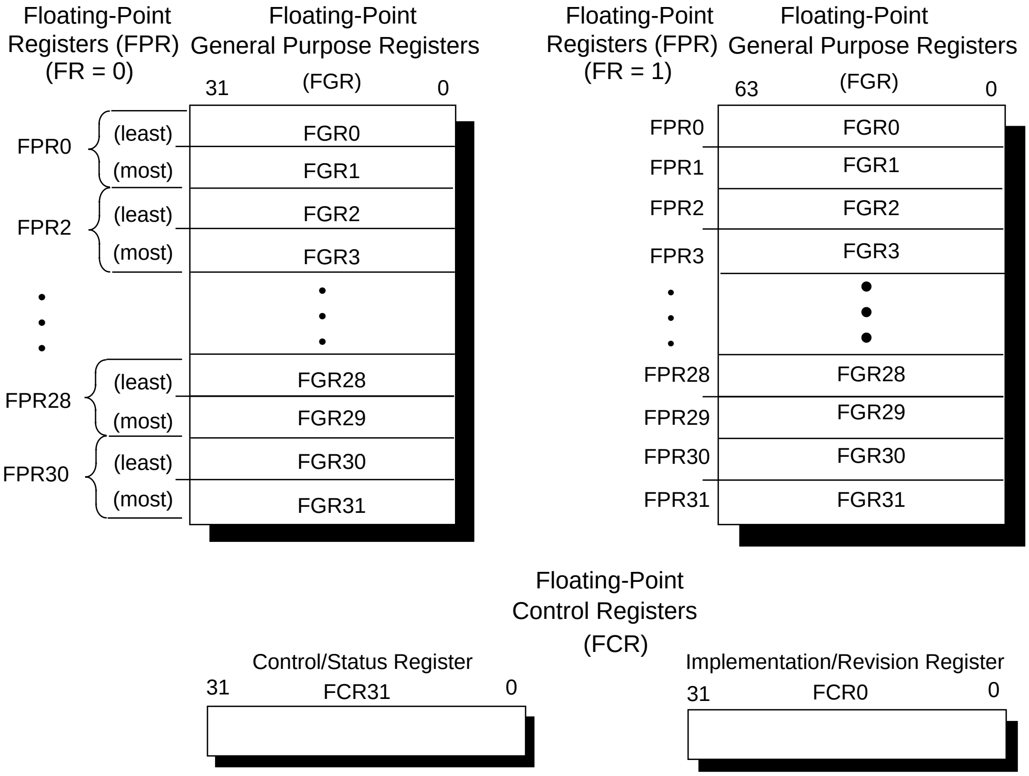


Figure 8-2 FPU Registers

8.5 Floating-Point Registers

The FPU provides:

- 16 *Floating-Point* registers (*FPR*s) when the *FR* bit in the *Status* register equals 0, or
- 32 *Floating-Point* registers (*FPR*s) when the *FR* bit in the *Status* register equals 1.

These 64-bit registers hold floating-point values during floating-point operations and are physically formed from the *General Purpose* registers (*FGRs*). When the *FR* bit in the *Status* register equals 1, the *FPR* references a single 64-bit *FGR.*

The *FPR*s hold values in either single- or double-precision floating-point format. If the *FR* bit equals 0, only even numbers (the *least* register, as shown in Figure 8-2) can be used to address *FPR*s. When the *FR* bit is set to a 1, all *FPR* register numbers are valid.

If the *FR* bit equals 0 during a double-precision floating-point operation, the general registers are accessed in double pairs. Thus, in a double-precision operation, selecting *Floating-Point Register 0* (*FPR0*) actually addresses adjacent *Floating-Point General Purpose* registers *FGR0* and *FGR1*.

8.6 Floating-Point Control Registers

The FPU has 32 control registers (*FCR*s) that can only be accessed by move operations. The *FCR*s are described below:

- The *Implementation/Revision* register *(FCR0)* holds revision information about the FPU.
- The *Control/Status* register *(FCR31)* controls and monitors exceptions, holds the result of compare operations, and establishes rounding modes.
- *FCR1* to *FCR30* are reserved.

Table 8-1 lists the assignments of the *FCR*s.

Table 8-1 Floating-Point Control Register Assignments

FCR Number	Use
FCR0	Coprocessor implementation and revision register
FCR1 to FCR30	Reserved
FCR31	Rounding mode, cause, trap enables, and flags

8.6.1 Implementation and Revision Register (FCR0)

The read-only *Implementation and Revision* register (*FCR0*) specifies the implementation and revision number of the FPU. This information can determine the coprocessor revision and performance level, and can also be used by diagnostic software.

Figure 8-3 shows the layout of the register; Table 8-2 describes the *Implementation and Revision* register (*FCR0*) fields.

Implementation/Revision Register (FCR0)

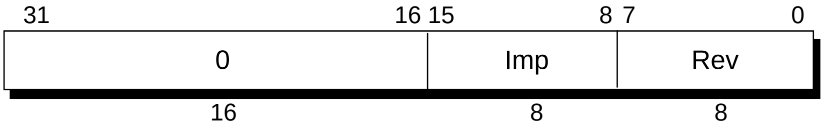


Figure 8-3 Implementation/Revision Register

Table 8-2 FCR0 Fields

Field	Description
Imp	Implementation number (0x23)
Rev	Revision number in the form of *y.x*
0	Reserved. Must be written as zeroes, and returns zeroes when read.

The revision number is a value of the form *y.x,* where:

- *y* is a major revision number held in bits 7:4.
- *x* is a minor revision number held in bits 3:0.

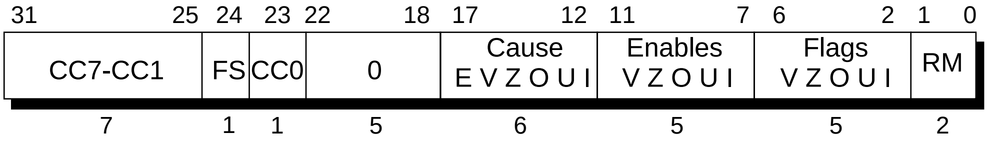
The revision number distinguishes some chip revisions; however, MIPS does not guarantee that changes to its chips are necessarily reflected by the revision number, or that changes to the revision number necessarily reflect real chip changes. For this reason revision number values are not listed, and software should not rely on the revision number to characterize the chip.

8.6.2 Control/Status Register (FCR31)

The *Control*/*Status* register *(FCR31*) contains control and status information that can be accessed by instructions in either Kernel or User mode. *FCR31* also controls the arithmetic rounding mode and enables User mode traps, as well as identifying any exceptions that may have occurred in the most recently executed instruction, along with any exceptions that may have occurred without being trapped.

Figure 8-4 shows the format of the *Control/Status* register, and Table 8-3 describes the *Control/Status* register fields. Figure 8-5 shows the *Control/Status* register *Cause, Flag,* and *Enable* fields.

Control/Status Register (FCR31)



31 25	24	23	22 18	17 12	11 7	6 2	1 0
CC7-CC1	FS	CC0	0	Cause E V Z O U I	Enables V Z O U I	Flags V Z O U I	RM
7	1	1	5	6	5	5	2

Legend:
E = Unimplemented Operation
V = Invalid Operation
Z = Division by zero
O = Overflow
U = Underflow
I = Inexact Operation

Figure 8-4 FP Control/Status Register Bit Assignments

Table 8-3 Control/Status Register Fields

Field	Description
CC7-CC1	Condition bits 7-1. See description of *Control/Status* register *Condition* bit.
FS	The *FS* bit enables a value that cannot be normalized (denormarlized number) to be flushed. When the *FS* bit is set and the enable bit is not set for the underflow exception and illegal exception, the result of the denormalized number does not cause the unimplemented operation exception, but is flushed. Whether the flushed result is 0 or the minimum normalized value is determined depending on the rounding mode (refer to **Table 8-4**). On the V_R5000, even if the FS bit is set, if a madd, msub, nmadd or nmsub instruction encounters a denormalized result during the multiply portion of the calculation, an unimplemented operation exception is always taken.
CC0	Condition bit 0. See description of *Control/Status* register *Condition* bit.
Cause	Cause bits. See description of *Control/Status* register *Cause, Flag,* and *Enable* bits.
Enables	Enable bits. See description of *Control/Status* register *Cause, Flag,* and *Enable* bits.
Flags	Flag bits. See description of *Control/Status* register *Cause, Flag,* and *Enable* bits.
RM	Rounding mode bits. See description of *Control/Status* register *Rounding Mode Control* bits.

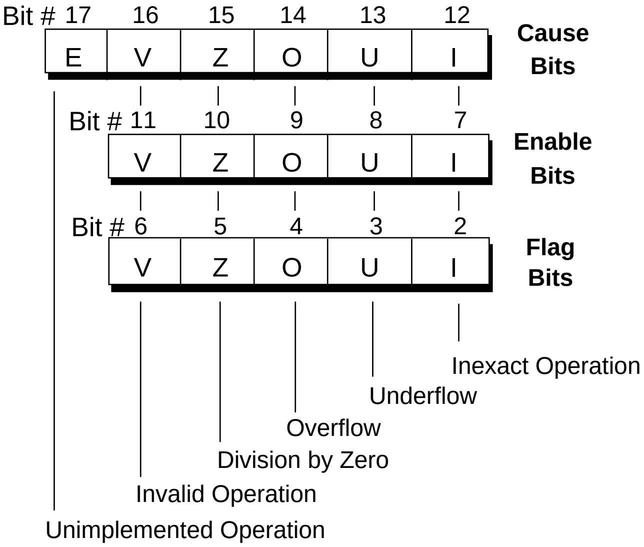


Figure 8-5   Control/Status Register Cause, Flag, and Enable Fields

(1) Accessing the Control/Status Register

When the *Control*/*Status* register is read by a Move Control From Coprocessor 1 (CFC1) instruction, all unfinished instructions in the pipeline are completed before the contents of the register are moved to the main processor. If a floating-point exception occurs as the pipeline empties, the FP exception is taken and the CFC1 instruction is re-executed after the exception is serviced.

The bits in the *Control/Status* register can be set or cleared by writing to the register using a Move Control To Coprocessor 1 (CTC1) instruction. *FCR31* must only be written to when the FPU is not actively executing floating-point operations; this can be ensured by reading the contents of the register to empty the pipeline.

(2) IEEE Standard 754

IEEE Standard 754 specifies that floating-point operations detect certain exceptional cases, raise flags, and can invoke an exception handler when an exception occurs. These features are implemented in the MIPS architecture with the *Cause*, *Enable*, and *Flag* fields of the *Control/Status* register. The *Flag* bits implement IEEE 754 exception status flags, and the *Cause* and *Enable* bits implement exception handling.

(3) Control/Status Register FS Bit

The *FS* bit enables a value that cannot be normalized (denormarlized number) to be flushed. When the *FS* bit is set and the enable bit is not set for the underflow exception and illegal exception, the result of the denormalized number does not cause the unimplemented operation exception, but is flushed. Whether the flushed result is 0 or the minimum normalized value is determined depending on the rounding mode (refer to **Table 8-4**).

However, for MADD.fmt, NMADD.fmt, MSUB.fmt, and NMSUB.fmt instructions, the V_R5000 will always take an unimplemented operation exception if the intermediate multiply result is a denormalized value regardless of the value of the *FS* bit.

Table 8-4 Flush Values of Denormalized Number Results

Denormalized Number Result	Flushed Result Rounding Mode			
	RN	RZ	RP	RM
Positive	+0	+0	$+2^{Emin}$	+0
Negative	–0	–0	–0	-2^{Emin}

(4) Control/Status Register Condition Bit

When a floating-point Compare operation takes place, the result is stored at bit 23 and bits 31:25, the *Condition* bits, to save or restore the state of the condition line. The *CC* bit is set to 1 if the condition is true; the bit is cleared to 0 if the condition is false. Bit 23 and bits 31:25 are affected only by compare and Move Control To FPU instructions.

(5) Control/Status Register Cause, Flag, and Enable Fields

Figure 8-5 illustrates the *Cause*, *Flag*, and *Enable* fields of the *Control/Status* register.

Cause Bits

Bits 17:12 in the *Control/Status* register contain *Cause* bits, as shown in Figure 8-5, which reflect the results of the most recently executed instruction. The *Cause* bits are a logical extension of the CP0 *Cause* register; they identify the exceptions raised by the last floating-point operation and raise an interrupt or exception if the corresponding enable bit is set. If more than one exception occurs on a single instruction, each appropriate bit is set.

The *Cause* bits are written by each floating-point operation (but not by load, store, or move operations). The Unimplemented Operation (*E*) bit is set to a 1 if software emulation is required, otherwise it remains 0. The other bits are set to 0 or 1 to indicate the occurrence or non-occurrence (respectively) of an IEEE 754 exception.

When a floating-point exception is taken, no results are stored, and the only state affected is the *Cause* bit.

Enable Bits

A floating-point exception is generated any time a *Cause* bit and the corresponding *Enable* bit are set. A floating-point operation that sets an enabled *Cause* bit forces an immediate exception, as does setting both *Cause* and *Enable* bits with CTC1.

There is no enable for Unimplemented Operation (*E*). Setting Unimplemented Operation always generates a floating-point exception.

Before returning from a floating-point exception, software must first clear the enabled *Cause* bits with a CTC1 instruction to prevent a repeat of the interrupt. Thus, User mode programs can never observe enabled *Cause* bits set; if this information is required in a User mode handler, it must be passed somewhere other than the *Status* register.

For a floating-point operation that sets only unenabled *Cause* bits, no exception occurs and the default result defined by IEEE 754 is stored. In this case, the exceptions that were caused by the immediately previous floating-point operation can be determined by reading the *Cause* field.

Flag Bits

The *Flag* bits are cumulative and indicate that an exception was raised by an operation that was executed since they were explicitly reset. *Flag* bits are set to 1 if an IEEE 754 exception is raised, otherwise they remain unchanged. The *Flag* bits are never cleared as a side effect of floating-point operations; however, they can be set or cleared by writing a new value into the *Status* register, using a Move To Coprocessor Control instruction.

When a floating-point exception is taken, the flag bits are not set by the hardware; floating-point exception software is responsible for setting these bits before invoking a user handler.

(6) Control/Status Register Rounding Mode Control Bits

Bits 1 and 0 in the *Control/Status* register constitute the *Rounding Mode* (*RM*) field.

As shown in Table 8-5 these bits specify the rounding mode that the FPU uses for all floating-point operations.

Table 8-5 Rounding Mode Bit Decoding

Rounding Mode RM(1:0)	**Mnemonic**	**Description**
0	RN	Round result to nearest representable value; round to value with least-significant bit 0 when the two nearest representable values are equally near.
1	RZ	Round toward 0: round to value closest to and not greater in magnitude than the infinitely precise result.
2	RP	Round toward +∞: round to value closest to and not less than the infinitely precise result.
3	RM	Round toward – ∞: round to value closest to and not greater than the infinitely precise result.

8.7 Floating-Point Formats

The FPU performs both 32-bit (single-precision) and 64-bit (double-precision) IEEE standard floating-point operations. The 32-bit single-precision format has a 24-bit signed-magnitude fraction field (*f*+*s*) and an 8-bit exponent (*e*), as shown in Figure 8-6.

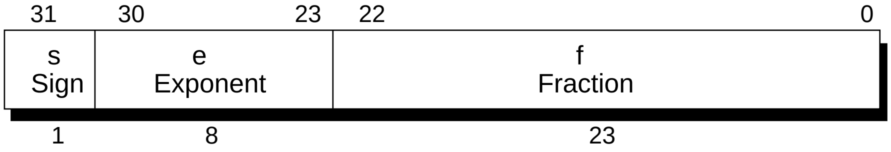


Figure 8-6 Single-Precision Floating-Point Format

The 64-bit double-precision format has a 53-bit signed-magnitude fraction field (*f*+*s)* and an 11-bit exponent, as shown in Figure 8-7.

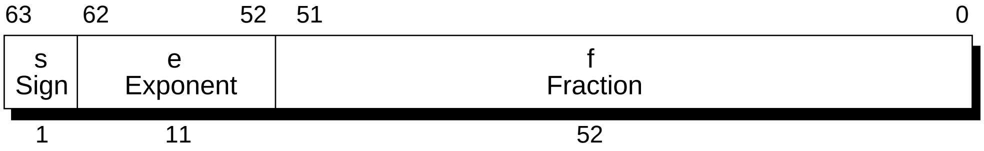


Figure 8-7 Double-Precision Floating-Point Format

As shown in the above figures, numbers in floating-point format are composed of three fields:

- sign field, s
- biased exponent, $e = E + bias$
- fraction, $f = .b_1b_2....b_{p-1}$

The range of the unbiased exponent E includes every integer between the two values E_{min} and E_{max} inclusive, together with two other reserved values:

- E_{min} -1 (to encode ±0 and denormalized numbers)
- E_{max} +1 (to encode $\pm^{\infty}$ and NaNs [Not a Number])

For single- and double-precision formats, each representable nonzero numerical value has just one encoding.

For single- and double-precision formats, the value of a number, v, is determined by the equations shown in Table 8-6.

Table 8-6 Calculating Values in Single and Double-Precision Formats

No.	Equation
(1)	if $E = E_{max}+1$ and $f \neq 0$, then v is NaN, regardless of s
(2)	if $E = E_{max}+1$ and $f = 0$, then $v = (-1)^{s}\ \infty$
(3)	if $E_{min} \leq E \leq E_{max}$, then $v = (-1)^{s}2^{E}(1.f)$
(4)	if $E = E_{min}-1$ and $f \neq 0$, then $v = (-1)^{s}2^{Emin}(0.f)$
(5)	if $E = E_{min}-1$ and $f = 0$, then $v = (-1)^{s}0$

For all floating-point formats, if v is NaN, the most-significant bit of f determines whether the value is a signaling or quiet NaN: v is a signaling NaN if the most-significant bit of f is set, otherwise, v is a quiet NaN.

Table 8-7 defines the values for the format parameters; minimum and maximum floating-point values are given in Table 8-8.

Table 8-7   Floating-Point Format Parameter Values

Parameter	Format	
	Single	**Double**
E_{max}	+127	+1023
E_{min}	–126	–1022
Exponent *bias*	+127	+1023
Exponent width in bits	8	11
Integer bit	hidden	hidden
f (Fraction width in bits)	24	53
Format width in bits	32	64

Table 8-8   Minimum and Maximum Floating-Point Values

Type	Value
Float Minimum	1.40129846e–45
Float Minimum Norm	1.17549435e–38
Float Maximum	3.40282347e+38
Double Minimum	4.9406564584124654e–324
Double Minimum Norm	2.2250738585072014e–308
Double Maximum	1.7976931348623157e+308

8.8 Binary Fixed-Point Format

Binary fixed-point values are held in 2’s complement format. Unsigned fixed-point values are not directly provided by the floating-point instruction set. Figure 8-8 illustrates binary fixed-point format; Table 8-9 lists the binary fixed-point format fields.

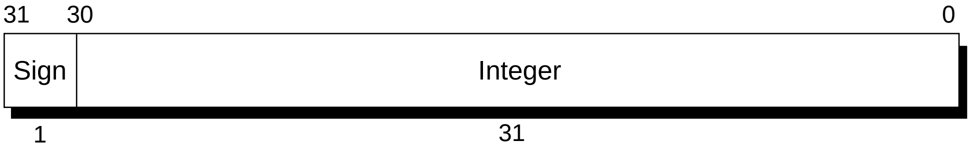


Figure 8-8 Binary Fixed-Point Format

Field assignments of the binary fixed-point format are:

Table 8-9 Binary Fixed-Point Format Fields

Field	Description
sign	sign bit
integer	integer value

8.9 Floating-Point Instruction Set Overview

All FPU instructions are 32 bits long, aligned on a word boundary. They can be divided into the following groups:

- **Load, Store, and Move** instructions move data between memory, the main processor, and the *FPU General Purpose* registers.
- **Conversion** instructions perform conversion operations between the various data formats.
- **Computational** instructions perform arithmetic operations on floating-point values in the FPU registers.
- **Compare** instructions perform comparisons of the contents of registers and set a conditional bit based on the results.
- **Branch on FPU Condition** instructions perform a branch to the specified target if the specified coprocessor condition is met.

In the instruction formats shown in Table 8-10 through Table 8-13, the *fmt* appended to the instruction opcode specifies the data format: *S* specifies single-precision binary floating-point, *D* specifies double-precision binary floating-point, *W* specifies 32-bit binary fixed-point, and *L* specifies 64-bit (long) binary fixed-point.

Table 8-10   FPU Instruction Summary: Load, Move and Store Instructions

OpCode	Description
LWC1	Load Word to FPU
LWXC1	Load Word Indexed to FPU
SWC1	Store Word from FPU
SWXC1	Store Word Indexed from FPU
LDC1	Load Doubleword to FPU
LDXC1	Load Doubleword Indexed to FPU
SDC1	Store Doubleword From FPU
SDXC1	Store Doubleword Indexed From FPU
MTC1	Move Word To FPU
MFC1	Move Word From FPU
CTC1	Move Control Word To FPU
CFC1	Move Control Word From FPU
DMTC1	Doubleword Move To FPU
DMFC1	Doubleword Move From FPU
PREFX	Prefetch Indexed - Register + Register

Table 8-11   FPU Instruction Summary: Conversion Instructions

OpCode	Description
CVT.S.fmt	Floating-point Convert to Single FP
CVT.D.fmt	Floating-point Convert to Double FP
CVT.W.fmt	Floating-point Convert to 32-bit Fixed Point
CVT.L.fmt	Floating-point Convert to 64-bit Fixed Point
ROUND.W.fmt	Floating-point Round to 32-bit Fixed Point
ROUND.L.fmt	Floating-point Round to 64-bit Fixed Point
TRUNC.W.fmt	Floating-point Truncate to 32-bit Fixed Point
TRUNC.L.fmt	Floating-point Truncate to 64-bit Fixed Point
CEIL.W.fmt	Floating-point Ceiling to 32-bit Fixed Point
CEIL.L.fmt	Floating-point Ceiling to 64-bit Fixed Point
FLOOR.W.fmt	Floating-point Floor to 32-bit Fixed Point
FLOOR.L.fmt	Floating-point Floor to 64-bit Fixed Point

Table 8-12 FPU Instruction Summary: Computational Instructions

OpCode	Description
ADD.fmt	Floating-point Add
SUB.fmt	Floating-point Subtract
MADD	Floating-point Multiply-Add
MSUB	Floating-point Multiply-Subtract
NMADD	Floating-point Negative Multiply-Add
NMSUB	Floating-point Negative Multiply-Subtract
MUL.fmt	Floating-point Multiply
DIV.fmt	Floating-point Divide
ABS.fmt	Floating-point Absolute Value
MOV.fmt	Floating-point Move
NEG.fmt	Floating-point Negate
SQRT.fmt	Floating-point Square Root
RECIP	Floating-point Reciprocal
RSQRT	Floating-point Reciprocal Square Root

Table 8-13 FPU Instruction Summary: Compare and Branch Instructions

OpCode	Description
C.cond.fmt	Floating-point Compare
BC1T	Branch on FPU True
BC1F	Branch on FPU False
BC1TL	Branch on FPU True Likely
BC1FL	Branch on FPU False Likely

8.9.1 Floating-Point Load, Store, and Move Instructions

This section discusses the manner in which the FPU uses the load, store and move instructions listed in Table 8-10.

(1) Transfers Between FPU and Memory

All data movement between the FPU and memory is accomplished by using one of the following instructions:

- Load Word To Coprocessor 1 (LWC1) or Store Word From Coprocessor 1 (SWC1) instructions, which reference a single 32-bit word of the FPU general registers
- Load Doubleword (LDC1) or Store Doubleword (SDC1) instructions, which reference a 64-bit doubleword.

These load and store operations are unformatted; no format conversions are performed and therefore no floating-point exceptions can occur due to these operations.

(2) Transfers Between FPU and CPU

Data can also be moved directly between the FPU and the CPU by using one of the following instructions:

- Move To Coprocessor 1 (MTC1)
- Move From Coprocessor 1 (MFC1)
- Doubleword Move To Coprocessor 1 (DMTC1)
- Doubleword Move From Coprocessor 1 (DMFC1)

Like the floating-point load and store operations, these operations perform no format conversions and never cause floating-point exceptions.

(3) Load Delay and Hardware Interlocks

The instruction immediately following a load can use the contents of the loaded register. In such cases the hardware interlocks, requiring additional real cycles; for this reason, scheduling load delay slots is desirable, although it is not required for functional code.

(4) Data Alignment

All coprocessor loads and stores reference the following aligned data items:

- For word loads and stores, the access type is always WORD, and the low-order 2 bits of the address must always be 0.
- For doubleword loads and stores, the access type is always DOUBLEWORD, and the low-order 3 bits of the address must always be 0.

(5) Endianness

Regardless of byte-numbering order (endianness) of the data, the address specifies the byte that has the smallest byte address in the addressed field. For a big-endian system, it is the leftmost byte; for a little-endian system, it is the rightmost byte.

8.9.2 Floating-Point Conversion Instructions

Conversion instructions perform conversions between the various data formats such as single- or double-precision, fixed- or floating-point formats.

8.9.3 Floating-Point Computational Instructions

Computational instructions perform arithmetic operations on floating-point values, in registers. There are two categories of computational instructions:

- 3-Operand Register-Type instructions, which perform floating-point addition, subtraction, multiplication, and division
- 2-Operand Register-Type instructions, which perform floating-point absolute value, move, negate, and square root operations

For a detailed description of each instruction, refer to the MIPS IV instruction set manual.

(1) Branch on FPU Condition Instructions

The Branch on FPU (coprocessor unit 1) condition instructions that can test the result of the FPU compare (C.cond) instructions. For a detailed description of each instruction, refer to the MIPS IV instruction set manual.

(2) Floating-Point Compare Operations

The floating-point compare (C.fmt.cond) instructions interpret the contents of two FPU registers (*fs*, *ft*) in the specified format (*fmt*) and arithmetically compare them. A result is determined based on the comparison and conditions (*cond*) specified in the instruction.

Table 8-14 lists the mnemonics for the compare instruction conditions.

Table 8-14 Mnemonics and Definitions of Compare Instruction Conditions

Mnemonic	Definition	Mnemonic	Definition
T	True	**F**	False
OR	Ordered	**UN**	Unordered
NEQ	Not Equal	**EQ**	Equal
OLG	Ordered or Less Than or Greater Than	**UEQ**	Unordered or Equal
UGE	Unordered or Greater Than or Equal	**OLT**	Ordered Less Than
OGE	Ordered Greater Than	**ULT**	Unordered or Less Than
UGT	Unordered or Greater Than	**OLE**	Ordered Less Than or Equal
OGT	Ordered Greater Than	**ULE**	Unordered or Less Than or Equal
ST	Signaling True	**SF**	Signaling False
GLE	Greater Than, or Less Than or Equal	**NGLE**	Not Greater Than or Less Than or Equal
SNE	Signaling Not Equal	**SEQ**	Signaling Equal
GL	Greater Than or Less Than	**NGL**	Not Greater Than or Less Than
NLT	Not Less Than	**LT**	Less Than
GE	Greater Than or Equal	**NGE**	Not Greater Than or Equal
NLE	Not Less Than or Equal	**LE**	Less Than or Equal
GT	Greater Than	**NGT**	Not Greater Than

8.10 FPU Instruction Pipeline Overview

The FPU provides an instruction pipeline that parallels the CPU instruction pipeline. It shares the same five-stage pipeline architecture with the CPU.

8.10.1 Instruction Execution

Figure 8-9 illustrates the 5-instruction overlap in the FPU pipeline.

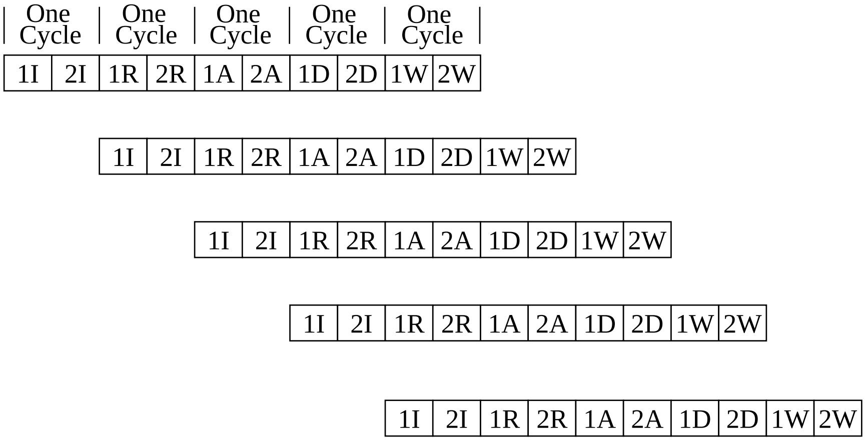

One Cycle
One Cycle
One Cycle
One Cycle
One Cycle
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W

Figure 8-9   FPU Instruction Pipeline

Figure 8-9 assumes that one instruction is completed every PCycle. Most FPU instructions, however, require more than one cycle in the EX stage. This means the FPU must stall the pipeline if an instruction execution cannot proceed because of register or resource conflicts.

8.10.2 Instruction Execution Cycle Time

Unlike the CPU, which executes almost all instructions in a single cycle, more time may be required to execute FPU instructions.

Table 8-15 gives the minimum latency, in processor pipeline cycles, of each floating-point operation for the currently implemented configurations. These latency calculations assume the result of the operation is immediately used in a succeeding operation.

Table 8-15 Floating-Point Operation Latencies

Operation	Pipeline Cycles Latency/Repeat				Operation	Pipeline Cycles Latency/Repeat	
	S	D	W	L		S	D
ADD.fmt	4/1	4/1			BC1T	1/1	
SUB.fmt	4/1	4/1			BC1F	1/1	
MUL.fmt	4/1	5/2			BC1TL	1/1	
DIV.fmt	21/19	36/34			BC1FL	1/1	
SQRT.fmt	21/19	36/34			SWC1, SDC1	2/1	
RECIP	21/19	36/34			LDC1, LWC1	2/1	
RSQRT	38/36	68/66			LWXC1, LDXC1	2/1	
ABS.fmt	1/1	1/1			SWXC1, SDXC1	2/1	
MOV.fmt	1/1	1/1			MTC1, DMTC1	2/1	
NEG.fmt	1/1	1/1			MFC1, DMFC1	2/1	
ROUND.W/ TRUNC.W	4/1	4/1			CTC1	3/3	
ROUND.L/ TRUNC.L	4/1**	4/1**			CFC1	2/2	
CEIL.W/ FLOOR.W	4/1	4/1			MADD	4/1	5/2
CEIL.L/ FLOOR.L	4/1**	4/1**			MSUB	4/1	5/2
CVT.D.fmt	4/1	(a)	4/1	4/1*	NMADD	4/1	5/2
CVT.S.fmt	(a)	4/1	6/3	6/3*	NMSUB	4/1	5/2
CVT.[W,L]	4/1	4/1					
C.cond.fmt	1/1	1/1					

(a).........These operations are illegal.
*..........Trap on greater than 52 bits of significance.
**........Trap on greater than 53 bits of significance.

8.10.3 Instruction Scheduling Constraints

The FPU resource scheduler is kept from issuing instructions to the FPU op units (adder, multiplier, and divider) by the limitations in their micro-architectures. An FPU ALU instruction can be issued at the same time as any other non-FP-ALU instructions. This includes all integer instructions as well as floating-point loads and stores.

Chapter 9 Floating Point Exceptions

This chapter describes FPU floating-point exceptions, including FPU exception types, exception trap processing, exception flags, saving and restoring state when handling an exception, and trap handlers for IEEE Standard 754 exceptions.

A floating-point exception occurs whenever the FPU cannot handle either the operands or the results of a floating-point operation in its normal way.   The FPU responds by generating an exception to initiate a software trap or by setting a status flag.

9.1 Exception Types

The FP *Control/Status* register described in Chapter 8 contains an *Enable* bit for each exception type; exception *Enable* bits determine whether an exception will cause the FPU to initiate a trap or set a status flag.

- If a trap is taken, the FPU remains in the state found at the beginning of the operation and a software exception handling routine executes.
- If no trap is taken, an appropriate value is written into the FPU destination register and execution continues.

The FPU supports the five IEEE Standard 754 exceptions:

- Inexact (I)
- Underflow (U)
- Overflow (O)

- Division by Zero (Z)
- Invalid Operation (V)

Cause bits, *Enables*, and *Flag* bits (status flags) are used.

The FPU adds a sixth exception type, Unimplemented Operation (E), to use when the FPU cannot implement the standard MIPS floating-point architecture, including cases in which the FPU cannot determine the correct exception behavior. This exception indicates the use of a software implementation. The Unimplemented Operation exception has no *Enable* or *Flag* bit; whenever this exception occurs, an unimplemented exception trap is taken (if the FPU interrupt input to the CPU is enabled).

Figure 9-1 illustrates the *Control/Status* register bits that support exceptions.

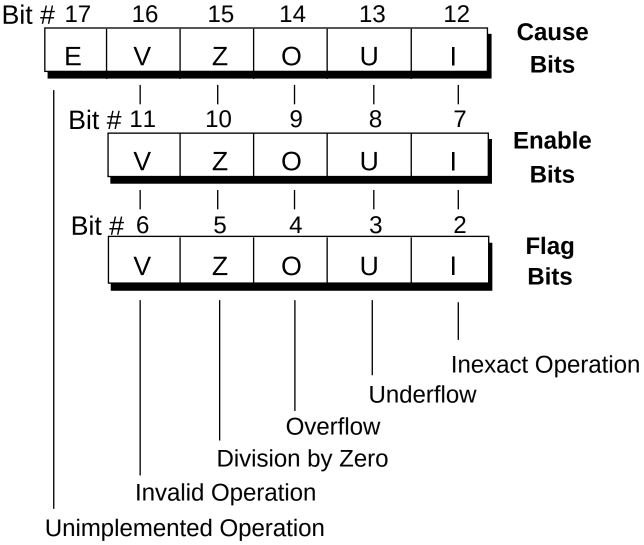


Figure 9-1 Control/Status Register Exception/Flag/Trap/Enable Bits

Each of the five IEEE Standard 754 exceptions (V, Z, O, U, I) is associated with a trap under user control, and is enabled by setting one of the five *Enable* bits. When an exception occurs, the corresponding *Cause* bit is set. If the corresponding *Enable* bit is not set, the *Flag* bit is also set. If the corresponding *Enable* bit is set, the *Flag* bit is not set and the FPU generates an interrupt to the CPU. Subsequent exception processing allows a trap to be taken.

9.2 Exception Trap Processing

When a floating-point exception trap is taken, the *Cause* register indicates the floating-point coprocessor is the cause of the exception trap. The Floating-Point Exception (FPE) code is used, and the *Cause* bits of the floating-point *Control/Status* register indicate the reason for the floating-point exception. These bits are, in effect, an extension of the system coprocessor *Cause* register.

9.3 Flags

A *Flag* bit is provided for each IEEE exception. This *Flag* bit is set to a 1 on the assertion of its corresponding exception, with no corresponding exception trap signaled.

The *Flag* bit is reset by writing a new value into the *Status* register; flags can be saved and restored by software either individually or as a group.

When no exception trap is signaled, floating-point coprocessor takes a default action, providing a substitute value for the exception-causing result of the floating-point operation. The particular default action taken depends upon the type of exception. Table 9-1 lists the default action taken by the FPU for each of the IEEE exceptions.

Table 9-1 Default FPU Exception Actions

Field	Description	Rounding Mode	Default action
I	Inexact exception	Any	Supply a rounded result
U	Underflow exception	RN	Modify underflow values to 0 with the sign of the intermediate result
		RZ	Modify underflow values to 0 with the sign of the intermediate result
		RP	Modify positive underflows to the format's smallest positive finite number; modify negative underflows to -0
		RM	Modify negative underflows to the format's smallest negative finite number; modify positive underflows to 0
O	Overflow exception	RN	Modify overflow values to ∞ with the sign of the intermediate result
		RZ	Modify overflow values to the format's largest finite number with the sign of the intermediate result
		RP	Modify negative overflows to the format's most negative finite number; modify positive overflows to + ∞
		RM	Modify positive overflows to the format's largest finite number; modify negative overflows to – ∞
Z	Division by zero	Any	Supply a properly signed ∞
V	Invalid operation	Any	Supply a quiet Not a Number (NaN)

Table 9-2 lists the exception-causing situations and contrasts the behavior of the FPU with the requirements of the IEEE Standard 754.

Table 9-2 FPU Exception-Causing Conditions

FPA Internal Result	IEEE Standard 754	Trap Enable	Trap Disable	Notes
Inexact result	I	I	I	Loss of accuracy
Exponent overflow	O,I[a]	O,I	O,I	Normalized exponent > E_{max}
Division by zero	Z	Z	Z	Zero is (exponent = E_{min}-1, mantissa = 0)
Overflow on convert	V	E	E	Source out of integer range
Signaling NaN source	V	V	V	
Invalid operation	V	V	V	0/0, etc.
Exponent underflow	U	E	E	Normalized exponent < E_{min}
Denormalized or QNaN	None	E	E	Denormalized is (exponent = E_{min}-1 and mantissa <> 0)

[a] The IEEE Standard 754 specifies an inexact exception on overflow only if the overflow trap is disabled.

9.4 FPU Exceptions

The following sections describe the conditions that cause the FPU to generate each of its exceptions, and details the FPU response to each exception-causing condition.

9.4.1 Inexact Exception (I)

The FPU generates the Inexact exception if one of the following occurs:

- the rounded result of an operation is not exact, or
- the rounded result of an operation overflows, or
- the rounded result of an operation underflows and both the Underflow and Inexact *Enable* bits are not set and the *FS* bit is set.

The FPU usually examines the operands of floating-point operations before execution actually begins, to determine (based on the exponent values of the operands) if the operation can *possibly* cause an exception. If there is a possibility of an instruction causing an exception trap, the FPU uses a coprocessor stall to execute the instruction.

It is impossible, however, for the FPU to predetermine if an instruction will produce an inexact result. If Inexact exception traps are enabled, the FPU uses the coprocessor stall mechanism to execute all floating-point operations that require more than one cycle. Since this mode of execution can impact performance, Inexact exception traps should be enabled only when necessary.

Trap Enabled Results: If Inexact exception traps are enabled, the result register is not modified and the source registers are preserved.

Trap Disabled Results: The rounded or overflowed result is delivered to the destination register if no other software trap occurs.

9.4.2 Invalid Operation Exception (V)

The Invalid Operation exception is signaled if one or both of the operands are invalid for an implemented operation. When the exception occurs without a trap, the MIPS ISA defines the result as a quiet Not a Number (NaN). The invalid operations are:

- Addition or subtraction: magnitude subtraction of infinities, such as: $(+ \infty) + (- \infty)$ or $(- \infty) - (- \infty)$
- Multiplication: 0 times ∞, with any signs
- Division: 0/0, or ∞/∞, with any signs
- Comparison of predicates involving **<** or **>** without **?**, when the operands are unordered
- Comparison or a Convert From Floating-point Operation on a signaling NaN.
- Any arithmetic operation on a signaling NaN. A move (MOV) operation is not considered to be an arithmetic operation, but absolute value (ABS) and negate (NEG) are considered to be arithmetic operations and cause this exception if one or both operands is a signaling NaN.
- Square root: √x, where x is less than zero

Software can simulate the Invalid Operation exception for other operations that are invalid for the given source operands. Examples of these operations include IEEE Standard 754-specified functions implemented in software, such as Remainder: *x* REM *y*, where *y* is 0 or *x* is infinite; conversion of a floating-point number to a decimal format whose value causes an overflow, is infinity, or is NaN; and transcendental functions, such as ln (–5) or cos–1(3).

Trap Enabled Results: The original operand values are undisturbed.

Trap Disabled Results: A quiet NaN is delivered to the destination register if no other software trap occurs.

9.4.3 Division-by-Zero Exception (Z)

The Division-by-Zero exception is signaled on an implemented divide operation if the divisor is zero and the dividend is a finite nonzero number. Software can simulate this exception for other operations that produce a signed infinity, such as ln(0), sec(π/2), csc(0), or $0^{-1.}$

Trap Enabled Results: The result register is not modified, and the source registers are preserved.

Trap Disabled Results: The result, when no trap occurs, is a correctly signed infinity.

9.4.4 Overflow Exception (O)

The Overflow exception is signaled when the magnitude of the rounded floating-point result, with an unbounded exponent range, is larger than the largest finite number of the destination format. (This exception also sets the Inexact exception and *Flag* bits.)

Trap Enabled Results: The result register is not modified, and the source registers are preserved.

Trap Disabled Results: The result, when no trap occurs, is determined by the rounding mode and the sign of the intermediate result (as listed in Table 9-1).

9.4.5 Underflow Exception (U)

Two related events contribute to the Underflow exception:

- creation of a tiny nonzero result between $\pm 2^{Emin}$ which can cause some later exception because it is so tiny
- extraordinary loss of accuracy during the approximation of such tiny numbers by denormalized numbers.

IEEE Standard 754 allows a variety of ways to detect these events, but requires they be detected the same way for all operations.

Tininess can be detected by one of the following methods:

- after rounding (when a nonzero result, computed as though the exponent range were unbounded, would lie strictly between $\pm 2^{Emin}$)
- before rounding (when a nonzero result, computed as though the exponent range and the precision were unbounded, would lie strictly between $\pm 2^{Emin}$).

The MIPS architecture requires that tininess be detected after rounding.

Loss of accuracy can be detected by one of the following methods:

- denormalization loss (when the delivered result differs from what would have been computed if the exponent range were unbounded)
- inexact result (when the delivered result differs from what would have been computed if the exponent range and precision were both unbounded).

The MIPS architecture requires that loss of accuracy be detected as an inexact result.

Trap Enabled Results: If Underflow or Inexact traps are enabled, or if the *FS* bit is not set, then an Unimplemented exception (E) is generated, and the result register is not modified.

Trap Disabled Results: If Underflow and Inexact traps are not enabled and the *FS* bit is set, the result is determined by the rounding mode and the sign of the intermediate result (as listed in Table 9-1).

9.4.6 Unimplemented Instruction Exception (E)

Any attempt to execute an instruction with an operation code or format code that has been reserved for future definition sets the *Unimplemented* bit in the *Cause* field in the FPU *Control/Status* register and traps. The operand and destination registers remain undisturbed and the instruction is emulated in software. Any of the IEEE Standard 754 exceptions can arise from the emulated operation, and these exceptions in turn are simulated.

The Unimplemented Instruction exception can also be signaled when unusual operands or result conditions are detected that the implemented hardware cannot handle properly. These include:

- Denormalized operand, except for Compare instruction
- Quiet Not a Number operand, except for Compare instruction
- Denormalized result or Underflow, when either Underflow or Inexact *Enable* bits are set or the *FS* bit is not set.
- Reserved opcodes
- Unimplemented formats
- Operations which are invalid for their format (for instance, CVT.S.S)

  NOTE: Denormalized and NaN operands are only trapped if the instruction is a convert or computational operation. Moves do not trap if their operands are either denormalized or NaNs.

On the V_R5000 additional causes of the unimplemented exception include:

- If the multiply portion of the madd, msub, nmadd, nmsub instruction would produce an overflow, underflow or denormal output

- ★ A floating-point to 64-bit fixed-point conversion with an output that would be greater than 2^{53}–1 (0×001F FFFF FFFF FFFF) or less than –2^{53} (0×FFE0 0000 0000 0000)

  Concerned instructions: CEIL.L.fmt, CVT.L.fmt, FLOOR.L.fmt, ROUND.L.fmt, TRUNC.L.fmt

- ★ A floating-point to 32-bit fixed-point conversion with an output that would be greater than 2^{31}–1 (0×7FFF FFFF) or less than –2^{31} (0×8000 0000)

  Concerned instructions: CEIL.W.fmt, CVT.W.fmt, FLOOR.W.fmt, ROUND.W.fmt, TRUNC.W.fmt

- ★ A 64-bit fixed-point to floating-point conversion with a source operand that would be greater than 2^{52}–1 (0×000F FFFF FFFF FFFF) or less than –2^{52} (0×FFF0 0000 0000 0000)

  Concerned instructions: CVT.D.fmt, CVT.S.fmt

- Attempting to execute a MIPS IV floating-point instruction if the MIPS IV instruction set has not been enabled

The use of this exception for such conditions is optional; most of these conditions are newly developed and are not expected to be widely used in early implementations. Loopholes are provided in the architecture so that these conditions can be implemented with assistance provided by software, maintaining full compatibility with the IEEE Standard 754.

Trap Enabled Results: The original operand values are undisturbed.

Trap Disabled Results: This trap cannot be disabled.

9.5 Saving and Restoring State

Sixteen or thirty-two doubleword coprocessor load or store operations save or restore the coprocessor floating-point register state in memory. The remainder of control and status information can be saved or restored through Move To/From Coprocessor Control Register instructions, and saving and restoring the processor registers. Normally, the *Control*/*Status* register is saved first and restored last.

When the coprocessor *Control/Status* register (*FCR31*) is read, and the coprocessor is executing one or more floating-point instructions, the instruction(s) in progress are either completed or reported as exceptions. The architecture requires that no more than one of these pending instructions can cause an exception. If the pending instruction cannot be completed, this instruction is placed in the *Exception* register, if present.

Information indicating the type of exception is placed in the *Control/Status* register. When state is restored, state information in the status word indicates that exceptions are pending.

Writing a zero value to the *Cause* field of *Control/Status* register clears all pending exceptions, permitting normal processing to restart after the floating-point register state is restored.

The *Cause* field of the *Control/Status* register holds the results of only one instruction; the FPU examines source operands before an operation is initiated to determine if this instruction can possibly cause an exception. If an exception is possible, the FPU executes the instruction in stall mode to ensure that no more than one instruction (that might cause an exception) is executed at a time.

9.6 Trap Handlers for IEEE Standard 754 Exceptions

The IEEE Standard 754 strongly recommends that users be allowed to specify a trap handler for any of the five standard exceptions that can compute; the trap handler can either compute or specify a substitute result to be placed in the destination register of the operation.

By retrieving an instruction using the processor *Exception Program Counter (EPC*) register, the trap handler determines:

- exceptions occurring during the operation
- the operation being performed
- the destination format

On Overflow or Underflow exceptions (except for conversions), and on Inexact exceptions, the trap handler gains access to the correctly rounded result by examining source registers and simulating the operation in software.

On Overflow or Underflow exceptions encountered on floating-point conversions, and on Invalid Operation and Divide-by-Zero exceptions, the trap handler gains access to the operand values by examining the source registers of the instruction.

The IEEE Standard 754 recommends that, if enabled, the overflow and underflow traps take precedence over a separate inexact trap. This prioritization is accomplished in software; hardware sets both bits.

Chapter 10 Initialization Interface

The V_R5000 processor has the following three types of resets; they use the **VccOk, ColdReset***, and **Reset*** input signals.

- **Power-on reset**: starts when the power supply is turned on and completely reinitializes the internal state machines of the processor without saving any state information.
- **Cold reset**: restarts all clocks, but the power supply remains stable. A cold reset completely reinitializes the internal state machines of the processor without saving any state information.
- **Warm reset**: restarts the processor, but does not affect clocks. A warm reset preserves the processor internal state.

The Initialization interface is a serial interface that operates at the frequency of the **SysClock** divided by 256: (**SysClock**/256). This low-frequency operation allows the initialization information to be stored in a low-cost ROM device.

10.1 Processor Reset Signals

This section describes the three reset signals, **VccOk, ColdReset***, and **Reset***.

V_{CC}Ok: When asserted[†], **V_{CC}Ok** indicates to the processor that the power supply (Vcc) has been within the specific range for more than 100 milliseconds (ms) and is expected to remain stable. The assertion of **VccOk** initiates the reading of the boot-time mode control serial stream (described in Initialization Sequence, in this chapter).

ColdReset*: The **ColdReset*** signal must be asserted (low) for either a power-on reset or a cold reset. **ColdReset*** must be deasserted synchronously with **SysClock**.

Reset*: the **Reset*** signal must be asserted for any reset sequence. It can be asserted synchronously or asynchronously for a cold reset, or synchronously to initiate a warm reset. **Reset*** must be deasserted synchronously with **SysClock**.

ModeIn: Serial boot mode data in.

ModeClock: Serial boot mode data clock, at the **SysClock** frequency divided by 256 (**SysClock**/256).

10.1.1 Power-on Reset

The sequence for a power-on reset is listed below.

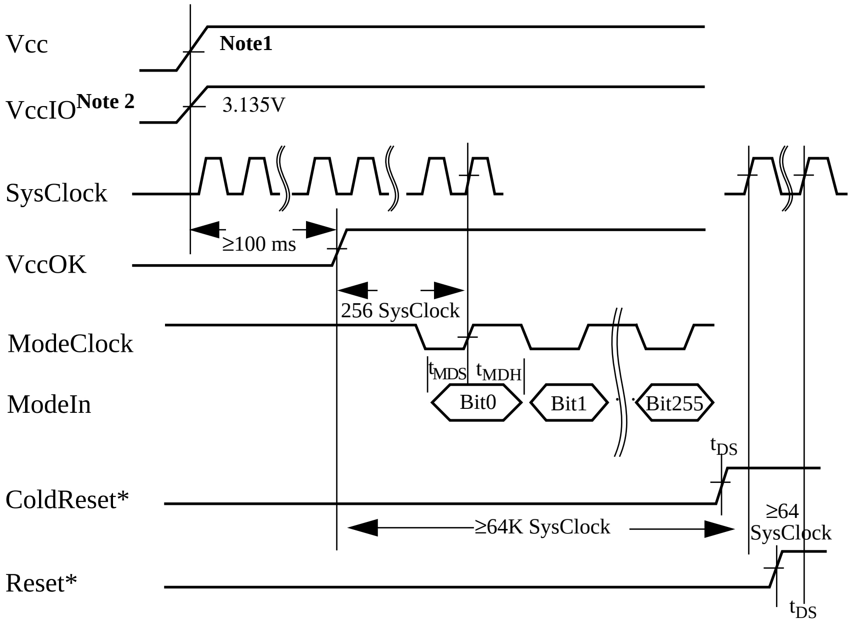
1. Power-on reset applies stable V_{CC} and V_{CC}IO**Note** within the specific range from the power supply to the processor. It also supplies a stable, continuous system clock at the processor operational frequency.
2. After at least 100 ms of stable $V_{CC,}$ V_{CC}IO**Note** and **SysClock**, the **V_{CC}Ok** signal is asserted to the processor. The assertion of **V_{CC}Ok** initializes the processor operating parameters. After the mode bits have been read in, the processor allows its internal phase locked loops to lock, stabilizing the processor internal clock, **PClock**.
3. **ColdReset*** is asserted for at least 64K (2^{16}) **SysClock** cycles after the assertion of **V_{CC}Ok**. Once the processor reads the boot-time mode control serial data stream, **ColdReset*** can be deasserted. **ColdReset*** must be deasserted synchronously with **SysClock**.
4. After **ColdReset*** is deasserted synchronously, **Reset*** is deasserted to allow the processor to begin running. (**Reset*** must be held asserted for at least 64 **SysClock** cycles after the deassertion of **ColdReset***.) **Reset*** must be deasserted synchronously with **SysClock**.

   NOTE: ColdReset* must be asserted when **V_{CC}Ok** asserts. The behavior of the processor is undefined if **V_{CC}Ok** asserts while **ColdReset*** is deasserted.

   Note V_{CC}IO is only for V_R5000A.

[†] *Asserted* means the signal is true, or in its valid state. For example, the low-active **Reset*** signal is said to be asserted when it is in a low (true) state; the high-active **V_{CC}Ok** signal is true when it is asserted high.

Figure 10-1 shows the power-on system reset timing diagram.



Notes 1. 3.135V (V_R5000), 2.3V (V_R5000A, 100 to 235MHz), 2.375V (V_R5000A, 236 to 250MHz), 2.5V (V_R5000A, 251 to 266MHz)

2. V_R5000A only

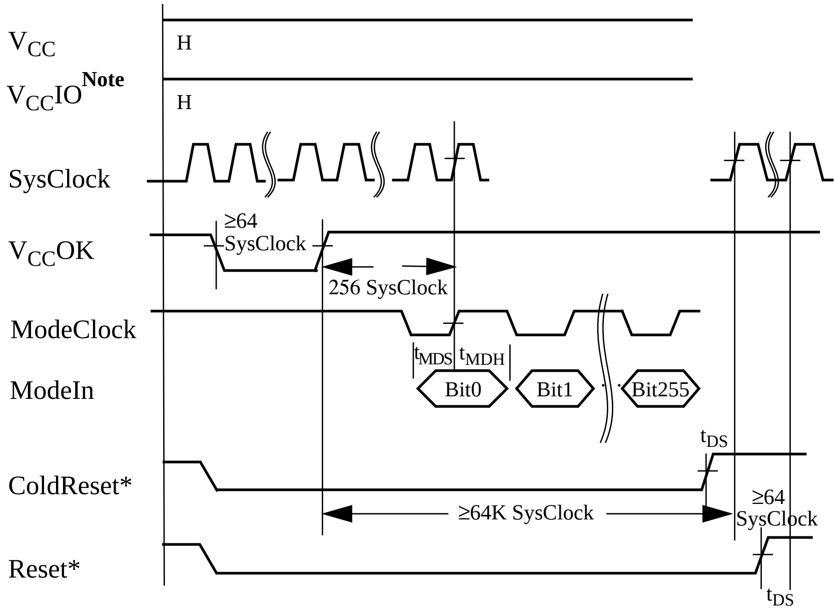
Figure 10-1 Power-on Reset Timing Diagram

10.1.2 Cold Reset

A cold reset can begin anytime after the processor has read the initialization data stream, causing the processor to start with the Reset exception. A cold reset requires the same sequence as a power-on reset except that the power is presumed to be stable before the assertion of the reset inputs and the deassertion of **V_{CC}Ok**.

To begin the reset sequence, **V_{CC}Ok** must be deasserted for a minimum of at least 64 MasterClock cycles before reassertion.

Figure 10-2 shows the cold reset timing diagram.



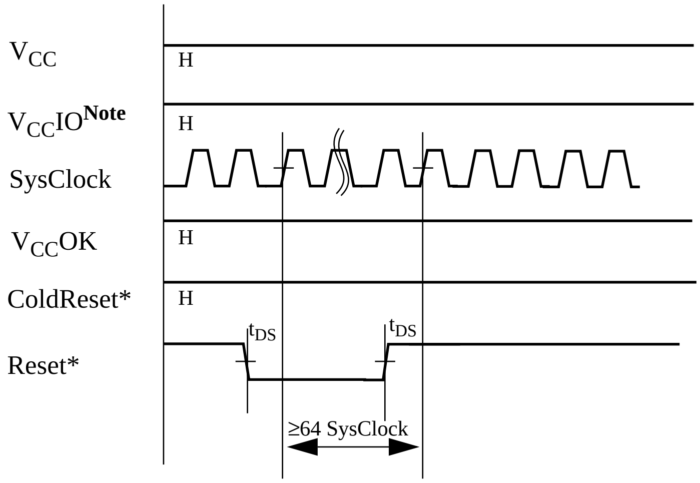
Note V_R5000A only

Figure 10-2 Cold Reset Timing Diagram

10.1.3 Warm Reset

To execute a warm reset, the **Reset*** input is asserted synchronously with **SysClock**. It is then held asserted for at least 64 **SysClock** cycles before being deasserted synchronously with **SysClock**. The boot-time mode control serial data stream is not read by the processor on a warm reset. A warm reset forces the processor to start with a Soft Reset exception.

Figure 10-3 shows the warm reset timing diagram.



Note V_R5000A only

Figure 10-3 Warm Reset Timing Diagram

10.1.4 Processor Reset State

After a power-on reset, cold reset, or warm reset, all processor internal state machines are reset, and the processor begins execution at the reset vector. All processor internal states are preserved during a warm reset, although the precise state of the caches depends on whether or not a cache miss sequence has been interrupted by resetting the processor state machines.

10.2 Initialization Sequence

The boot-mode initialization sequence begins immediately after $\mathbf{V_{CC}Ok}$ is asserted. As the processor reads the serial stream of 256 bits through the **ModeIn** pin, the boot-mode bits initialize all fundamental processor modes.

The initialization sequence is listed below.

1. The system deasserts the $\mathbf{V_{CC}Ok}$ signal. The **ModeClock** output is held asserted.
2. The processor synchronizes the **ModeClock** output at the time $\mathbf{V_{CC}Ok}$ is asserted. The first rising edge of **ModeClock** occurs 256 **SysClock** cycles after $\mathbf{V_{CC}Ok}$ is asserted.

3. Each bit of the initialization stream is presented at the **ModeIn** pin after each rising edge of the **ModeClock**. The processor samples 256 initialization bits from the **ModeIn** input.

10.3 Boot-Mode Settings

The following rules apply to the boot-mode settings:

- Bit 0 of the stream is presented to the processor when **V_{CC}Ok** is first asserted.
- Selecting a reserved value results in undefined processor behavior.
- Zeros must be scanned in for all reserved bits.

Table 10-1 shows the boot mode settings.

Table 10-1 Boot Mode Settings

Bit	Value	Mode Setting
0	Reserved: must be zero	
1:4	XmitDatPat: System interface data rate for block writes only	
	0	DDDD
	1	DDxDDx
	2	DDxxDDxx
	3	DxDxDxDx
	4	DDxxxDDxxx
	5	DDxxxxDDxxxx
	6	DxxDxxDxxDxx
	7	DDxxxxxxDDxxxxxx
	8	DxxxDxxxDxxxDxxx
	9:15	Reserved
5:7	SysCkRatio: PClock to SysClock Multiplier.	
	0	Multiply by 2
	1	Multiply by 3
	2	Multiply by 4
	3	Multiply by 5
	4	Multiply by 6
	5	Multiply by 7
	6	Multiply by 8
	7	Reserved
8	EndBit: Specifies byte ordering. Logically ORed with the BigEndian signal.	
	0	Little-Endian
	1	Big Endian
9:10	Non-Block Write: Determines how non-block writes are handled.	
	0	V_R4x00 compatible
	1	Reserved
	2	Pipelined writes
	3	Write-reissue
11	TmrIntEn: Disables Timer Interrupt on Int*[5]	
	0	Timer Interrupt Enabled
	1	Timer Interrupt Disabled

Bit	Value	Mode Setting
12	Secondary Cache Enable	
	0	Secondary Cache Disabled
	1	Secondary Cache Enabled
13:14	DrvOut: Output driver slew rate control	
	10	100% (fastest)
	11	83%
	00	67%
	01	50% (slowest)
15	Secondary cache SRAM protocol	
	0	Pipelined
	1	Burst
16:17	Secondary cache size	
	0	512 KB secondary cache
	1	1 MB secondary cache
	2	2 MB secondary cache
	3	Reserved
18	CP0 Count Register Update Rate	
	0	1/2 x PClocK
	1	1 x PClocK
19	Reserved: Must be zero	
20	Reserved: Must be zero However, must be set for Rev. 2.41 or lower of V_R5000	
21:32	Reserved: Must be zero	
33	Reserved: Must be zero However, must be set for Rev. 2.41 or lower of V_R5000	
34:36	Reserved: Must be zero	
37	Reserved: Must be zero However, must be set for Rev. 2.x or lower of V_R5000	
38	Enable 2.5PClock to SysClock Multiplier[Note 1, Note 2]	
	0	Disable
	1	Enable
39:255	Reserved: Must be zero	

Notes 1. This is for V_R5000A. This bit must be zero for V_R5000.

2. In case bit38 is set, the SysCkRatio (bit5-7) is ignored.

Chapter 11   Clock Interface

11.1 Basic System Clocks

The various clock signals used in the V_R5000 processor are described below, starting with **SysClock**, upon which the processor bases all internal and external clocking.

11.1.1 SysClock

The processor bases all internal and external clocking on the single **SysClock** input signal.

11.1.2 PClock

The processor generates an internal clock, PClock, at the initialization-interface-specified frequency multiplier of **SysClock** and phase-aligned to **SysClock**. All internal registers and latches use **PClock**.

11.1.3 Alignment to SysClock

- Processor output data changes a minimum of t_{DM} ns and becomes stable a maximum of t_{DO} ns after the rising edge of **SysClock**. This drive-time is the sum of the maximum delay through the processor output drivers together with the maximum clock-to-Q delay of the processor output registers.
- Processor input data must be stable for a maximum of t_{DS} ns before the rising edge of **SysClock** and must remain stable a minimum of t_{DH} ns after the rising edge of **SysClock**.

11.1.4 Phase-Locked Loop (PLL)

The processor aligns **PClock** and **SysClock** with internal phase-locked loop (PLL) circuits that generate aligned clocks. By their nature, PLL circuits are only capable of generating aligned clocks for **SysClock** frequencies within a limited range.

Clocks generated using PLL circuits contain some inherent inaccuracy, or jitter; a clock aligned with **SysClock** by the PLL can lead or trail **SysClock** by as much as the related maximum jitter t_{ji} allowed by the individual vendor. The t_{ji} parameter must be added to the t_{DS}, t_{DH}, and t_{DO} parameters, and subtracted from the t_{DM} parameters to get the total input and output timing parameters.

Figure 11-1 shows the **SysClock** timing parameters.

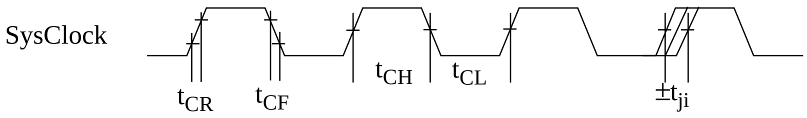


Figure 11-1 SysClock Timing

11.2 Connecting Clocks to a Phase-Locked System

When the processor is used in a phase-locked system, the external agent must phase lock its operation to a common **SysClock**. In such a system, the transmission of data and data sampling have common characteristics, even if the components have different delay values. For example, *transmission time* (the amount of time a signal takes to move from one component to another along a trace on the board) between any two components A and B of a phase-locked system can be calculated from the following equation:

Transmission Time = (SClock period) – (t_{DO} for A) – (t_{DS} for B) –
(Clock Jitter for A Max) – (Clock Jitter for B Max)

Figure 11-2 shows a block-level diagram of a phase-locked system using the V_R5000 processor.

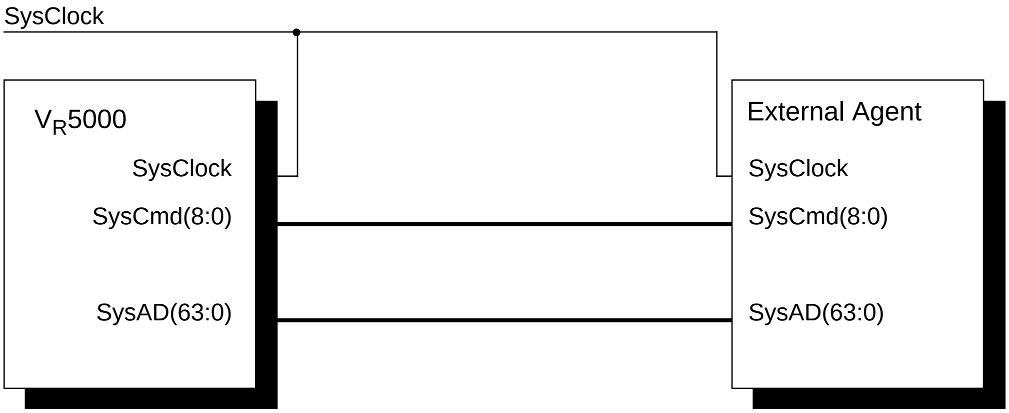


Figure 11-2 Phase-Locked System

Chapter 12 Cache Organization and Operation

This chapter describes in detail the cache memory: its place in the V_R5000 memory organization, and individual organization of the caches.

This chapter uses the following terminology:

- The data cache may also be referred to as the D-cache.
- The instruction cache may also be referred to as the I-cache.

These terms are used interchangeably throughout this book.

12.1 Memory Organization

Figure 12-1 shows the V_R5000 system memory hierarchy. In the logical memory hierarchy, the caches lie between the CPU and main memory. They are designed to make the speedup of memory accesses transparent to the user.

Each functional block in Figure 12-1 has the capacity to hold more data than the block above it. For instance, physical main memory has a larger capacity than the caches. At the same time, each functional block takes longer to access than any block above it. For instance, it takes longer to access data in main memory than in the CPU on-chip registers.

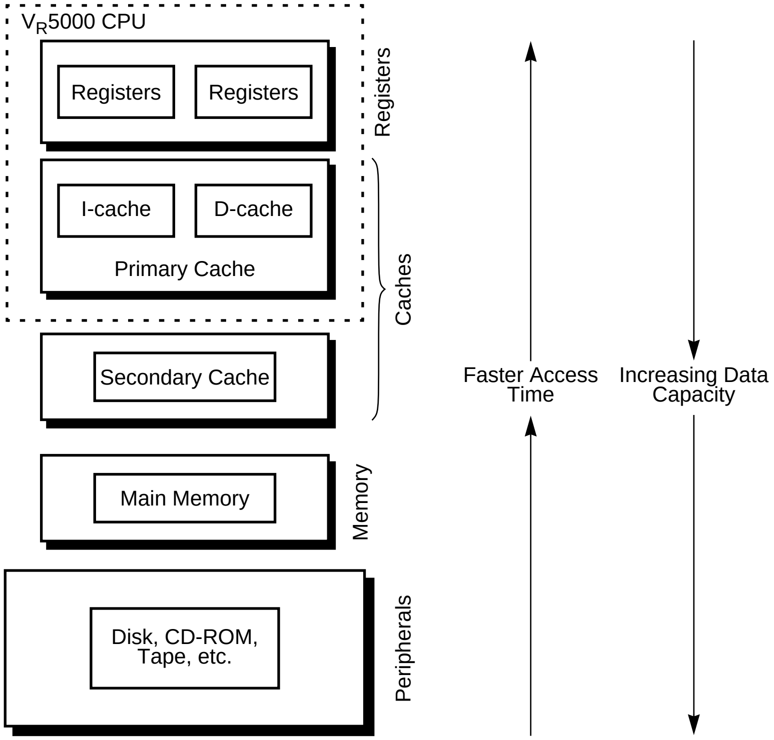


Figure 12-1 Logical Hierarchy of Memory

The V_R5000 processor has two on-chip caches: one holds instructions (the instruction cache), the other holds data (the data cache). The instruction and data caches can be read in one PClock cycle.

Data writes are pipelined and can complete at a rate of one per PClock cycle. In the first stage of the cycle, the store address is translated and the tag is checked; in the second stage, the data is written into the data RAM.

Figure 12-2 provides a block diagram of the V_R5000 cache and memory model.

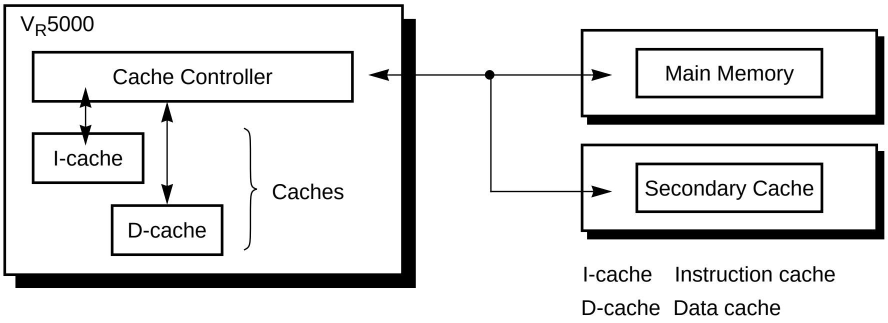


Figure 12-2 V_R5000 Cache Support

12.2 Primary Cache Organization

This section describes the organization of the on-chip data and instructio caches.

12.2.1 Cache Line Lengths

A cache line is the smallest unit of information that can be fetched from main memory for the cache, and that is represented by a single tag.

The line size fot the instruciton/data cache is 32 bytes.

12.2.2 Cache Sizes

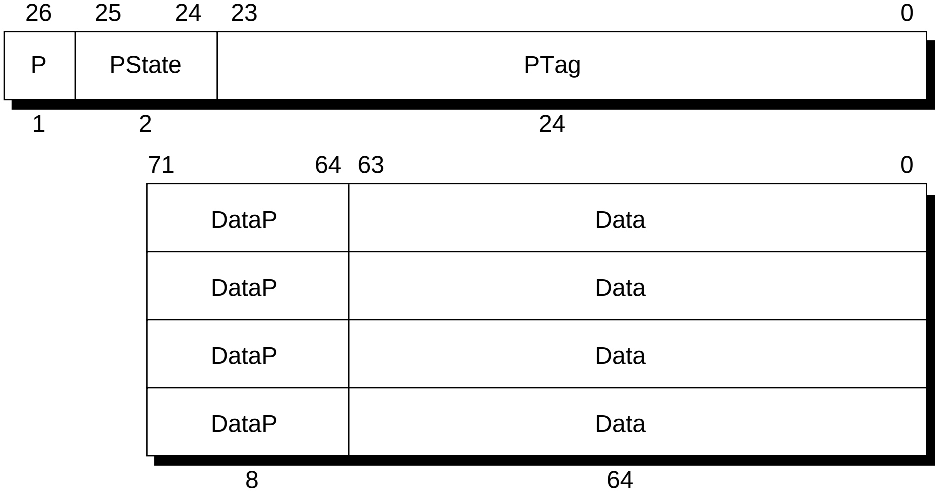
The V_R5000 instruciton cache is 32 KB; the data cache is 32 KB.

12.2.3 Organization of the Instruction Cache (I-Cache)

The V_R5000 procesosr I-cache has the following characteristics:

- 2-way set associative
- indexed with a virtual address
- checked with a physical tag

organized with a 32-byte cache line.



P: Even parity for the PTag
PState: Primary cache state
PTag: Primary cache tag (bits 35:12 of the physical address)
DataP: Even parity for the data
Data: I-cache data

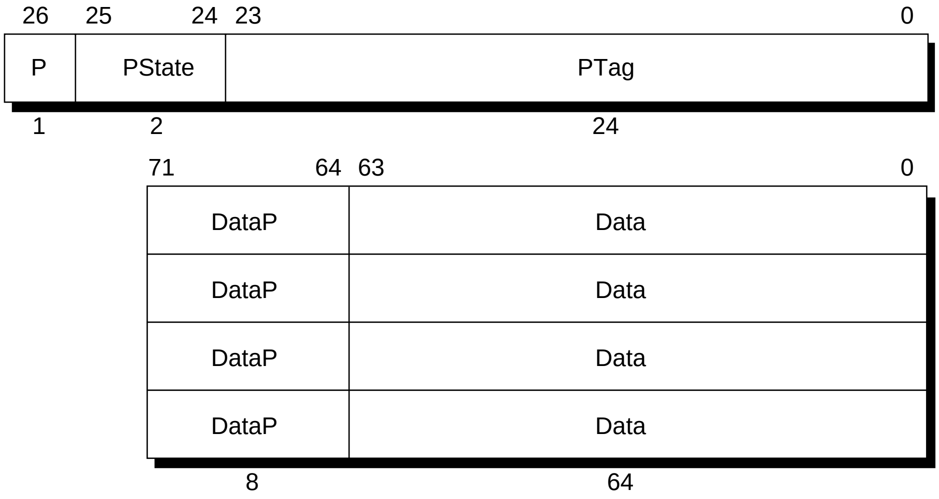
Figure 12-3 Primary Instruction Cache Line Format

12.2.4 Organization of the Data Cache (D-Cache)

The V_R5000 processor D-cache has the following characteristics:

- write-back or write-through
- 2-way set associative
- indexed with a virtual address
- checked with a physical tag

organized with a 32-byte cache line.



P: Even parity for the PTag
PState: Primary cache state
PTag: Primary cache tag (bits 35:12 of the physical address)
DataP: Even parity for the data
Data: D-cache data

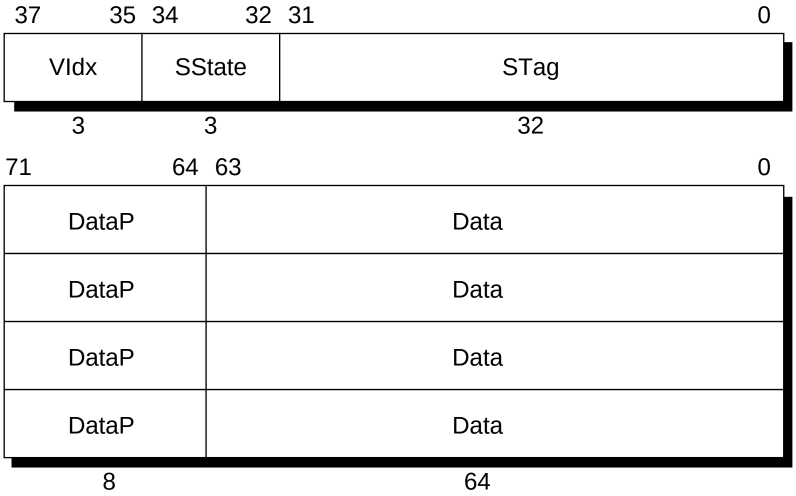
Figure 12-4 Primary Data Cache Line Format

12.3 Secondary Cache Organization

The V_R5000 has a secondary cache interface and can operate with an external secondary cache.

The secondary cache is:

- direct-mapped
- indexed with a virtual address
- checked with a physical tag
- organized with an 8-word (32-byte) cache line
- either 512 KB, 1 MB, or 2 MB in size.



VIdx: Virtual index of the associated primary cache line (bits 14:12 of the virtual address)
SState: Secondary cache state
STag: Secondary cache tag (bits 35:17 of the physical address)
DataP: Even parity for the data
Data: Secondary cache data

Figure 12-5 Secondary Cache Line Format

Chapter 13 V_R5000 Processor Bus Interface

The System interface allows the processor to access external resources needed to satisfy cache misses and uncached operations, while permitting an external agent access to some of the processor internal resources.

The clock portion of the V_R5000 system interface has been simplified and many of the external clock signals have been deleted from the system interface of the V_R4000 Series.

The V_R5000 processor supports up to a 100 MHz pipelined **SysAD** bus. V_R5000 also implements a unified, write-through secondary cache which has the same 32-byte line size as the primary caches. Secondary cache index and control signals are supplied by the processor. Secondary cache sizes of 512 KB, 1 MB, and 2 MB are supported.

This chapter describes the System interface from the point of view of both the processor and the external agent.

13.1 Terms Used

The following terms are used in this document:

- An *external agent* is any logic device connected to the processor, over the System interface, that allows the processor to issue requests.
- A *system event* is an event that occurs within the processor and requires access to external system resources.
- *Sequence* refers to the precise series of requests that a processor generates to service a system event.
- *Protocol* refers to the cycle-by-cycle signal transitions that occur on the System interface pins to assert a processor or external request.
- *Syntax* refers to the precise definition of bit patterns on encoded buses, such as the command bus.

13.2 Interface Buses

Figure 13-1 shows the primary communication paths for the System interface: a 64-bit address and data bus, **SysAD[63:0]**, and a 9-bit command bus, **SysCmd[8:0]**. The **SysAD** and the **SysCmd** buses are bidirectional; that is, they are driven by the processor to issue a processor request, and by the external agent to issue an external request.

A request through the System interface consists of:

- an address
- a System interface command that specifies the precise nature of the request
- a series of data elements if the request is for a write or read response.

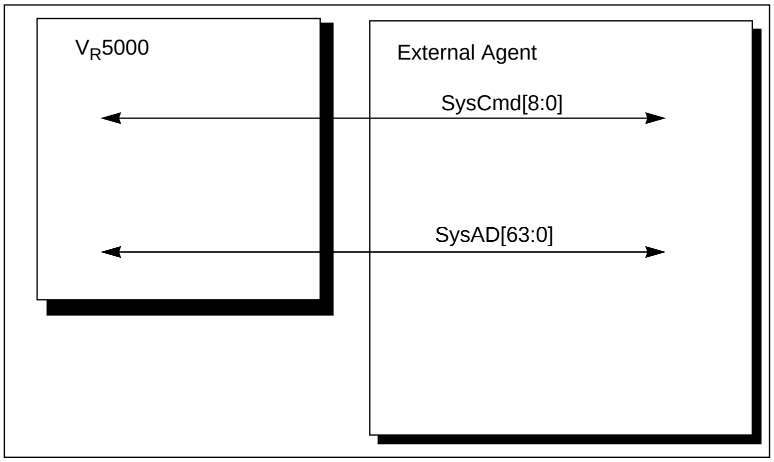


Figure 13-1 System Interface Buses

Figure 13-2 shows the primary communication paths for a secondary cache configuration. The secondary cache shares the **SysAD** and **SysADC** buses between the processor and the external agent. The processor implements the **ScLine** and **ScWord** address buses to the secondary cache to access a cache line within the secondary cache and 64-bit cache doublewords within the cache line, respectively.

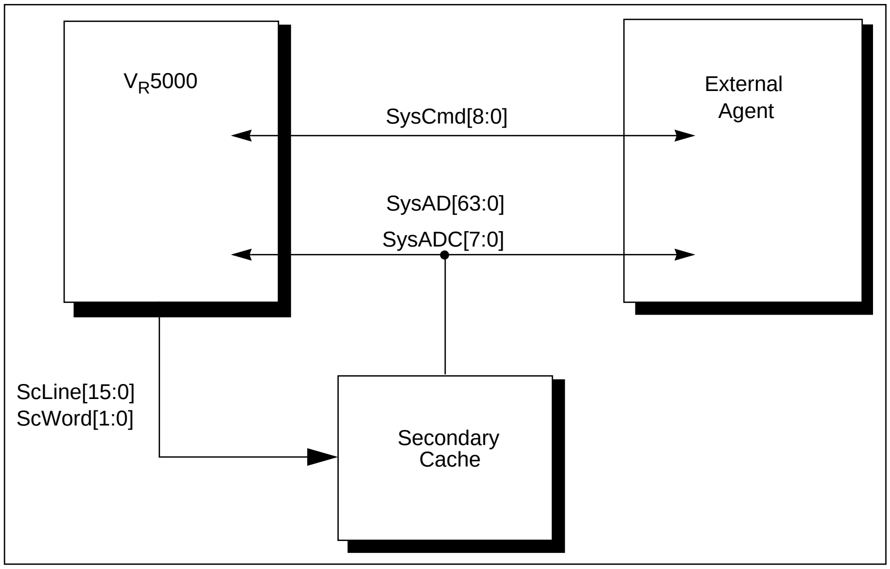


Figure 13-2 Secondary Cache Interface

Chapter 14 System Interface Transactions

There are two broad categories of transactions: *processor requests* and *external requests*. This chapter describes them.

14.1 Processor Requests

The processor issues either a single request or a series of requests—called *processor requests*—through the System interface, to access an external resource. For this to work, the processor System interface must be connected to an external agent that is compatible with the System interface protocol, and can coordinate access to system resources.

An external agent requesting access to a processor internal resource generates an *external request*. This access request passes through the System interface. System events and request cycles are shown in Figure 14-1.

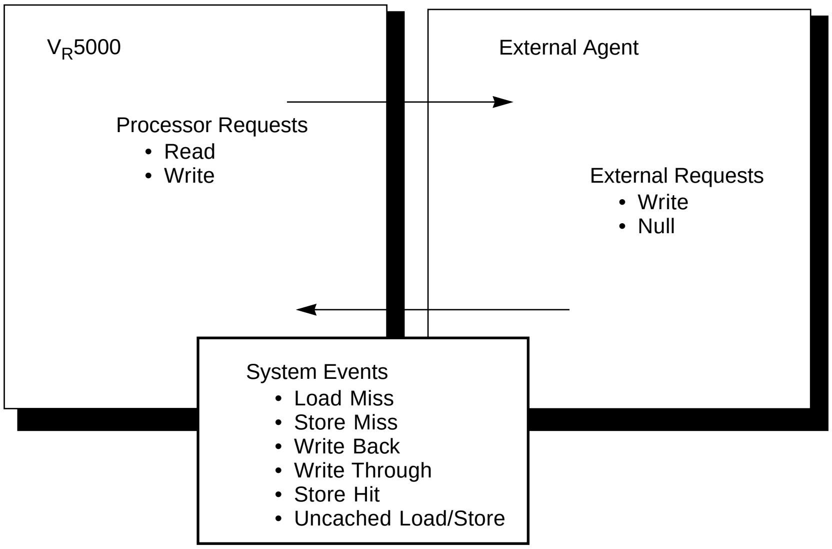


Figure 14-1 Requests and System Events

14.1.1 Rules for Processor Requests

A processor request is a request or a series of requests, through the System interface, to access some external resource. As shown in Figure 14-2, processor requests include read and write.

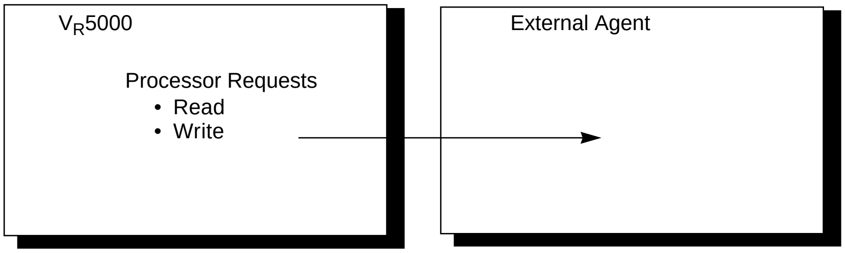


Figure 14-2 Processor Requests to External Agent

Read request asks for a block, doubleword, partial doubleword, word, or partial word of data either from main memory or from another system resource.

Write request provides a block, doubleword, partial doubleword, word, or partial word of data to be written either to main memory or to another system resource.

The processor is only allowed to have one request pending at any time. For example, the processor issues a read request and waits for a read response before issuing any subsequent requests. The processor submits a write request only if there are no read requests pending.

The processor has the input signals **RdRdy*** and **WrRdy*** to allow an external agent to manage the flow of processor requests. **RdRdy*** controls the flow of processor read requests, while **WrRdy*** controls the flow of processor write requests. The processor request cycle sequence is shown in Figure 14-3.

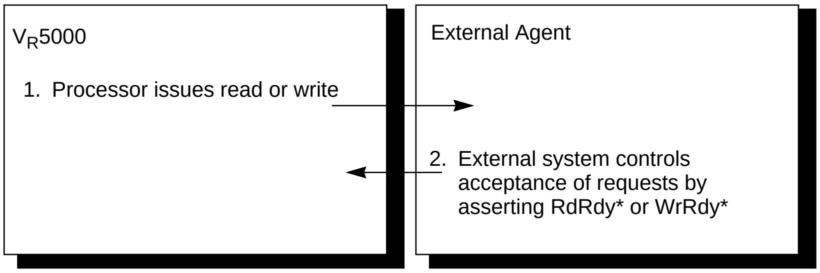

V_R5000
1. Processor issues read or write
External Agent
2. External system controls acceptance of requests by asserting RdRdy* or WrRdy*

Figure 14-3   Processor Request Flow Control

14.1.2 Processor Read Request

When a processor issues a read request, the external agent must access the specified resource and return the requested data.

A processor read request can be split from the external agent’s return of the requested data; in other words, the external agent can initiate an unrelated external request before it returns the response data for a processor read. A processor read request is completed after the last word of response data has been received from the external agent.

Note that the data identifier associated with the response data can signal that the returned data is erroneous, causing the processor to take a bus error.

Processor read requests that have been issued, but for which data has not yet been returned, are said to be *pending*. A read remains pending until the requested read data is returned.

The external agent must be capable of accepting a processor read request any time the following two conditions are met:

- There is no processor read request pending.
- The signal **RdRdy*** has been asserted for two or more cycles before the issue cycle.

14.1.3 Processor Write Request

When a processor issues a write request, the specified resource is accessed and the data is written to it. A processor write request is complete after the last word of data has been transmitted to the external agent. The V_R5000 processor supports *V_R4000 compatible, write-reissue* and *pipelined write* operations as defined in Chapter 15.

The external agent must be capable of accepting a processor write request any time the following two conditions are met:

- No processor read request is pending.
- The signal **WrRdy*** has been asserted for two or more cycles.

14.2 External Requests

External requests include write, and null requests, as shown in Figure 14-4. This section also includes a description of read response, a special case of an external request.

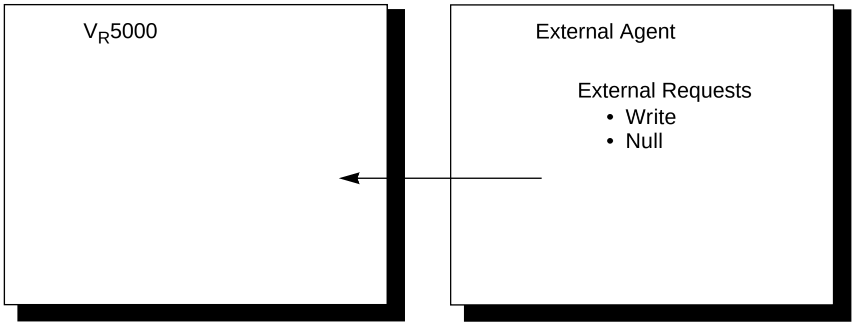
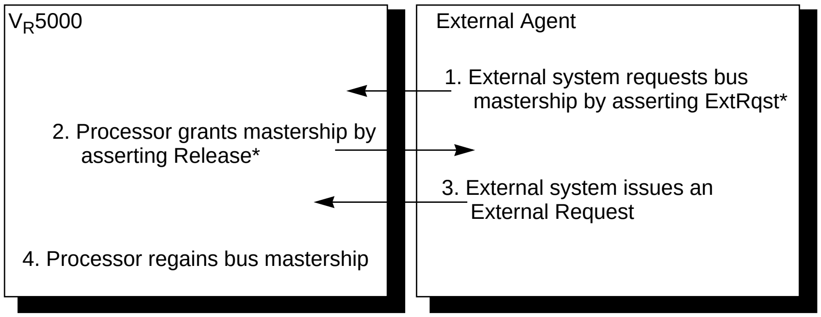


Figure 14-4 External Requests to Processor

Write request provides a word of data to be written to the processor’s internal resource.

Null request requires no action by the processor; it provides a mechanism for the external agent to return the System interface to the master state without affecting the processor.

The processor controls the flow of external requests through the arbitration signals **ExtRqst*** and **Release***, as shown in Figure 14-5. The external agent must acquire mastership of the System interface before it is allowed to issue an external request; the external agent arbitrates for mastership of the System interface by asserting **ExtRqst*** and then waiting for the processor to assert **Release*** for one cycle. If **Release*** is asserted as part of an uncompelled change to slave state during a processor read request, and the secondary cache is enabled, the secondary cache access must be resolved and be a miss. Otherwise the system interface returns to the master state.



V_R5000
External Agent
1. External system requests bus mastership by asserting ExtRqst*
2. Processor grants mastership by asserting Release*
3. External system issues an External Request
4. Processor regains bus mastership

Figure 14-5 External Request Arbitration

Mastership of the System interface always returns to the processor after an external request is issued. The processor does not accept a subsequent external request until it has completed the current request.

If there are no processor requests pending, the processor decides, based on its internal state, whether to accept the external request, or to issue a new processor request. The processor can issue a new processor request even if the external agent is requesting access to the System interface.

The external agent asserts **ExtRqst*** indicating that it wishes to begin an external request. The external agent then waits for the processor to signal that it is ready to accept this request by asserting **Release***. The processor signals that it is ready to accept an external request based on the criteria listed below.

- The processor completes any request in progress.

- While waiting for the assertion of **RdRdy*** to issue a processor read request, the processor can accept an external request if the external request is delivered to the processor one or more cycles before **RdRdy*** is asserted.
- While waiting for the assertion of **WrRdy***** to issue a processor write request, the processor can accept an external request provided the external request is delivered to the processor one or more cycles before **WrRdy*** is asserted.
- If waiting for the response to a read request after the processor has made an uncompelled change to a slave state, the external agent can issue an external request before providing the read response data.

14.2.1 External Write Request

When an external agent issues a write request, the specified resource is accessed and the data is written to it. An external write request is complete after the word of data has been transmitted to the processor.

The only processor resource available to an external write request is the Interrupt register. Refer to Chapter 17 for more information.

14.2.2 Read Response

A *read response* returns data in response to a processor read request, as shown in Figure 14-6. While a read response is technically an external request, it has one characteristic that differentiates it from all other external requests—it does not perform System interface arbitration. For this reason, read responses are handled separately from all other external requests, and are simply called read responses.

The data identifier associated with the response data can signal that the returned data is erroneous, causing the processor to take a bus error.

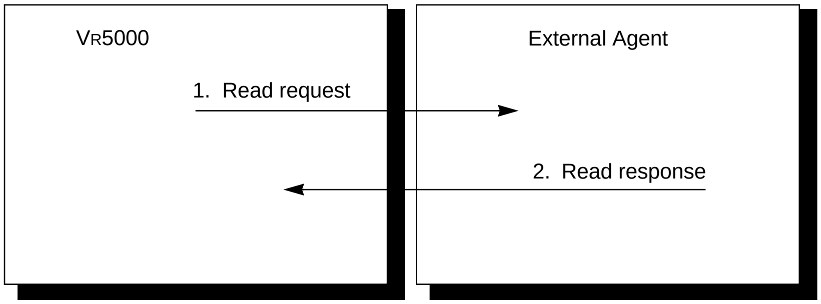


Figure 14-6   External Agent Read Response to Processor

14.3 Handling Requests

This section details the *sequence*, *protocol*, and *syntax* of both processor and external requests. The following system events are discussed:

- load miss
- store miss
- store hit
- uncached loads/stores
- uncached instruction fetch
- load linked store conditional

14.3.1 Load Miss

When a processor load misses in the primary cache, before the processor can proceed it must obtain the cache line that contains the data element to be loaded from the external agent.

If the new cache line replaces a current dirty exclusive or dirty shared cache line, the current cache line must be written back before the new line can be loaded in the primary cache.

The processor examines the coherency attribute in the TLB entry for the page that contains the requested cache line, and executes one of the following request:

- The coherency attribute is *noncoherent*, the processor issues a noncoherent read request.

Table 14-1 shows the actions taken on a load miss to primary cache.

Table 14-1   Load Miss to Primary Caches

Page Attribute	**State of Data Cache Line Being Replaced**	
	Clean/Invalid	**Dirty (W=1)**
Noncoherent	NCBR	NCBR/W

NCBR................. Processor noncoherent block read request

NCBR/W............ Processor noncoherent block read request followed by processor block write request

The processor takes the following steps:

1. The processor issues a noncoherent block read request for the cache line that contains the data element to be loaded. If the secondary cache is enabled and the page coherency attribute is write-back, the response data will also be written into the secondary cache.
2. The processor then waits for an external agent to provide the read response.
3. The processor restarts the pipeline after the first doubleword of the data cache miss is received. The remaining three doublewords are placed in the cache after all three doublewords have been received and the dcache is otherwise idle.

If the current cache line must be written back, the processor issues a block write request to save the dirty cache line in memory. If the secondary cache is enabled and the page attribute is write-back, the write back data will also be written into the secondary cache.

14.3.2 Store Miss

When a processor store misses in the primary cache, the processor may request, from the external agent, the cache line that contains the target location of the store for pages that are either write-back or write-through with write-allocate only. The processor examines the coherency attribute in the TLB entry for the page that contains the requested cache line to see if the cache line is being maintained with either a write-allocate or no-write-allocate policy.

The processor then executes one of the following requests:

- If the coherency attribute is *noncoherent* write-back, or write-through with write-allocate, a noncoherent block read request is issued.
- If the coherency attribute is *noncoherent* write-through with no write-allocate, a non-block write request is issued.

Table 14-2 shows the actions taken on a store miss to the primary cache.

Table 14-2 Store Miss to Primary and Secondary Caches

Page Attribute	**State of Data Cache Line Being Replaced**	
	Clean/Invalid	**Dirty (W=1)**
Noncoherent-write-back or noncoherent-write-through with write-allocate	NCBR	NCBR/W
Noncoherent-write-through with no-write-allocate	NCW	NA

NCBR................. Processor noncoherent block read request

NCBR/W............ Processor noncoherent block read request followed by processor block write request

NCW .... Processor noncoherent write request

If the coherency attribute is write-back, or write-through with write-allocate, the processor issues a non-coherent block read request for the cache line that contains the data element to be loaded, then waits for the external agent to provide read data in response to the read request. If the secondary cache is enabled and the page coherency attribute is write-back, the response data will also be written into the secondary cache. If the current cache line must be written back, the processor issues a write request for the current cache line.

If the page coherency attribute is write-through, the processor issues a non-block write request.

For a write-through, no-write-allocate store miss, the processor issues a non-block write request only.

14.3.3 Store Hit

The action on the system bus is determined by whether the line is write-back or write-through. For lines with a write-back policy, a store hit does not cause any processor request on the bus. For lines with a write-through policy, the store generates a processor non-block write request for the store data.

14.3.4 Uncached Loads or Stores

When the processor performs an uncached load, it issues a noncoherent doubleword, partial doubleword, word, or partial word read request. When the processor performs an uncached store, it issues a doubleword, partial doubleword, word, or partial word write request. All writes by the processor are buffered from the system interface by a 4-entry write buffer. The write requests are sent to the system bus only when no other requests are in progress. However, once the emptying of the write buffer has begun, it is allowed to complete. Therefore, if the write buffer contains any entries when a block read is requested, the write buffer is allowed to empty before the block read request is serviced. Uncached loads and stores do not affect the secondary cache.

14.3.5 Uncached Instruction Fetch

The processor issues doubleword reads for instruction fetches to uncached addresses. Thus any system ROM address space accessed during a processor boot-restart must support 64-bit reads.

14.3.6 Load Linked Store Conditional Operation

The execution of a Load-Linked/Store-Conditional instruction sequence is not visible at the System interface; that is, no special requests are generated due to the execution of this instruction sequence.

Chapter 15   System Interface Protocols

The following sections contain a cycle-by-cycle description of the system interface protocols for each type of processor and external request.

15.1 Address and Data Cycles

Cycles in which the **SysAD** bus contains a valid address are called *address cycles*. Cycles in which the **SysAD** bus contains valid data are called *data cycles*. Validity of addresses and data from the processor is determined by the state of the **ValidOut*** signal. Validity of addreses and data from the external agent is determined by the state of the **ValidIn*** signal. Validity of data from the secondary cache is determined by the state of the pipelined **ScDCE*** and **ScCWE*** signals from the processor and the **ScDOE*** signal from the external agent.

The **SysCmd** bus identifies the contents of the **SysAD** bus during any cycle in which it is valid from the processor or the external agent. The most significant bit of the **SysCmd** bus is always used to indicate whether the current cycle is an address cycle or a data cycle.

- During address cycles **SysCmd(8)** = 0. The remainder of the **SysCmd** bus, **SysCmd(7:0)**, contains the encoded system interface command.

- During data cycles [**SysCmd(8)** = 1], the remainder of the **SysCmd** bus, **SysCmd(7:0)**, contains an encoded data identifier. There is no **SysCmd** associated with a secondary cache read response.

15.2 Issue Cycles

There are two types of processor issue cycles:

- processor read request.
- processor write request.

The processor samples the signal **RdRdy*** to determine the issue cycle for a processor read; the processor samples the signal **WrRdy*** to determine the issue cycle of a processor write request.

As shown in Figure 15-1, **RdRdy*** must be asserted two cycles prior to the address cycle of the processor read request in order to define the address cycle as the issue cycle.

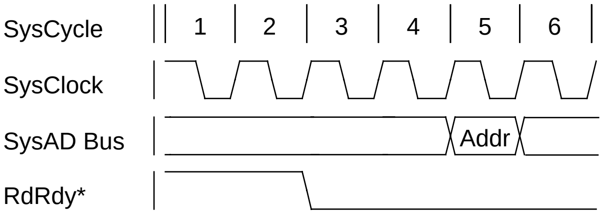


Figure 15-1 State of RdRdy Signal for Read Requests*

As shown in Figure 15-2, **WrRdy*** must be asserted two cycles prior to the first address cycle of the processor write request in order to define the address cycle as the issue cycle.

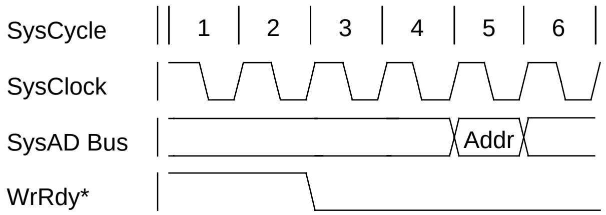


Figure 15-2   State of WrRdy Signal for Write Requests*

The processor repeats the address cycle for the request until the conditions for a valid issue cycle are met. After the issue cycle, if the processor request requires data to be sent, the data transmission begins. There is only one issue cycle for any processor request.

The processor accepts external requests, even while attempting to issue a processor request, by releasing the System interface to slave state in response to an assertion of **ExtRqst*** by the external agent.

Note that the rules governing the issue cycle of a processor request are strictly applied to determine which action the processor takes. The processor can either:

- complete the issuance of the processor request in its entirety before the external request is accepted, or
- release the System interface to slave state without completing the issuance of the processor request.

In the latter case, the processor issues the processor request (provided the processor request is still necessary) after the external request is complete. The rules governing an issue cycle again apply to the processor request.

15.3   Handshake Signals

The V_R5000 processor manages the flow of requests through the following six control signals:

- **RdRdy***, **WrRdy*** are used by the external agent to indicate when it can accept a new read (**RdRdy***) or write (**WrRdy***) transaction.

- **ExtRqst***, **Release*** are used to transfer control of the **SysAD** and **SysCmd** buses. **ExtRqst*** is used by an external agent to indicate a need to control the interface. **Release*** is asserted by the processor when it transfers the mastership of the System interface to the external agent. For secondary cache reads, assertion of **Release*** to the external agent is speculative, and is aborted if there is a hit in the secondary cache.
- The V_R5000 processor uses **ValidOut*** and the external agent uses **ValidIn*** to indicate valid command/data on the **SysCmd**/**SysAD** buses.
- The secondary cache uses the **ScDCE***, **ScCWE*** and **ScDOE*** signals to control validation on the **SysAD** and **SysADC** buses.

15.4 System Interface Operation

Figure 15-3 shows how the system interface operates from register to register. That is, processor outputs come directly from output registers and begin to change with the rising edge of **SysClock.**

Processor inputs are fed directly to input registers that latch these input signals with the rising edge of **SysClock**. This allows the System interface to run at the highest possible clock frequency.

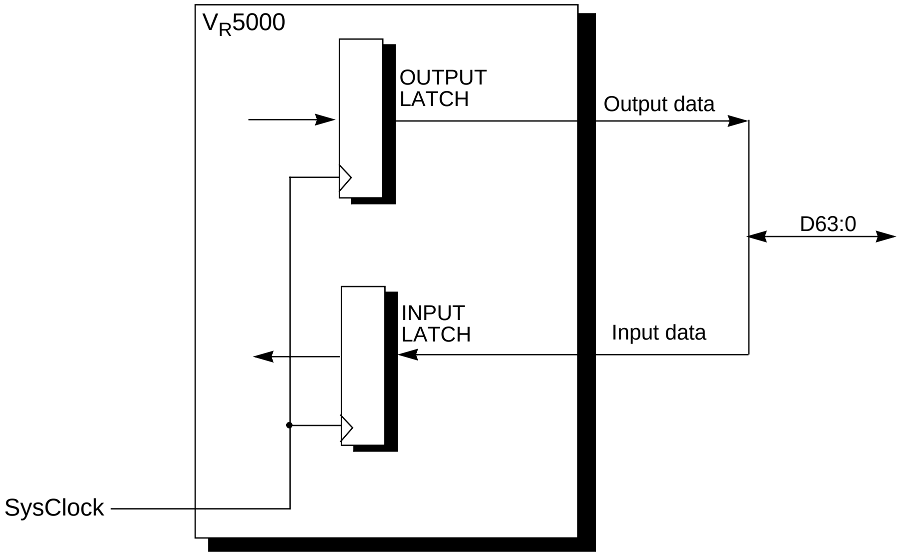


Figure 15-3 System Interface Register-to-Register Operation

15.4.1 Master and Slave States

When the V$_R$5000 processor is driving the **SysAD** and **SysCmd** buses, the System interface is in *master state*. When the external agent is driving the **SysAD** and **SysCmd** buses, the System interface is in *slave state*. When the secondary cache is driving the **SysAD** and **SysADC** buses, the System interface is in slave state.

In master state, the processor asserts the signal **ValidOut*** whenever the **SysAD** and **SysCmd** buses are valid.

In slave state, the external agent asserts the signal **ValidIn*** whenever the **SysAD** and **SysCmd** buses are valid and the secondary cache drives the **SysAD** and **SysADC** buses in response to the **ScDCE***, **ScCWE***, and **ScDOE*** signals.

The System interface remains in master state unless one of the following occurs:

- The external agent requests and is granted the System interface (external arbitration).
- The processor issues a read request.

15.4.2 External Arbitration

The System interface must be in slave state for the external agent to issue an external request through the System interface. The transition from master state to slave state is arbitrated by the processor using the System interface handshake signals **ExtRqst*** and **Release***. This transition is described by the following procedure:

1. An external agent signals that it wishes to issue an external request by asserting **ExtRqst***.
2. When the processor is ready to accept an external request, it releases the System interface from master to slave state by asserting **Release*** for one cycle.
3. The System interface returns to master state as soon as the issue of the external request is complete.

15.4.3 Uncompelled Change to Slave State

An *uncompelled* change to slave state is the transition of the System interface from master state to slave state, initiated by the processor when a processor read request is pending. **Release*** is asserted automatically at the same time a read request is issued and an uncompelled change to slave state then occurs. This transition to slave state allows the external agent to return read response data without arbitrating for bus ownership.

If the secondary cache is enabled and a secondary cache hit occurs, then the bus is returned to master state.

After an uncompelled change to slave state, the processor returns to master state at the end of the next external request. This can be a read response, or some other type of external request. If the external agent issues some other type of external request while there is a pending read request, the processor performs another uncompelled change to slave state by asserting **Release*** for one cycle.

An external agent must note that the processor has performed an uncompelled change to slave state and begin driving the **SysAD** bus along with the **SysCmd** bus. As long as the System interface is in slave state, the external agent can begin an external request without arbitrating for the System interface; that is, without asserting **ExtRqst***.

Table 15-1 lists the abbreviations and definitions for each of the buses that are used in the timing diagrams that follow.

Table 15-1 System Interface Requests

Scope	Abbreviation	Meaning
Global	Unsd	Unused
SysAD bus	Addr	Physical address
	Data<n>	Data element number n of a block of data
SysCmd bus	Cmd	An unspecified System interface command
	Read	A processor read request command
	Write	A processor or external write request command
	SINull	A System interface release external null request command
	NData	A noncoherent data identifier for a data element other than the last data element
	NEOD	A noncoherent data identifier for the last data element

15.5 Processor Request Protocols

Processor request protocols described in this section include:

- read
- write

NOTE: In the timing diagrams, the two closely spaced, wavy vertical lines, such as those shown in Figure 15-4, indicate one or more identical cycles which are not illustrated due to space constraints.

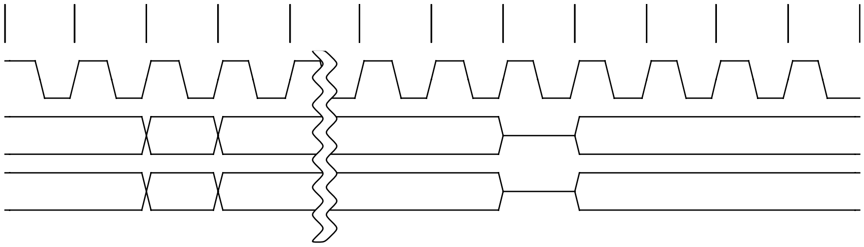


Figure 15-4   Symbol for Undocumented Cycles

15.5.1 Processor Read Request Protocol

The following sequence describes the protocol for doubleword, partial doubleword, word, partial word, and non-secondary cache mode processor read requests. The secondary cache block read request protocol is described later in this section. The numbered steps below correspond to Figure 15-5.

1. **RdRdy*** is asserted low, indicating the external agent is ready to accept a read request.
2. With the System interface in master state, a processor read request is issued by driving a read command on the **SysCmd** bus and a read address on the **SysAD** bus. The physical address is driven onto **SysAD[35:0]**, and virtual address bits [13:12] are driven onto **SysAD[57:56]**. All other bits are driven to zero.
3. At the same time, the processor asserts **ValidOut*** for one cycle, indicating valid data is present on the **SysCmd** and the **SysAD** buses.

   NOTE: Only one processor read request can be pending at a time.
4. The processor makes an uncompelled change to slave state during the issue cycle of the read request. The external agent must not assert the signal **ExtRqst*** for the purposes of returning a read response, but rather must wait for the uncompelled change to slave state. The signal **ExtRqst*** can be asserted before or during a read response to perform an external request other than a read response.
5. The processor releases the **SysCmd** and the **SysAD** buses one **SysClock** after the assertion of **Release***.
6. The external agent drives the **SysCmd** and the **SysAD** buses within two cycles after the assertion of **Release***.

Once in slave state the external agent can return the requested data through a read response. The read response can return the requested data or, if the requested data could not be successfully retrieved, an indication that the returned data is erroneous. If the returned data is erroneous, the processor takes a bus error exception.

Figure 15-5 illustrates a processor read request, coupled with an uncompelled change to slave state, that occurs as the read request is issued.

Timings for the **SysADC** and **SysCmdP** buses are the same as those of the **SysAD** and **SysCmd** buses, respectively.

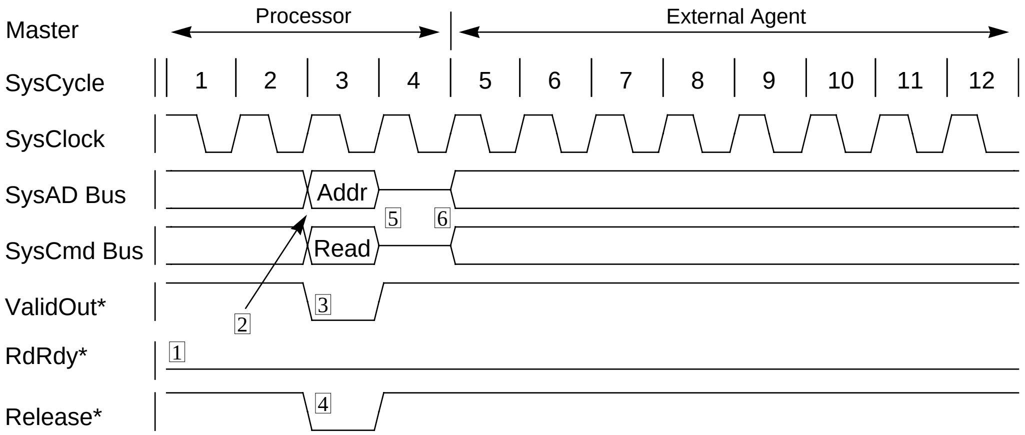


Figure 15-5   Processor Read Request Protocol

Any time a read request has been issued (indicating a read request is pending), the processor will assert **Release*** to perform an uncompelled change to slave state. Once in the slave state the processor will always accept either a read response or an **ExtRqst*** (if a read is pending).

15.5.2 Processor Write Request Protocol

Processor write requests are issued using one of three protocols.

- Doubleword, partial doubleword, word, or partial word writes use a non-block write request protocol.
- Non-secondary cache block writes use a block write request protocol.
- Secondary cache block write request protocol.

Processor non-block write requests are issued with the System interface in master state, as described below in the steps below; Figure 15-6 shows a processor noncoherent non-block write request cycle.

1. **WrRdy*** is asserted low, indicating the external agent is ready to accept a write request.
2. A processor single non-block write request is issued by driving a write command on the **SysCmd** bus and a write address on the **SysAD** bus. The physical address is driven onto **SysAD[35:0]**, and virtual address bits [13:12] are driven onto **SysAD[57:56]**. All other bits are driven to zero.
3. The processor asserts **ValidOut***.
4. The processor drives a data identifier on the **SysCmd** bus and data on the **SysAD** bus.
5. The data identifier associated with the data cycle must contain a last data cycle indication. At the end of the cycle, **ValidOut*** is deasserted.

   NOTE: Timings for the **SysADC** and **SysCmdP** buses are the same as those of the **SysAD** and **SysCmd** buses, respectively.

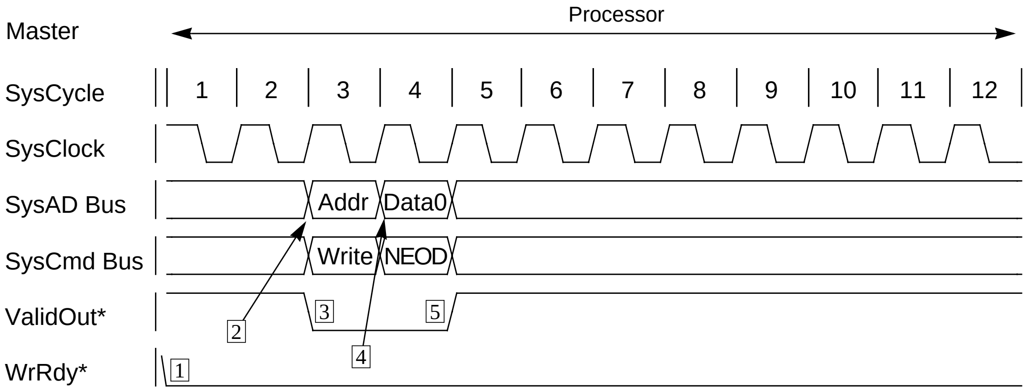


Figure 15-6   Processor Non-Coherent Non-Block Write Request Protocol

Figure 15-7 illustrates a non-secondary cache block write request.

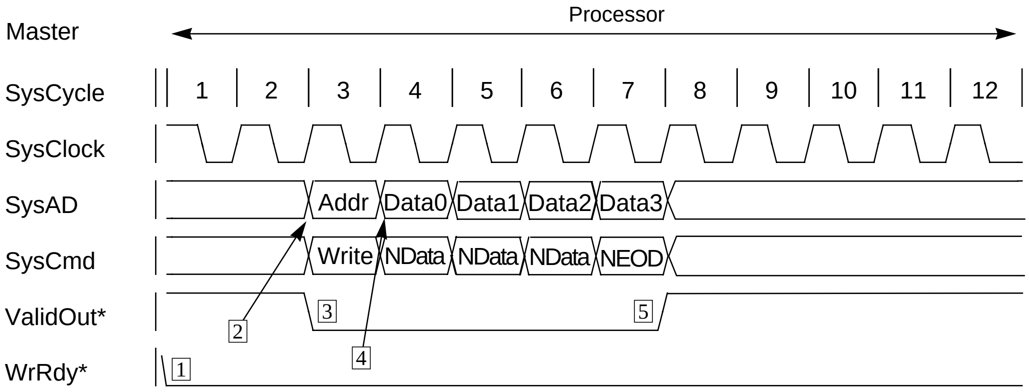


Figure 15-7 Processor Non-Coherent, Non-Secondary Cache Block Write Request

15.5.3 Processor Request Flow Control

The external agent uses **RdRdy*** to control the flow of processor read requests.

Figure 15-8 illustrates this flow control, as described in the steps below.

1. The processor samples the **RdRdy*** signal to determine if the external agent is capable of accepting a read request.
2. Read request is issued to the external agent.
3. The external agent deasserts **RdRdy***, indicating it cannot accept additional read requests.
4. The read request issue is stalled because **RdRdy*** was negated two cycles earlier.
5. Read request is again issued to the external agent.

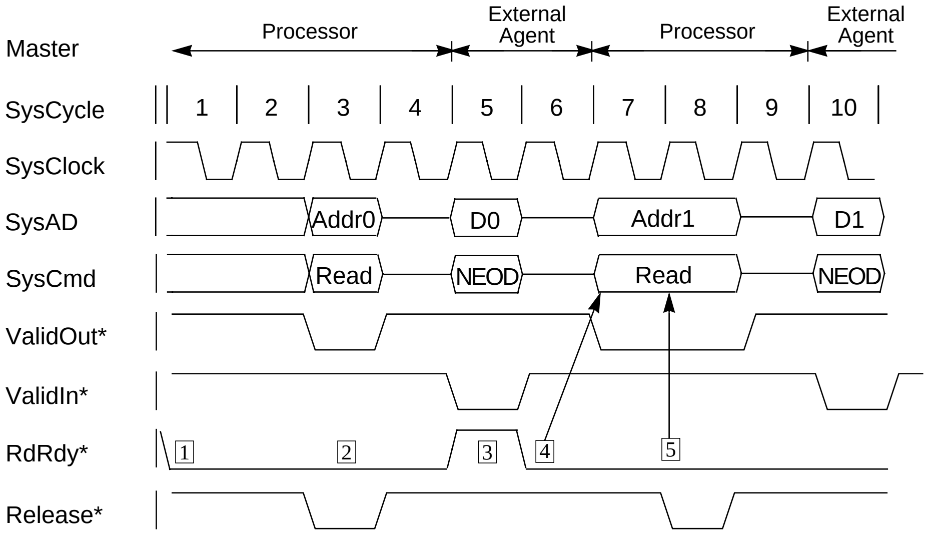


Figure 15-8 Processor Request Flow Control

Figure 15-9 illustrates two processor write requests in which the issue of the second is delayed for the state of **WrRdy***.

1. **WrRdy*** is state low, indicating the external agent is ready to accept a write request.
2. The processor asserts **ValidOut***, a write command on the **SysCmd** bus, and a write address on the **SysAD** bus.
3. The second write request is delayed until the **WrRdy*** signal is again asserted.

4. The processor does not complete the issue of a write request until it issues an address cycle in response to the write request for which the signal **WrRdy*** was asserted two cycles earlier.

   NOTE: Timings for the **SysADC** and **SysCmdP** buses are the same as those of the **SysAD** and **SysCmd** buses, respectively.

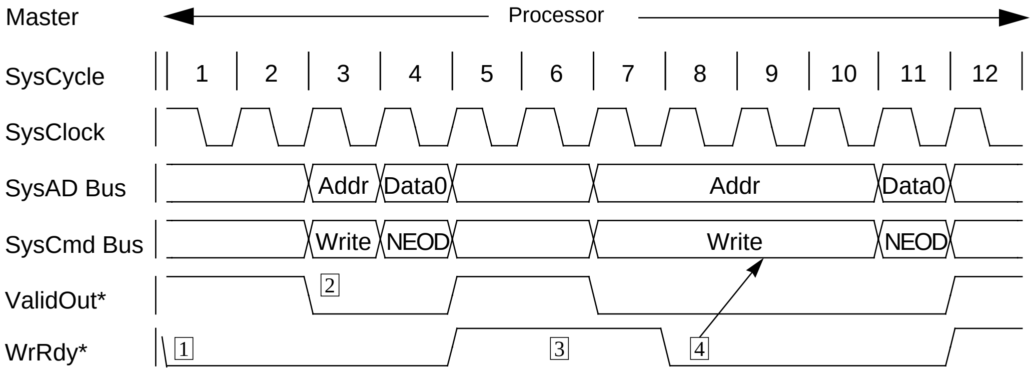


Figure 15-9 Two Processor Write Requests with Second Write Delayed

The V_R5000 processor interface requires that **WrRdy*** be asserted two system cycles prior to the issue of a write cycle. An external agent that negates **WrRdy*** immediately upon receiving the write that fills its buffer will suspend any subsequent writes for four system cycles in V_R4000 non-block write-compatible mode. The processor always inserts at least two unused system cycles after a write address/data pair in order to give the external agent time to suspend the next write.

Figure 15-10 shows back-to-back write cycles in V_R4000-compatible mode.

1. **WrRdy*** is asserted, indicating the processor can issue a write request.
2. **WrRdy*** remains asserted, indicating the external agent can accept another write request.
3. **WrRdy*** deasserts, indicating the external agent cannot accept another write request, stalling the issue of the next write request.

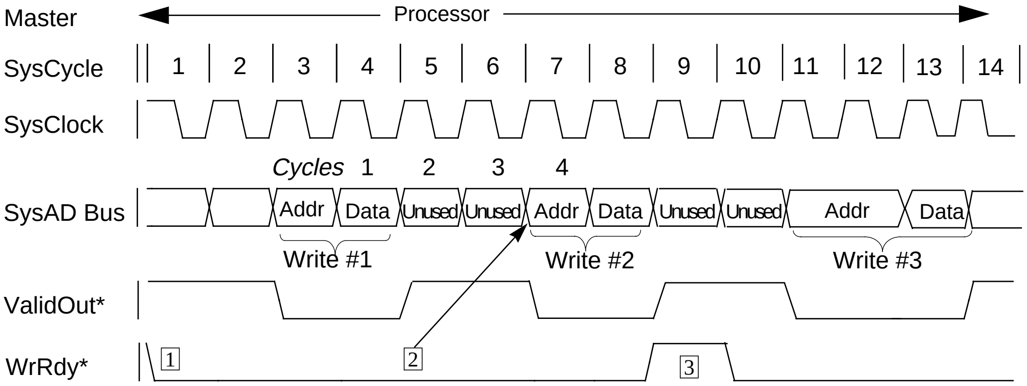


Figure 15-10 V_R4000-Compatible Back-to-Back Write Cycle Timing

An address/data pair every four system cycles is not sufficiently high performance for all applications. For this reason, the V_R5000 processor provides two protocol options that modify the V_R4000 back-to-back write protocol to allow an address/data pair every two system cycles. These two protocols are as follows:

Write Reissue allows **WrRdy*** to be negated during the address cycle and forces the write cycle to be re-issued.

Pipelined Writes leave the sample point of **WrRdy*** unchanged and require that the external agent accept one more write than dictated by the V_R4000 protocol.

The write re-issue protocol is shown in Figure 15-11. Writes issue when **WrRdy*** is asserted both two cycles prior to the address cycle and during the address cycle.

1. **WrRdy*** is asserted, indicating the external agent can accept a write request.
2. **WrRdy*** remains asserted as the write is issued, and the external agent is ready to accept another write request.
3. **WrRdy*** deasserts during the address cycle. This write request is aborted and reissued.
4. **WrRdy*** is asserted, indicating the external agent can accept a write request.
5. **WrRdy*** remains asserted as the write is issued, and the external agent is able to accept another write request.

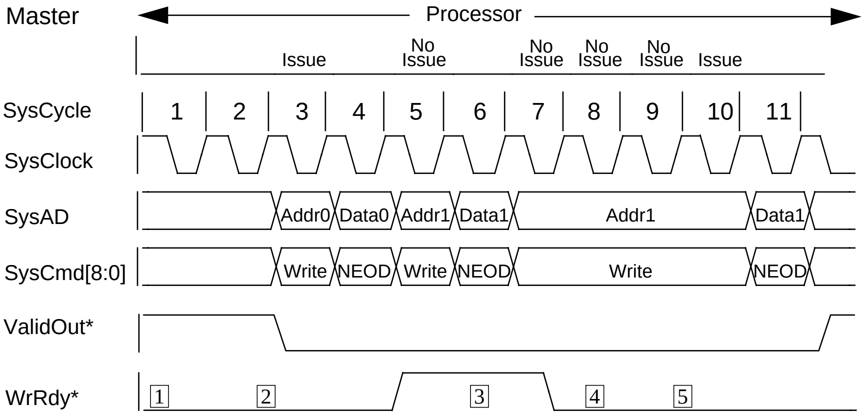


Figure 15-11   Write Reissue

The pipelined write protocol is shown in Figure 15-2. Writes issue when **WrRdy*** is asserted two cycles before the address cycle and the external agent is required to accept one more write after **WrRdy*** is negated.

1. **WrRdy*** is asserted, indicating the external agent can accept a write request.
2. **WrRdy*** remains asserted as the write is issued, and the external agent is able to accept another write request.
3. **WrRdy*** is deasserted, indicating the external agent cannot accept another write request; it does, however, accept this write.
4. **WrRdy*** is asserted, indicating the external agent can accept a write request.

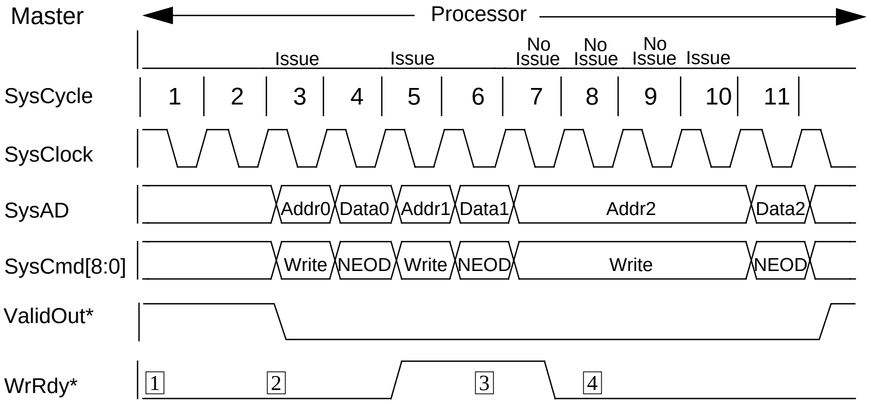


Figure 15-12 Pipelined Writes

15.6 External Request Protocols

External requests can only be issued with the System interface in slave state. An external agent asserts **ExtRqst*** to arbitrate for the System interface, then waits for the processor to release the System interface to slave state by asserting **Release*** before the external agent issues an external request. If the System interface is already in slave state—that is, the processor has previously performed an uncompelled change to slave state—the external agent can begin an external request immediately.

After issuing an external request, the external agent must return the System interface to master state. If the external agent does not have any additional external requests to perform, **ExtRqst*** must be deasserted two cycles after the cycle in which **Release*** was asserted. For a string of external requests, the **ExtRqst*** signal is asserted until the last request cycle, whereupon it is deasserted two cycles after the cycle in which **Release*** was asserted.

The processor continues to handle external requests as long as **ExtRqst*** is asserted; however, the processor cannot release the System interface to slave state for a subsequent external request until it has completed the current request. As long as **ExtRqst*** is asserted, the string of external requests is not interrupted by a processor request.

This section describes the following external request protocols:

- null
- write
- read response

15.6.1 External Arbitration Protocol

System interface arbitration uses the signals **ExtRqst*** and **Release*** as described above. Figure 15-13 is a timing diagram of the arbitration protocol, in which slave and master states are shown.

The arbitration cycle consists of the following steps:

1. The external agent asserts **ExtRqst*** when it wishes to submit an external request.
2. The processor waits until it is ready to handle an external request, whereupon it asserts **Release*** for one cycle.
3. The processor sets the **SysAD** and **SysCmd** buses to tristate.
4. The external agent must wait at least two cycles after the assertion of **Release*** before it drives the **SysAD** and **SysCmd** buses.
5. The external agent negates **ExtRqst*** two cycles after the assertion of **Release***, unless the external agent wishes to perform an additional external request.
6. The external agent sets the **SysAD** and the **SysCmd** buses to tristate at the completion of an external request.

The processor can start issuing a processor request one cycle after the external agent sets the bus to tristate.

NOTE: Timings for the **SysADC** and **SysCmdP** buses are the same as those of the **SysAD** and **SysCmd** buses, respectively.

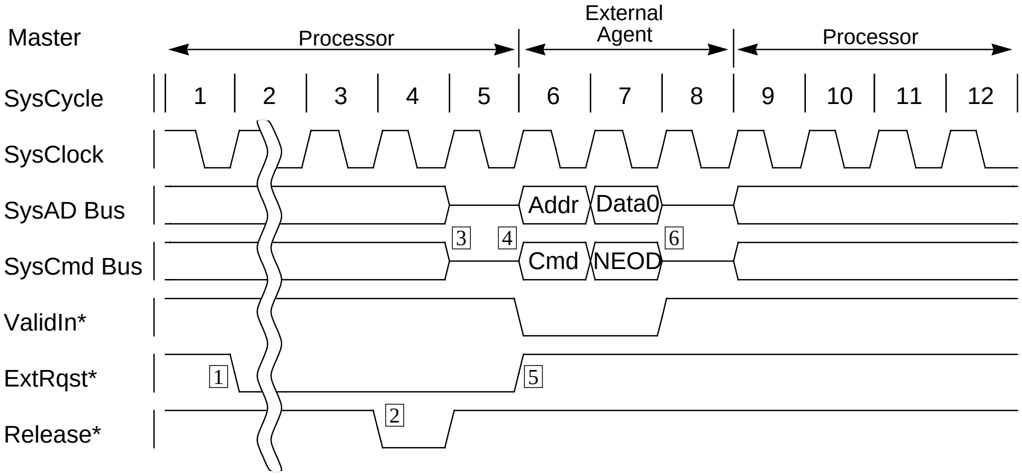


Figure 15-13 Arbitration Protocol for External Requests

15.6.2 External Null Request Protocol

The processor supports a system interface external null request, which returns the System interface to master state from slave state without otherwise affecting the processor.

External null requests require no action from the processor other than to return the System interface to master state.

Figure 15-14 shows a timing diagram of an external null request, which consist of the following steps:

1. The external agent drives a system interface release external null request command on the **SysCmd** bus, and asserts **ValidIn*** for one cycle to return system interface ownership to the processor.
2. The **SysAD** bus is unused (does not contain valid data) during the address cycle associated with an external null request.
3. After the address cycle is issued, the null request is complete.

For a *System interface release external null request*, the external agent releases the **SysCmd** and **SysAD** buses, and expects the System interface to return to the master state.

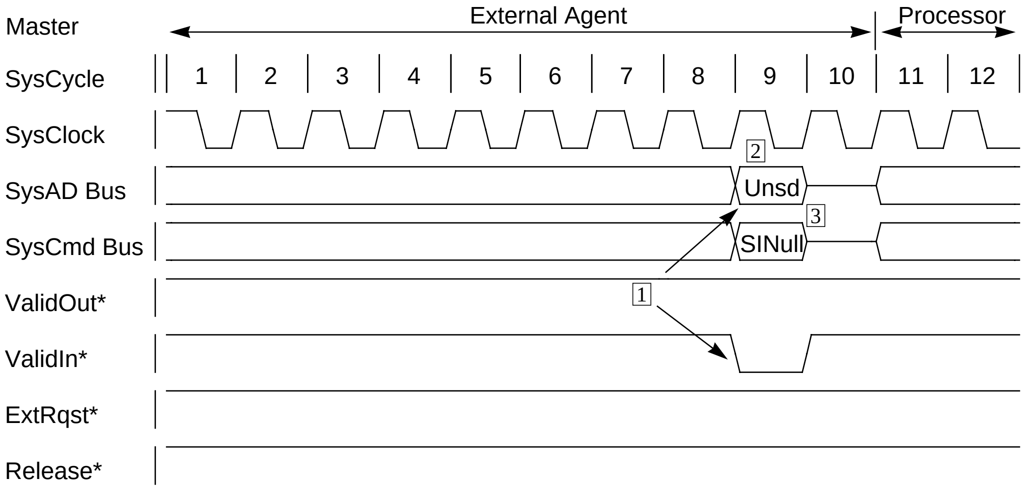


Figure 15-14   System Interface Release External Null Request

15.6.3 External Write Request Protocol

External write requests use a protocol identical to the processor single word write protocol except the **ValidIn*** signal is asserted instead of **ValidOut***. Figure 15-15 shows a timing diagram of an external write request, which consists of the following steps:

1. The external agent asserts **ExtRqst*** to arbitrate for the System interface.
2. The processor releases the System interface to slave state by asserting **Release***.
3. The external agent drives a write command on the **SysCmd** bus, a write address on the **SysAD** bus, and asserts **ValidIn***.
4. The external agent drives a data identifier on the **SysCmd** bus, data on the **SysAD** bus, and asserts **ValidIn***.
5. The data identifier associated with the data cycle must contain a coherent or noncoherent last data cycle indication.
6. After the data cycle is issued, the write request is complete and the external agent sets the **SysCmd** and **SysAD** buses to a tristate, allowing the System interface to return to master state. Timings for the **SysADC** and **SysCmdP** buses are the same as those of the **SysAD** and **SysCmd** buses, respectively.

External write requests are only allowed to write a word of data to the processor. Processor behavior in response to an external write request for any data element other than a word is undefined.

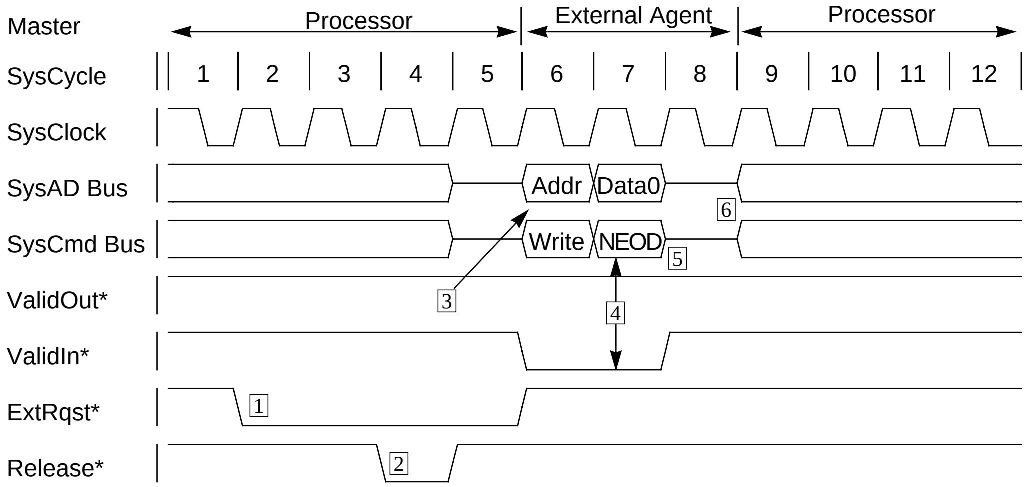


Figure 15-15 External Write Request, with System Interface Initially a Bus Master

15.6.4 Read Response Protocol

An external agent must return data to the processor in response to a processor read request by using a read response protocol. A read response protocol consists of the following steps:

1. The external agent waits for the processor to perform an uncompelled change to slave state.
2. The processor returns the data through a single data cycle or a series of data cycles.
3. After the last data cycle is issued, the read response is complete and the external agent sets the **SysCmd** and **SysAD** buses to a tristate.
4. The System interface returns to master state.

   NOTE: The processor always performs an uncompelled change to slave state after issuing a read request.
5. The data identifier for data cycles must indicate the fact that this data is *response data*.
6. The data identifier associated with the last data cycle must contain a *last data cycle* indication.

For read responses to non-coherent block read requests, the response data does not need to identify the initial cache state. The cache state is automatically assigned as dirty exclusive by the processor.

The data identifier associated with a data cycle can indicate that the data transmitted during that cycle is erroneous; however, an external agent must return a data block of the correct size regardless of the fact that the data may be in error.

The processor only checks the error bit for the first doubleword of the block. The remaining error bits for the block are ignored.

Read response data must only be delivered to the processor when a processor read request is pending. The behavior of the processor is undefined when a read response is presented to it and there is no processor read pending.

Figure 15-16 illustrates a processor word read request followed by a word read response. Figure 15-17 illustrates a read response for a processor block read with the System interface already in slave state.

> **NOTE:** Timings for the **SysADC** and **SysCmdP** buses are the same as those of the **SysAD** and **SysCmd** buses, respectively.

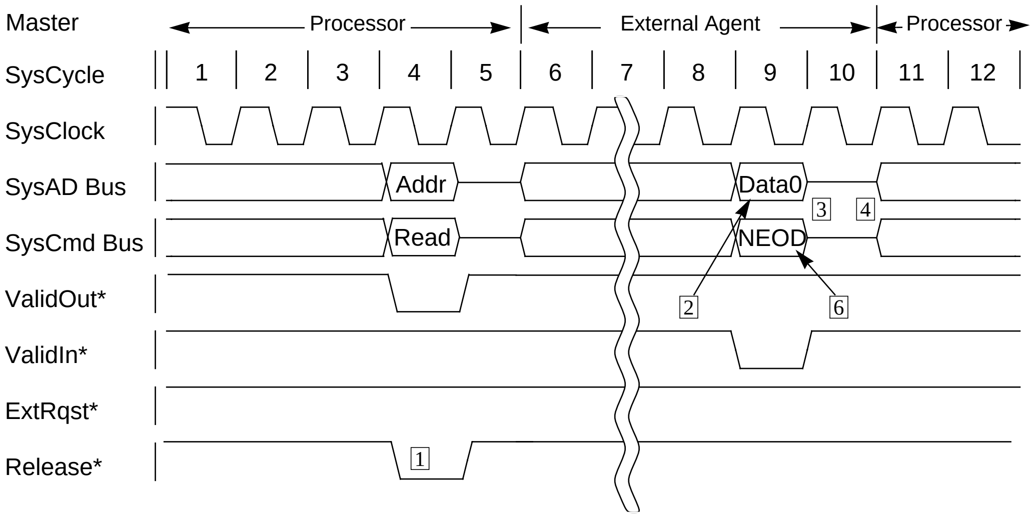


Figure 15-16   Processor Word Read Request, Followed by a Word Read Response

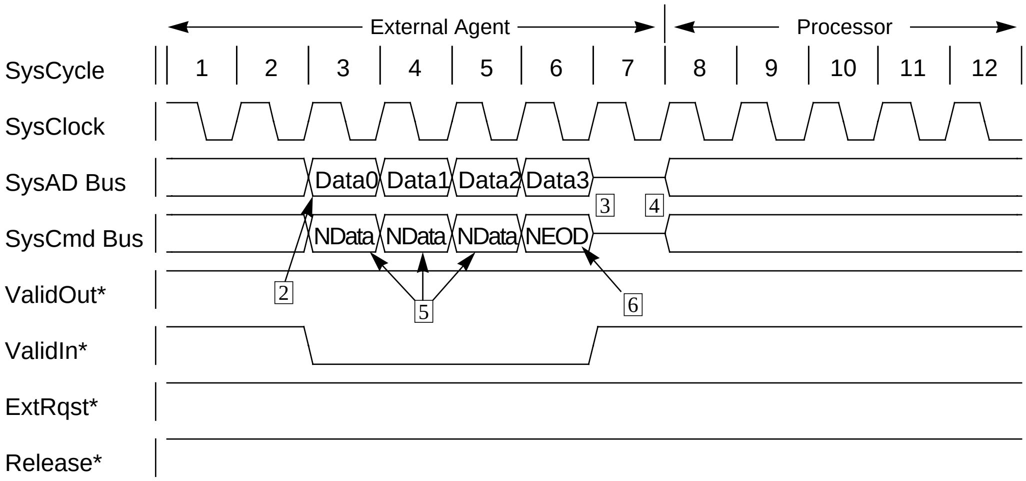


Figure 15-17 Block Read Response, System Interface already in Slave State

15.7 SysADC[7:0] Protocol

The following rules apply to the use of **SysADC[7:0]** during a block read response.

- Data is checked on only the first doubleword of the transfer. If data is erroneous (**SysCmd[5]**=1), the primary and secondary cache lines are invalidated and a bus error exception is generated.
- A parity error on the first doubleword will be detected as it issused and will cause a cache parity error exception. The cache line will be valid. Parity errors in subsequent doubles will be detected if they are used.
- On the following three doublewords; The data erroneous bit is ignored. Parity for each of the three doublewords is written into the cache, but is not checked until the data is referenced.
- Any read that will fill the secondary cache must receive correct parity for all 4 doublewords (**SysCmd[4]**=0) for data going to the secondary cache.
- For a secondary cache mode read hit cycle; Data erroneous is implicitly OFF. Check parity is implicitly ON, indicating that the secondary cache must implement the **SysADC** bits.

- If a memory error occurs during a block read operation, the **SysADC** bits should be forced to bad parity for all bytes affected by the memory error during the read response. Since the processor performs an early-restart on data cache line fills, setting the **SysCmd[5]** bit on any transfer other than the first doubleword does not cause a bus error. Forcing bad parity will generate a cache error if any of the remaining three doublewords of the transfer are referenced.

15.8 Data Rate Control

The System interface supports a maximum data rate of one doubleword per cycle. The rate at which data is delivered to the processor can be determined by the external agent—for example, the external agent can drive data and assert **ValidIn*** every *n* cycles, instead of every cycle. An external agent can deliver data at any rate it chooses.

The processor only accepts cycles as valid when **ValidIn*** is asserted and the **SysCmd** bus contains a data identifier; thereafter, the processor continues to accept data until it receives the data word tagged as the last one.

Figure 15-18 shows a read response in which data is provided to the processor at a rate of two doublewords every three cycles using the data pattern **DDx**.

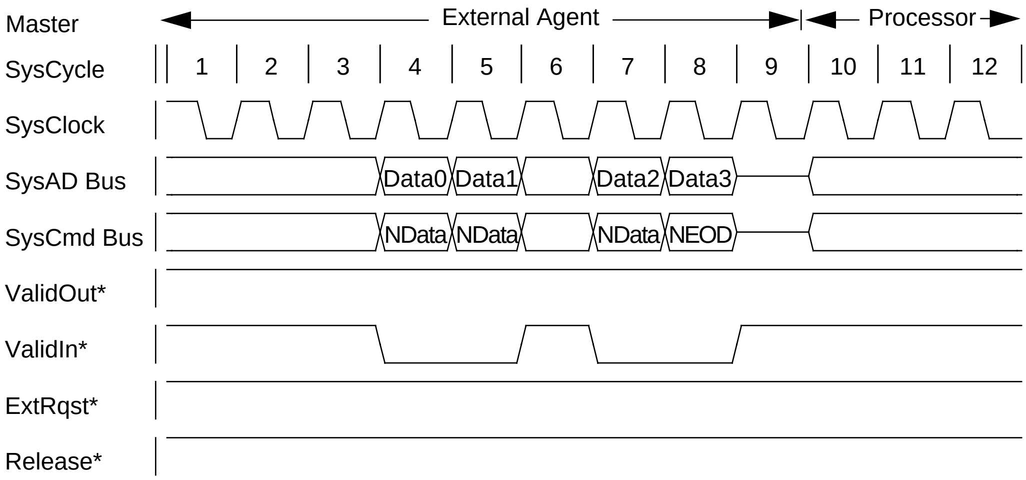


Figure 15-18 Read Response, Reduced Data Rate, System Interface in Slave State

15.9 Data Transfer Patterns

A data pattern is a sequence of letters indicating the *data* and *unused* cycles that repeat to provide the appropriate data rate. For example, the data pattern **DDxx** specifies a repeatable data rate of two doublewords every four cycles, with the last two cycles unused. Table 15-2 lists the maximum processor data rate for each of the possible block write modes that may be specified at boot time.

Table 15-2   Transmit Data Rates and Patterns

Maximum Data Rate	Data Pattern
1 Double/1 SysClock Cycle	DDDD
2 Doubles/3 SysClock Cycles	DDxDDx
1 Double/2 SysClock Cycles	DDxxDDxx
1 Double/2 SysClock Cycles	DxDxDxDx
2 Doubles/5 SysClock Cycles	DDxxxDDxxx
1 Double/3 SysClock Cycles	DDxxxxDDxxxx
1 Double/3 SysClock Cycles	DxxDxxDxxDxx
1 Double/4 SysClock Cycles	DDxxxxxxDDxxxxxx
1 Double/4 SysClock Cycles	DxxxDxxxDxxxDxxx

In Table 15-2, data patterns are specified using the letters **D** and **x**; **D** indicates a data cycle and **x** indicates an unused cycle.

15.10 Independent Transmissions on the SysAD Bus

In most applications, the **SysAD** bus is a point-to-point connection, running from the processor to a bidirectional registered transceiver residing in an external agent. For these applications, the **SysAD** bus has only two possible drivers, the processor or the external agent.

Certain applications may require connection of additional drivers and receivers to the **SysAD** bus, to allow transmissions over the **SysAD** bus that the processor is not involved in. These are called *independent transmissions*. To effect an independent transmission, the external agent must coordinate control of the **SysAD** bus by using arbitration handshake signals and external null requests.

An independent transmission on the **SysAD** bus follows this procedure:

1. The external agent requests mastership of the **SysAD** bus, to issue an external request.
2. The processor releases the System interface to slave state.
3. The external agent then allows the independent transmission to take place on the **SysAD** bus, making sure that **ValidIn*** is not asserted while the transmission is occurring.
4. When the transmission is complete, the external agent must issue a *System interface release external null request* to return the System interface to master state.

15.11 System Interface Endianness

The endianness of the System interface is programmed at boot time through the boot-time mode control interface and the **BigEndian** pin. The **BigEndian** pin allows the system to change the processor addressing mode without rewriting the mode ROM. If endianness is to be specified via the **BigEndian** pin, program mode ROM bit 8 to zero. If endianness is to be specified by the mode ROM, ground the **BigEndian** pin. Software cannot change the endianness of the System interface and the external system; software can set the reverse endian bit to reverse the interpretation of endianness inside the processor, but the endianness of the System interface remains unchanged.

15.12 System Interface Cycle Time

The processor specifies minimum and maximum cycle counts for various processor transactions and for the processor response time to external requests. Processor requests themselves are constrained by the System interface request protocol, and request cycle counts can be determined by examining the protocol. The following System interface interactions can vary within minimum and maximum cycle counts:

- waiting period for the processor to release the System interface to slave state in response to an external request (*release latency*)
- response time for an external request that requires a response (*external response latency*).

The remainder of this section describes and tabulates the minimum and maximum cycle counts for these System interface interactions.

15.13 Release Latency

Release latency is generally defined as the number of cycles the processor can wait to release the System interface to slave state for an external request. When no processor requests are in progress, internal activity can cause the processor to wait some number of cycles before releasing the System interface. Release latency is therefore more specifically defined as the number of cycles that occur between the assertion of **ExtRqst*** and the assertion of **Release***.

There are three categories of release latency:

- Category 1: when the external request signal is asserted two cycles before the last cycle of a processor request.
- Category 2: when the external request signal is not asserted during a processor request or is asserted during the last cycle of a processor request.
- Category 3: when the processor makes an uncompelled change to slave state.

Table 15-3 summarizes the minimum and maximum release latencies for requests that fall into categories 1, 2, and 3. Note that the maximum and minimum cycle count values are subject to change.

Table 15-3 Release Latency for External Requests

Category	Minimum PCycles	Maximum PCycles
1	4	6
2	4	24
3	0	0

15.14 System Interface Commands/Data Identifiers

System interface commands specify the nature and attributes of any System interface request; this specification is made during the address cycle for the request. System interface data identifiers specify the attributes of data transmitted during a System interface data cycle.

The following sections describe the syntax, that is, the bitwise encoding of System interface commands and data identifiers.

Reserved bits and reserved fields in the command or data identifier should be set to 1 for System interface commands and data identifiers associated with external requests. For System interface commands and data identifiers associated with processor requests, reserved bits and reserved fields in the command and data identifier are undefined.

15.14.1 Command and Data Identifier Syntax

System interface commands and data identifiers are encoded in 9 bits and are transmitted on the **SysCmd** bus from the processor to an external agent, or from an external agent to the processor, during address and data cycles. Bit 8 (the most-significant bit) of the **SysCmd** bus determines whether the current content of the **SysCmd** bus is a command or a data identifier and, therefore, whether the current cycle is an address cycle or a data cycle. For System interface commands, **SysCmd(8)** must be set to 0. For System interface data identifiers, **SysCmd(8)** must be set to 1.

15.14.2 System Interface Command Syntax

This section describes the **SysCmd** bus encoding for System interface commands. Figure 15-19 shows a common encoding used for all System interface commands.

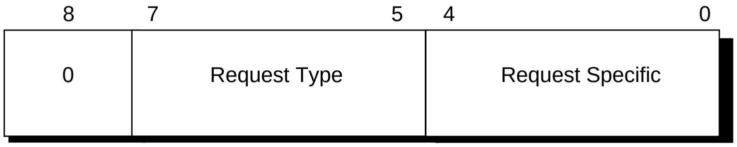


Figure 15-19 System Interface Command Syntax Bit Definition

SysCmd(8) must be set to 0 for all System interface commands.

SysCmd(7:5) specify the System interface request type which may be read, write, or null. Table 15-4 shows the types of requests encoded by the **SysCmd(7:5)** bits.

Table 15-4 Encoding of SysCmd(7:5) *for System Interface Commands*

SysCmd(7:5)	Command
0	Read Request
1	Reserved
2	Write Request
3	Null Request
4-7	Reserved

SysCmd(4:0) are specific to each type of request and are defined in each of the following sections.

(1) Read Requests

Figure 15-20 shows the format of a **SysCmd** read request.

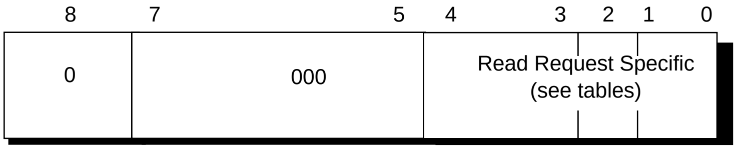


Figure 15-20 Read Request SysCmd *Bus Bit Definition*

Tables 15-5 through 15-7 list the encodings of **SysCmd(4:0)** for read requests.

Table 15-5   Encoding of SysCmd(4:3) *for Read Requests*

SysCmd(4:3)	Read Attributes
0-1	Reserved
2	Noncoherent block read
3	Doubleword, partial doubleword, word, or partial word

Table 15-6   Encoding of SysCmd(1:0) *for Block Read Request*

SysCmd(1:0)	Read Block Size
0	Reserved
1	8 words
2-3	Reserved

Table 15-7   Read Request Data Size Encoding of SysCmd(2:0)

SysCmd(2:0)	Read Data Size
0	1 byte valid (Byte)
1	2 bytes valid (Halfword)
2	3 bytes valid (Tribyte)
3	4 bytes valid (Word)
4	5 bytes valid (Quintibyte)
5	6 bytes valid (Sextibyte)
6	7 bytes valid (Septibyte)
7	8 bytes valid (Doubleword)

(2) Write Requests

Figure 15-21 shows the format of a **SysCmd** write request.

Table 15-8 lists the write attributes encoded in bits **SysCmd(4:3)**. Table 15-9 lists the block write replacement attributes encoded in bits **SysCmd(2:0)**. Table 15-10 lists the write request bit encodings in **SysCmd(2:0)**.

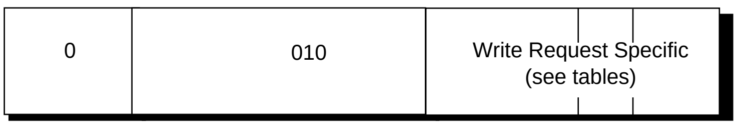


Figure 15-21   Write Request SysCmd *Bus Bit Definition*

Table 15-8   Write Request Encoding of SysCmd(4:3)

SysCmd(4:3)	Write Attributes
0	Reserved
1	Reserved
2	Block write
3	Doubleword, partial doubleword, word, or partial word

Table 15-9   Block Write Request Encoding of SysCmd(2:0)

SysCmd(2)	Reserved
SysCmd(1:0)	**Write Block Size**
0	Reserved
1	8 words
2-3	Reserved

Table 15-10   Write Request Data Size Encoding of SysCmd(2:0)

SysCmd(2:0)	Write Data Size
0	1 byte valid (Byte)
1	2 bytes valid (Halfword)
2	3 bytes valid (Tribyte)
3	4 bytes valid (Word)
4	5 bytes valid (Quintibyte)
5	6 bytes valid (Sextibyte)
6	7 bytes valid (Septibyte)
7	8 bytes valid (Doubleword)

(3) Null Requests

Figure 15-22 shows the format of a **SysCmd** null request.

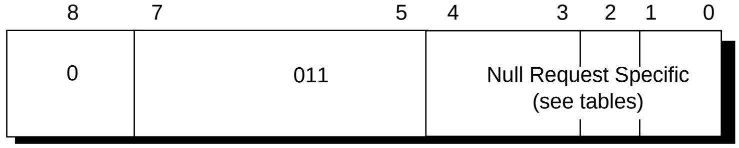


Figure 15-22   Null Request SysCmd *Bus Bit Definition*

System interface release external null requests use the null request command. Table 15-11 lists the encodings of **SysCmd(4:3)** for external null requests.

SysCmd(2:0) are reserved for null requests.

Table 15-11   External Null Request Encoding of SysCmd(4:3)

SysCmd(4:3)	Null Attributes
0	System Interface release
1-3	Reserved

15.14.3 System Interface Data Identifier Syntax

This section defines the encoding of the **SysCmd** bus for System interface data identifiers. Figure 15-23 shows a common encoding used for all System interface data identifiers.

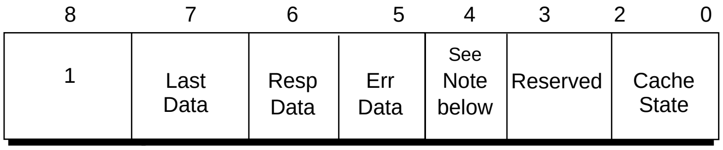


Figure 15-23   Data Identifier SysCmd *Bus Bit Definition*

SysCmd(8) must be set to 1 for all System interface data identifiers.

> **NOTE: SysCmd(4)** is reserved for processor data identifier.   In an external data identifier, **SysCmd(4)** indicates whether or not to check the data and check bits for error.

(1) Noncoherent Data

Noncoherent data is defined as follows:

- data that is associated with processor block write requests and processor doubleword, partial doubleword, word, or partial word write requests
- data that is returned in response to a processor noncoherent block read request or a processor doubleword, partial doubleword, word, or partial word read request
- data that is associated with external write requests
- data that is returned in response to an external read request

(2) Data Identifier Bit Definitions

SysCmd(7) marks the last data element and **SysCmd(6)** indicates whether or not the data is response data, for both processor and external coherent and noncoherent data identifiers. Response data is data returned in response to a read request.

SysCmd(5) indicates whether or not the data element is error free. Erroneous data contains an uncorrectable error and is returned to the processor, forcing a bus error. In the case of a block response, the entire line must be delivered to the processor no matter how minimal the error. Note that the processor only checks **SysCmd[5]** during the first doubleword of a block read response.

SysCmd(4) indicates to the processor whether to check the data and check bits for this data element, for both coherent and noncoherent external data identifiers.

SysCmd(3) is reserved for external data identifiers.

SysCmd(4:3) are reserved for noncoherent processor data identifiers.

SysCmd(2:0) are reserved for non-coherent data identifiers.

Table 15-12 lists the encodings of **SysCmd(7:3)** for processor data identifiers. Table 15-13 lists the encodings of **SysCmd(7:3)** for external data identifiers.

Table 15-12 Processor Data Identifier Encoding of SysCmd(7:3)

SysCmd(7)	**Last Data Element Indication**
0	Last data element
1	Not the last data element
SysCmd(6)	**Response Data Indication**
0	Data is response data
1	Data is not response data
SysCmd(5)	**Good Data Indication**
0	Data is error free
1	Data is erroneous
SysCmd(4)	**Data Parity Checking Enable**
0	Check data parity
1	Ignore data parity
SysCmd(3)	Reserved

Table 15-13 External Data Identifier Encoding of SysCmd(7:3)

SysCmd(7)	**Last Data Element Indication**
0	Last data element
1	Not the last data element
SysCmd(6)	**Response Data Indication**
0	Data is response data
1	Data is not response data
SysCmd*(5)*	**Good Data Indication**
0	Data is error free
1	Data is erroneous
SysCmd(4)	**Data Checking Enable**
0	Check the data and check bits
1	Do not check the data and check bits
SysCmd(3)	Reserved

15.15 System Interface Addresses

System interface addresses are full 36-bit physical addresses presented on the least-significant 36 bits (bits 35 through 0) of the **SysAD** bus during address cycles. Virtual address bits **VA[13:12]** appear on **SysAD[57:56]**. The remaining bits of the **SysAD** bus are unused during address cycles.

15.15.1 Addressing Conventions

Addresses associated with doubleword, partial doubleword, word, or partial word transactions and update requests, are aligned for the size of the data element. The system uses the following address conventions:

- Addresses associated with block requests are aligned to double-word boundaries; that is, the low-order 3 bits of address are 0.
  However, when the Branch instruction is used to jump to a word boundary (**SysAD[2:0]**=100) which is not a double-word boundary (**SysAD[2:0]**=000) of the non-cache area, LOW is not output for the low-order 3rd bit of the address that is output to **SysAD** for instruction fetching; instead, **SysAD[2:0]**=100 is output.

In other words, when a jump to the non-cache area with a low-order byte address of 0x4 and 0xC has occurred, double-word access occurs but the low-order bytes of the output address remain as 0x4 and 0xC. Immediately after such a branch, the CPU uses the word data whose byte addresses are indicated by 0x4 and 0xC.

- Doubleword requests set the low-order 3 bits of address to 0.
- Word requests set the low-order 2 bits of address to 0.
- Halfword requests set the low-order bit of address to 0.
- Byte, tribyte, quintibyte, sextibyte, and septibyte requests use the byte address.

15.15.2 Subblock Ordering

The order in which data is returned in response to a processor block read request is subblock ordering. In subblock ordering, the processor delivers the address of the requested doubleword within the block. An external agent must return the block of data using subblock ordering, starting with the addressed doubleword.

For block write requests, the processor always delivers the address of the doubleword at the beginning of the block; the processor delivers data beginning with the doubleword at the beginning of the block and progresses sequentially through the doublewords that form the block.

During data cycles, the valid byte lines depend upon the position of the data with respect to the aligned doubleword (this may be a byte, halfword, tribyte, quadbyte/word, quintibyte, sextibyte, septibyte, or an octalbyte/doubleword). For example, in little-endian mode, on a byte request where the address modulo 8 is 0, **SysAD(7:0)** are valid during the data cycles. Table 15-14 lists the byte lanes used for partial word transfers for both big and little endian.

Table 15-14 Partial Word Transfer Byte Lane Usage

# Bytes SysCmd[2:0]	**Address Mod 8**	**SysAD byte lanes used (Big Endian)**							
		63:56	**55:48**	**47:40**	**39:32**	**31:24**	**23:16**	**15:8**	**7:0**
1 (000)	0	X							
	1		X						
	2			X					
	3				X				
	4					X			
	5						X		
	6							X	
	7								X
2 (001)	0	X	X						
	2			X	X				
	4					X	X		
	6							X	X
3 (010)	0	X	X	X					
	1		X	X	X				
	4					X	X	X	
	5						X	X	X
4 (011)	0	X	X	X	X				
	4					X	X	X	X
5 (100)	0	X	X	X	X	X			
	3				X	X	X	X	X
6 (101)	0	X	X	X	X	X	X		
	2			X	X	X	X	X	X
7 (110)	0	X	X	X	X	X	X	X	
	1		X	X	X	X	X	X	X
8 (111)	0	X	X	X	X	X	X	X	X
		7:0	15:8	23:16	31:24	39:32	47:40	55:48	63:56
		SysAD byte lanes used (Little Endian)							

15.15.3 Processor Internal Address Map

External reads and writes provide access to processor internal resources that may be of interest to an external agent. The processor decodes bits **SysAD(6:4)** of the address associated with an external read or write request to determine which processor internal resource is the target. However, the processor does not contain any resources that are *readable* through an external read request. Therefore, in response to an external read request the processor returns undefined data and a data identifier with its *Erroneous Data* bit, **SysCmd(5)**, set. The *Interrupt* register is the only processor internal resource available for *write* access by an external request. The *Interrupt* register is accessed by an external write request with an address of 000_2 on bits 6:4 of the **SysAD** bus.

15.16 Error Checking

15.16.1 Parity Error Checking

The V_R5000 processor uses only parity (error detection only).

Parity is the simplest error detection scheme. By appending a bit to the end of an item of data—called a *parity bit*—single bit errors can be detected; however, these errors cannot be corrected.

There are two types of parity:

- **Odd Parity** adds 1 to any even number of 1s in the data, making the total number of 1s odd (including the parity bit).
- **Even Parity** adds 1 to any odd number of 1s in the data, making the total number of 1s even (including the parity bit).

Odd and even parity are shown in the example below:

Data(3:0)	Odd Parity Bit	Even Parity Bit
0 0 1 0	0	1

The example above shows a single bit in **Data(3:0)** with a value of 1; this bit is **Data(1)**.

- In even parity, the parity bit is set to 1. This makes 2 (an even number) the total number of bits with a value of 1.
- Odd parity makes the parity bit a 0 to keep the total number of 1-value bits an odd number—in the case shown above, the single bit **Data(1)**.

The example below shows odd and even parity bits for various data values:

Data(3:0)	Odd Parity Bit	Even Parity Bit
0 1 1 0	1	0
0 0 0 0	1	0
1 1 1 1	1	0
1 1 0 1	0	1

Parity allows single-bit error detection, but it does not indicate which bit is in error—for example, suppose an odd-parity value of 00011 arrives. The last bit is the parity bit, and since odd parity demands an odd number (1,3,5) of 1s, this data is in error: it has an even number of 1s. However it is impossible to tell *which* bit is in error.

15.16.2 Error Checking Operation

The processor verifies data correctness by using parity as it passes data from the System interface to/from the primary caches.

(1) System Interface

The processor generates correct check bits for doubleword, word, or partial-word data transmitted to the System interface. As it checks for data correctness, the processor passes data check bits from the primary cache, directly without changing the bits, to the System interface.

The processor does not check data received from the System interface for external writes. By setting the **SysCmd[4]** bit in the data identifier, it is possible to prevent the processor from checking read response data from the System interface.

The processor does not check addresses received from the System interface and does not generate check bits for addresses transmitted to the System interface.

The processor does not contain a data corrector; instead, the processor takes a cache error exception when it detects an error based on data check bits. Software is responsible for error handling.

(2) System Interface Command Bus

In the V_R5000 processor, the System interface command bus has a single parity bit, **SysCmdP**, that provides even parity over the 9 bits of this bus. The **SysCmdP** parity bit is not generated when the system interface is in master state and is not checked when the System interface is in slave state. This signal is defined to maintain V_R4000 compatibility and is not functional in the V_R5000.

(3) Summary of Error Checking Operations

Error checking operations are summarized in Table 15-15 and 15-16.

Table 15-15 Error Checking Operation for Internal Transactions

Bus	Uncached Load	Uncached Store	Primary Cache Load from System Interface	Primary Cache Write to System Interface	Cache Instruction
Processor Data	From system	Not checked	From system interface unchanged	Checked; Trap on error	Check on cache write-back; Trap on error
System Address, Command, and Check bits; Transmit	Not Generated	Not Generated	Not Generated	Not Generated	Not Generated
System Address, Command, and Check Bits; Receive	Not Checked	Not Checked	Not Checked	Not Checked	Not Checked
System Interface Data	Checked, Trap on error	From Processor	Checked on requested doubleword, Trap on error	From primary cache	From primary cache
System Interface Data Check Bits	Checked, Trap on error	Generated	Checked on requested doubleword, Trap on error	From primary cache	From primary cache

Table 15-16   Error Checking Operation for External Transactions

Bus	External Write
Processor Data	NA
System Address, Command, and Check bits; Transmit	NA
System Address, Command, and Check Bits; Receive	Not Checked
System Interface Data	Not Checked
System Interface Data Check Bits	Not Checked

Chapter 16 Secondary Cache Interface

The V_R5000 processor supports an external secondary cache by providing an internal secondary cache controller with a dedicated secondary cache port.

16.1 Secondary Cache Transactions

For processors configured with a secondary cache, the secondary cache is a special form of external agent that is jointly controlled by both the processor and the external agent. Figure Figure 16-1 illustrates a processor request to the secondary cache and external agent.

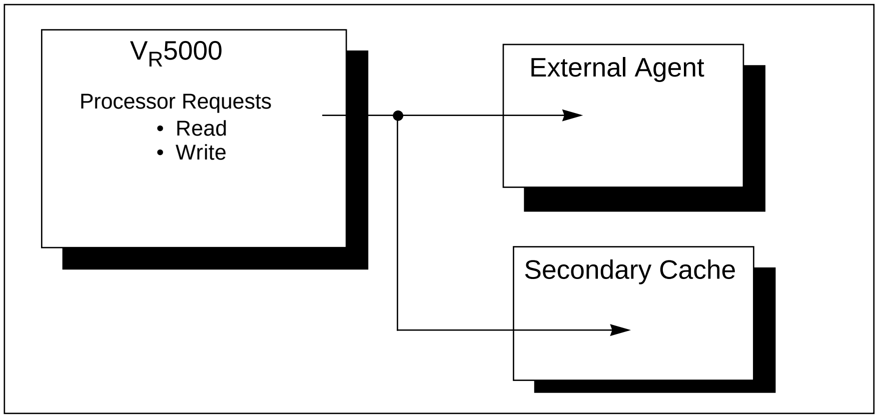


Figure 16-1 Processor Requests to Secondary Cache and External Agent

16.1.1 Secondary Cache Probe, Invalidate, and Clear

For secondary cache invalidate, clear, and probe operations, the secondary cache is controlled by the processor and the external agent is not involved in these operations. Issuance of secondary cache invalidate, clear, and probe operations is not flow-controlled and proceeds at the maximum data rate. Figures 16-2 and 16-3 shows the secondary cache invalidate and tag probe operations.

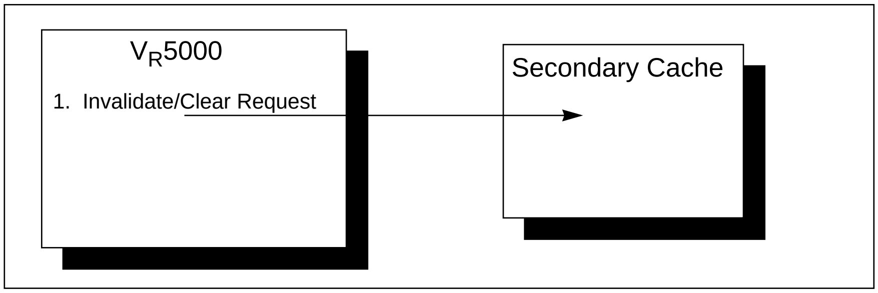


Figure 16-2 Secondary Cache Invalidate and Clear

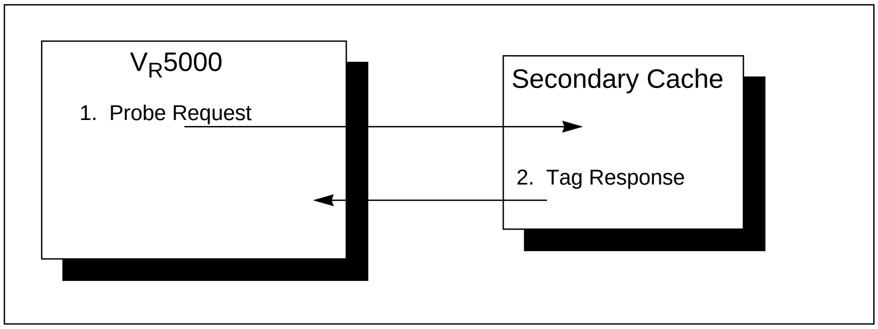


Figure 16-3 Secondary Cache Tag Probe

16.1.2 Secondary Cache Write

For secondary cache write-through, the processor issues a block write operation that is directed to both the secondary cache and the external agent. Issuance of secondary cache writes is controlled by the normal **WrRdy*** flow control mechanism. Secondary cache write data transfers proceed at the data transfer rate specified in the Mode ROM for block writes. Figure 16-4 illustrates a secondary cache write operation.

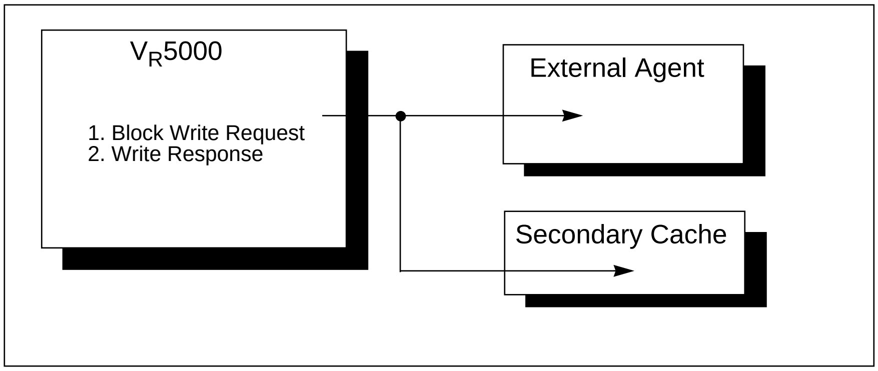


Figure 16-4 Secondary Cache Write Through

16.1.3 Secondary Cache Read

For secondary cache reads, the processor issues a block read speculatively to both the secondary cache and the external agent.

- If the block is present in the secondary cache, the secondary cache provides the read response and the block read to the external agent is aborted.
- If the block is not present in the secondary cache, the secondary cache read is aborted and the external agent provides the read response to both the secondary cache and the processor.

Figures 16-5 and 16-6 shows a secondary cache read hit and miss respectively.

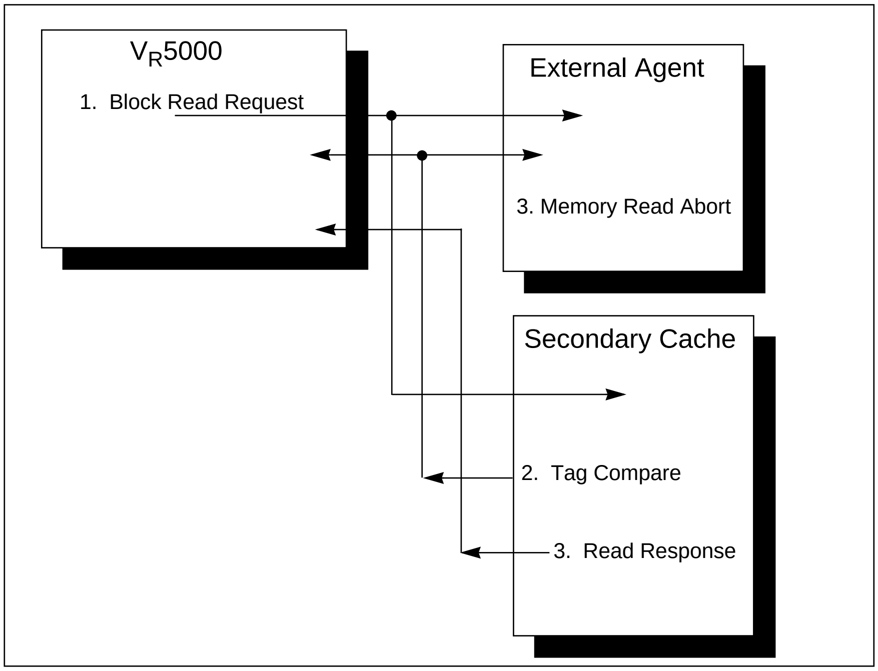


Figure 16-5 Secondary Cache Read Hit

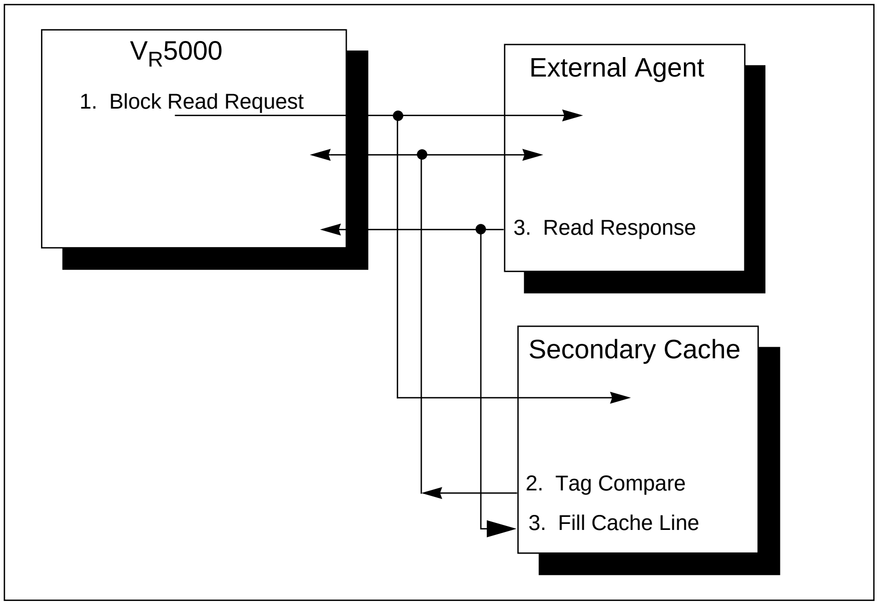


Figure 16-6 Secondary Cache Read Miss

Issuance of the secondary cache read is controlled by the normal **RdRdy*** flow control mechanism. Secondary cache read responses always proceed at the maximum data transfer rate. External agent read responses to the secondary cache proceed at the data transfer rate generated by the external agent.

16.2 Secondary Cache Read Protocol

There are three possible scenarios which can occur on a secondary cache access.

1) Secondary cache read hit

2) Secondary cache miss

3) Secondary cache miss with bus error

16.2.1 Secondary Cache Read Hit

Figure 16-7 shows the secondary cache read hit protocol. When a block read request is speculatively issued to both the secondary cache and the external agent, but completed by the secondary cache:

1. The processor issues a block read request and also asserts the **ScTCE***, **ScTDE***, and **ScDCE*** secondary cache control signals. In addition the processor drives the cache index onto **ScLine[15:0]** and the sub-block order doubleword onto **ScWord[1:0]**. Assertion of **ScTCE***, along with **ValidOut*** and **SysCmd**, indicates to the external agent that this is a secondary cache read request. In addition, the assertion of **ScTCE*** initiates a tag RAM probe. The assertion of **ScTDE*** loads the tag portion of the **SysAD** bus into the tag RAM. The **ScValid** signal is asserted to probe for a valid cache tag. The assertion of **ScDCE*** initiates a speculative read of the secondary cache data RAMs.
2. The **ScMatch** signal from the tag RAM is sampled by both the processor and the external agent. Assertion of **ScMatch** indicates a secondary cache tag hit, causing the external agent to abort the memory read. Hence there is no uncompelled change to slave state. The data RAMs now own **SysAD** and supply the first of a 4 doubleword burst in response to the 4-cycle **ScDCE*** burst. The **SysCmd** bus is not driven during the secondary cache read.
3. Ownership of the **SysAD** bus is returned to the processor.

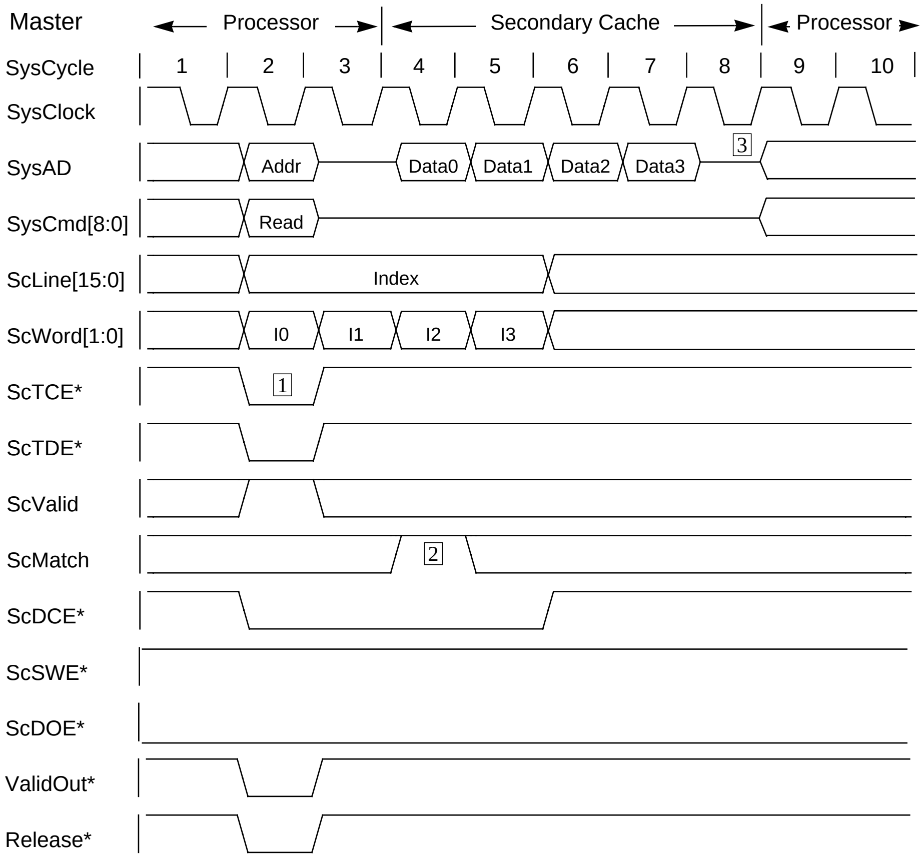


Figure 16-7 Secondary Cache Read Hit

16.2.2 Secondary Cache Read Miss

Figure 16-8 shows the secondary cache read miss protocol when a block read request is speculatively issued to both the secondary cache and the external agent, but is completed by the external agent with a response to both the secondary cache and the processor.

1. The processor issues a block read request and also asserts the **ScTCE***, **ScTDE***, **ScDCE***, and **ScValid** signals and drives the cache index onto **ScLine[15:0]** and **ScWord[1:0]**.
2. The **ScMatch** signal from the tag RAM is sampled by the processor and external agent. Since the signal is negated, indicating a secondary cache miss, the SysAD data from the secondary cache is invalid.

3. The external agent negates **ScDOE*** to tri-state the data RAM outputs, indicating that it will be supplying the read response. The processor tri-states its **ScWord[1:0]** outputs to allow the external agent to drive them during the read response.
4. The processor asserts **ScCWE*** to prepare the data RAMs for a write of the response data.
5. The external agent supplies the first doubleword of the read response and asserts **ValidIn***. The data is both written into the secondary cache and accepted by the processor. **SysCmd** indicates that data is not erroneous. Note that this response may be delayed additional cycles.
6. The processor asserts **ScTCE*** to write the tag value stored in the tag RAM data input register two cycles after **ValidIn*** is asserted.
7. The external agent asserts **ScDOE*** to indicate that it will supply the last doubleword of the read response in the next cycle.
8. The processor negates **ScDCE*** two cycles after the next assertion of **ScDOE*** in order to complete the secondary cache line fill.

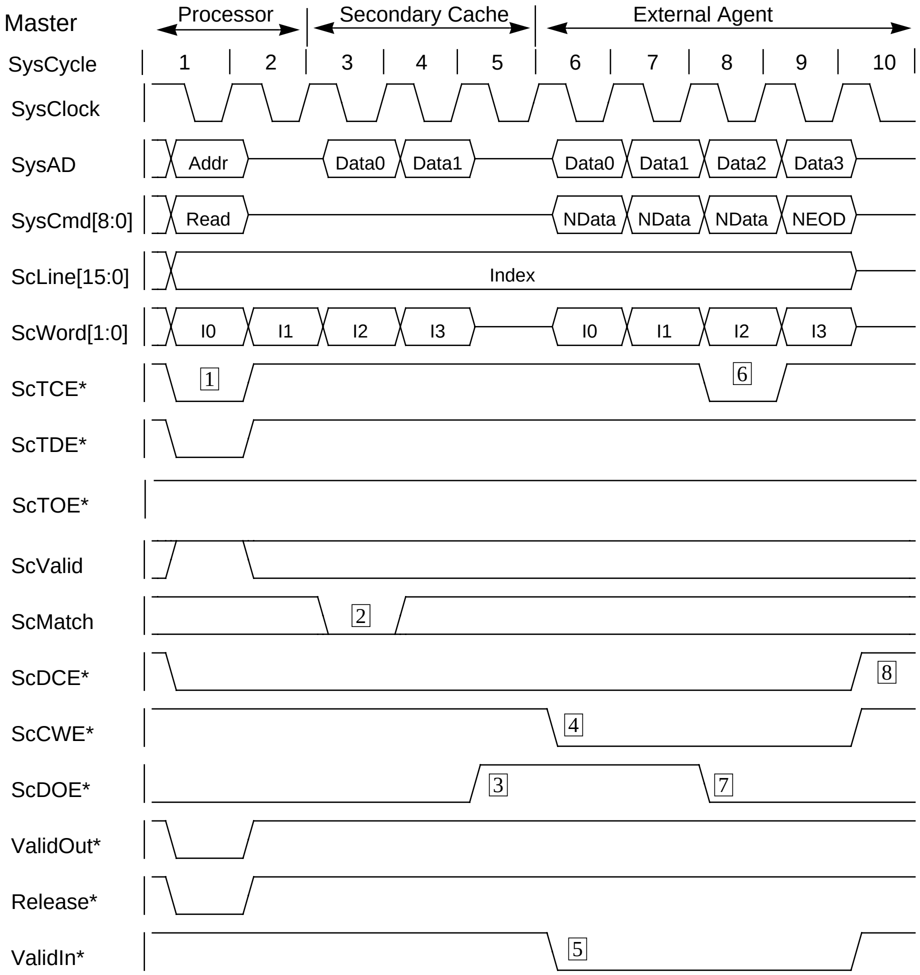


Figure 16-8 Secondary Cache Read Miss

16.2.3 Secondary Cache Read Miss with Bus Error

Figure 16-9 shows a secondary cache read miss with bus error protocol. This protocol is the same as the secondary cache read miss except:

1. The external agent supplies the first doubleword of the read response data with the data error bit set (**SysCmd[5]=1**). Note that the data error bit of **SysCmd** is only checked during the first coubleword of a read response.
2. The processor asserts **ScTCE*** and **SCTDE*** to write the new tag value into the secondary cache tag RAM with **ScValid** negated to invalidate this line.

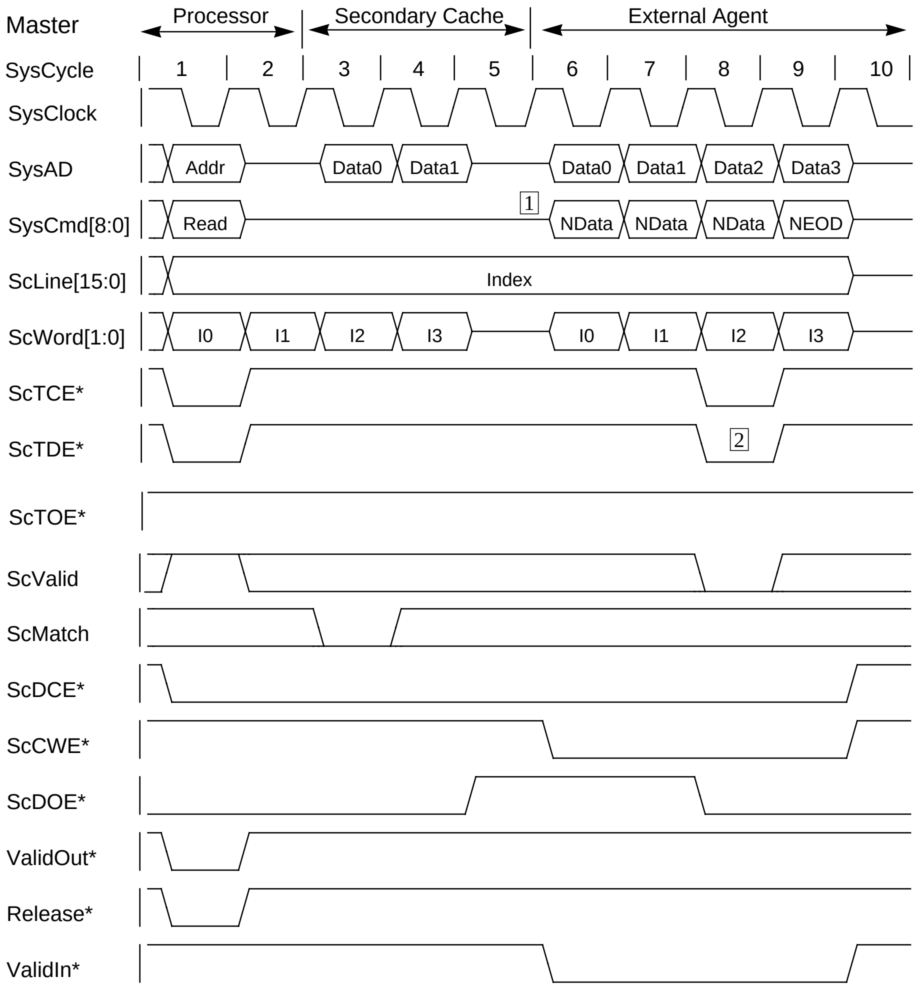


Figure 16-9   Secondary Cache Read Miss with Bus Error

16.3 Secondary Cache Write

Figure 16-10 shows a secondary cache write protocol. For the external agent, this protocol is the same as a non-secondary cache mode block write to the external agent, but the data is also written into the secondary cache.

1. The processor issues a block write and also asserts **ScTCE***, **ScTDE***, and **ScCWE*** in order to write the tag portion of the address on **SysAD** into the secondary cache tag RAM. The processor asserts **ScValid** to set the secondary cache tag to valid.
2. The processor asserts **ScDCE*** to write the block into the secondary cache data RAMs.

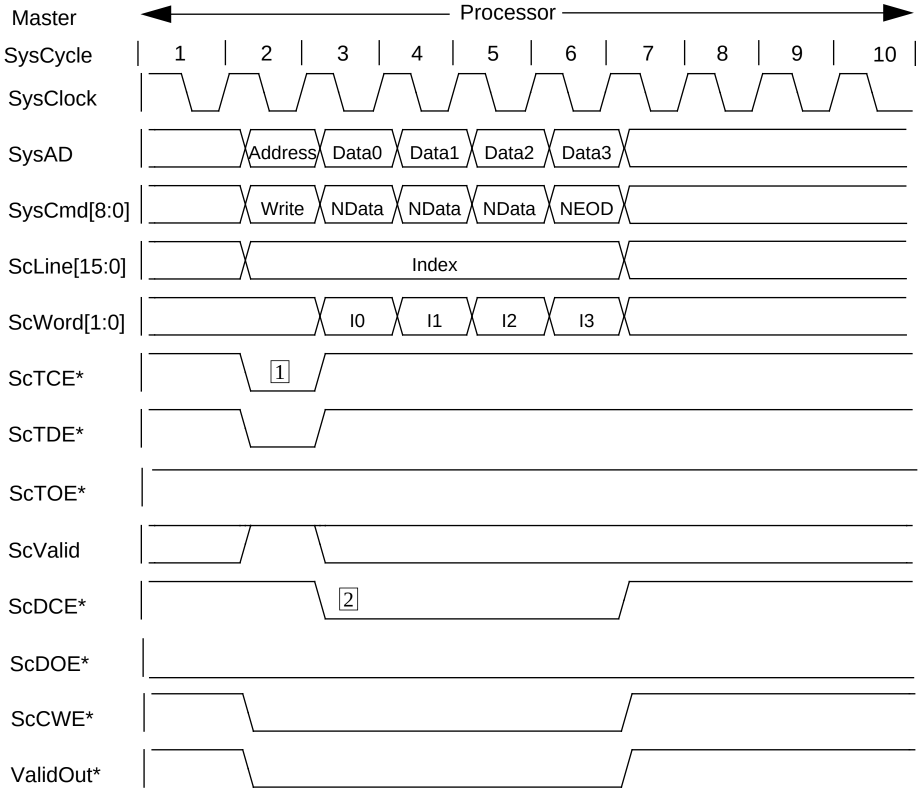


Figure 16-10 Secondary Cache Write Operation

16.4 Secondary Cache Line Invalidate

The V_R5000 processor has the ability to invalidate either a single line or 128 consecutive lines (address aligned) of the secondary cache. The invalidate operation is analogous to writing to the Tag RAM and invalidating the line in question. The **ScTCE***, **ScTDE***, and **ScCWE*** signals are driven active in the same clock as the **SysAD** and **ScLine** busses with **ScValid** negated. Invalidates are the only cache operations which may occur back-to-back. Note that **ValidOut*** is not asserted during secondary cache invalidate operations as the external agent does not participate in secondary cache invalidates.

Figure 16-11 shows the secondary cache invalidate protocol.

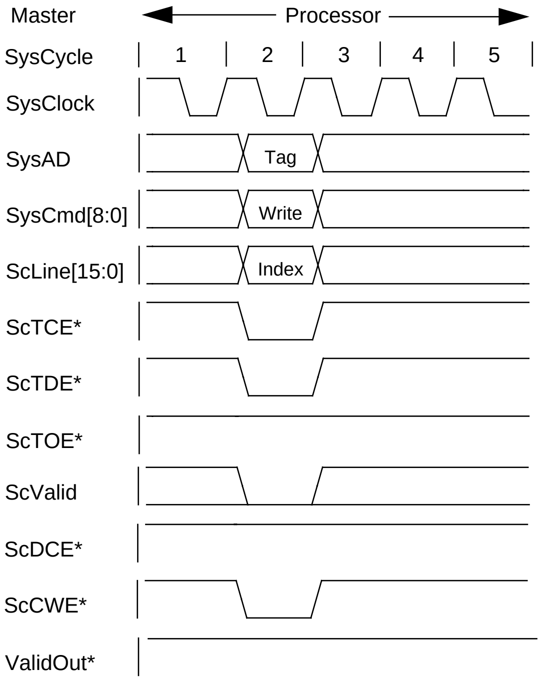


Figure 16-11 Secondary Cache Line Invalidate

The repeat rate for cache line invalidate instructions is two **SysClock**s. The repeat rate for cache page invalidate is one **SysClock** per line for 128 consecutive **SysClock** cycles.

16.5 Secondary Cache Probe Protocol

The secondary cache probe operation is analogous to a Tag RAM read operation. The **ScTCE*** and **ScTDE*** signals are asserted in the same clock as system address and the secondary cache line index. The processor then tri-states the **SysAD** bus. **ScTOE*** is asserted one clock later and the tag information is driven onto the **SysAD** bus. **ValidOut*** is not asserted during a secondary cache probe operation as the external agent does not participate in secondary cache probes. The Tag RAM bits are driven onto **SysAD [35:19]** and **ScValid**, which are the only **SysAD** signals valid during a probe operation. Figure 16-12 shows a timing diagram of a secondary cache probe protocol.

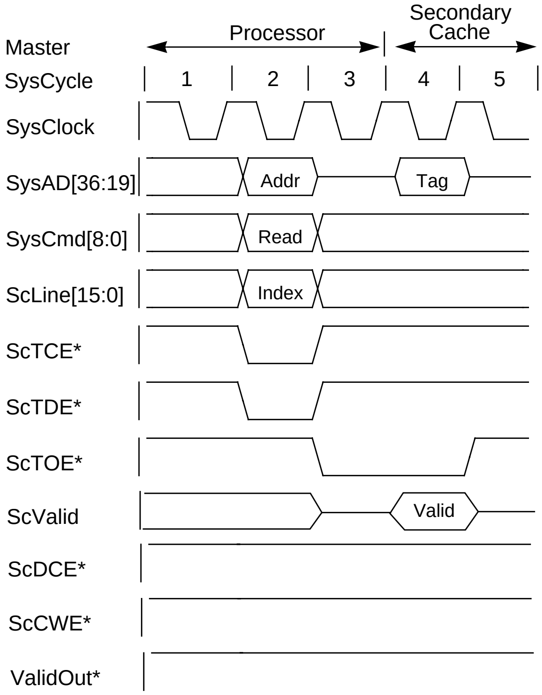


Figure 16-12 Secondary Cache Probe (Tag RAM Read)

16.6 Secondary Cache Flash Clear Protocol

In addition to the line invalidate operation, the V_R5000 processor also has the ability to invalidate the entire secondary cache in one operation. This operation allows the processor to clear the entire column of Tag RAM valid bits. In order to execute this operation the Tag RAM must support a flash clear of the valid bit column. As with the line invalidate operation, **ValidOut*** is not asserted during the flash clear operation as the external agent does not participate in flash clear operations. In addition, the **ScTCE***, **ScTDE***, and **ScCWE*** signals need not be asserted. The assertion of **ScCLR*** is all that is necessary for the Tag RAM to perform the requested operation. Figure 16-13 illustrates the secondary cache flash clear protocol.

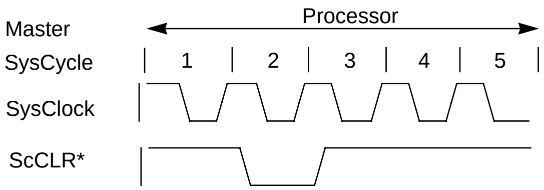


Figure 16-13 Secondary Cache Flash Clear

16.7 Secondary Cache Mode Configuration

The secondary cache configuration is specified by the processor ROM mode serial bit [15]. The state of this bit is indicated by the Secondary Cache (SC) bit in the CP0 config register (bit 17). If bit [17] is zero, a secondary cache is present in the system. If no secondary cache is present, or the secondary cache is disabled, the processor drives all secondary cache signals to their inactive state.

If no secondary cache is present and the mode ROM is configured for no secondary cache, the **ScMatch** and **ScDOE*** signals become don't-care inputs and must be terminated to valid logic levels. If the secondary cache is present and enabled, then the **SysADC** signals must implement valid parity during block read responses.

The doublewords transferred on **SysAD** during secondary cache block read transactions are in sub-block order. The doublewords transferred on **SysAD** during secondary cache block write transactions are in sequential order.

The size of the secondary cache is indicated by the processor mode ROM serial bits [17:16], and are encoded as follows:

[17:16] = 00 - 512 KB

[17:16] = 01 - 1 MB

[17:16] = 10 - 2 MB

[17:16] = 11 - Reserved

The state of these bits appear as CP0 config register bits [21:20].

Chapter 17 Interrupts

The V_R5000 processor supports the following interrupts: six hardware interrupts, one internal “timer interrupt,” two software interrupts, and one nonmaskable interrupt. The processor takes an exception on any interrupt. This chapter describes the six hardware and single nonmaskable interrupts.

17.1 Hardware Interrupts

The six CPU hardware interrupts can be caused by either an external write request to the V_R5000, or through dedicated interrupt pins. These pins are latched into an internal register by the rising edge of **SysClock**.

17.2 Nonmaskable Interrupt (NMI)

The nonmaskable interrupt is caused either by an external write request to the V_R5000 or by a dedicated pin in the V_R5000. This pin is latched into an internal register by the rising edge of **SysClock**.

Caution If a pipeline cancelling logic (e.g. cache error, bus error) occurs after the V_R5000 detects an NMI by the V_R5000 starts the NMI handling, the NMI will be cancelled and only the pipeline cancelling logic will be handled.
If an NMI cancellation occurred, make NMI* inactive once and then make it active again after the NMI cancellation.

17.3 Asserting Interrupts

External writes to the CPU are directed to various internal resources, based on an internal address map of the processor. When **SysAD[6:4]** = 0, an external write to any address writes to an architecturally transparent register called the *Interrupt* register; this register is available for external write cycles, but not for external reads.

During a data cycle, **SysAD[22:16]** are the write enables for the seven individual *Interrupt* register bits and **SysAD[6:0]** are the values to be written into these bits. This allows any subset of the *Interrupt* register to be set or cleared with a single write request. Figure 17-1 shows the mechanics of an external write to the *Interrupt* register.

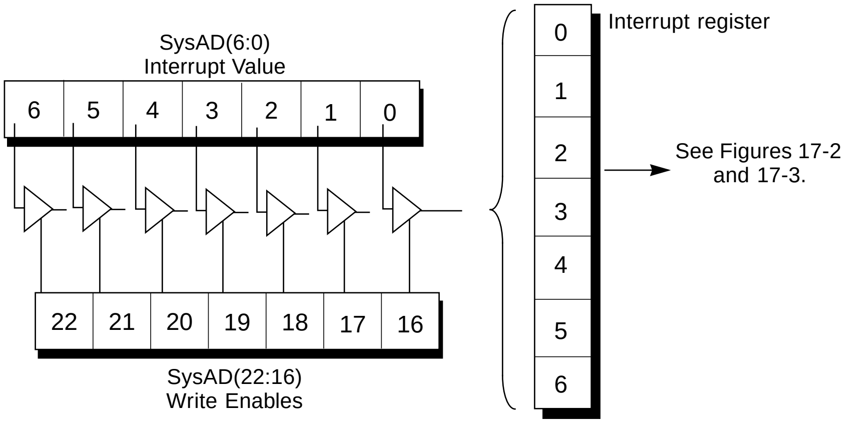


Figure 17-1 Interrupt Register Bits and Enables

Figure 17-2 shows how the V_R5000 interrupts are readable through the Cause register.

- Bit 5 of the *Interrupt* register is OR'ed with the **Int*[5]** pin and then multiplexed with the **TimerInterrupt** signal. The result is directly readable as bit 15 of the *Cause* register.
- Bits 4:0 of the *Interrupt* register are bit-wise OR'ed with the current value of interupt pins **Int*[4:0]**. The result is directly readable as bits 14:10 of the *Cause* register.

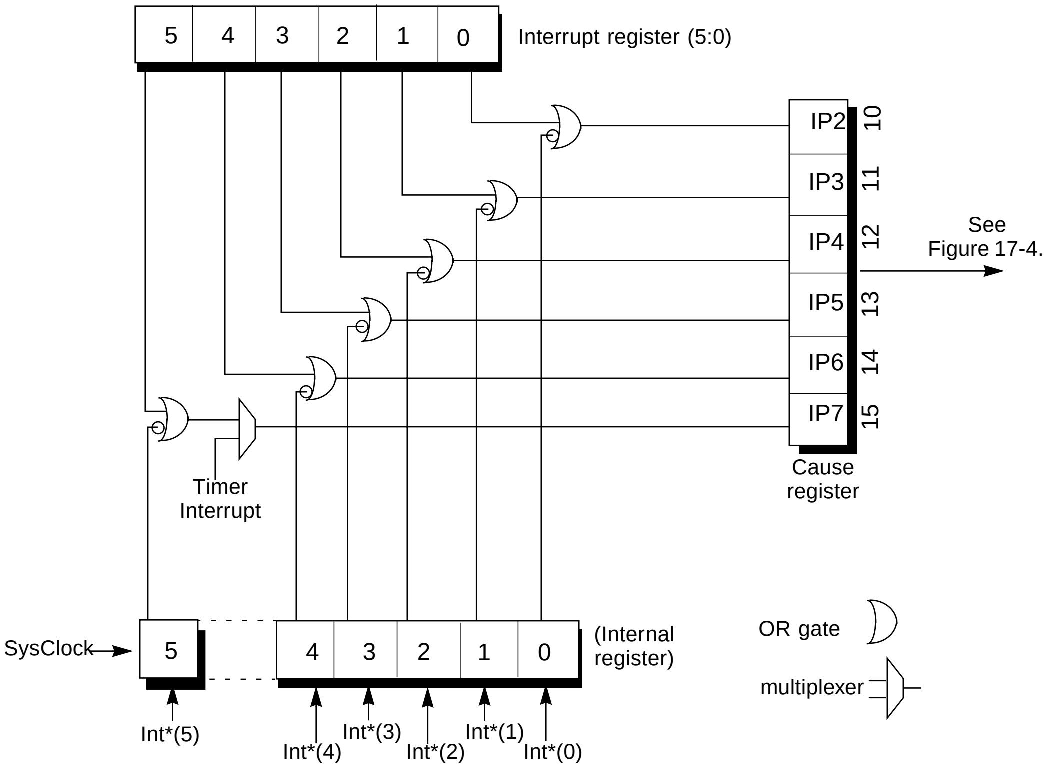


Figure 17-2 V_R5000 Interrupt Signals

Figure 17-3 shows the internal derivation of the **NMI** signal for the V$_R$5000 processor.

The **NMI*** pin is latched by the rising edge of **SysClock**. Bit 6 of the *Interrupt* register is then OR'ed with the inverted value of **NMI*** to form the nonmaskable interrupt. Only the falling edge of the latched signal will cause the NMI.

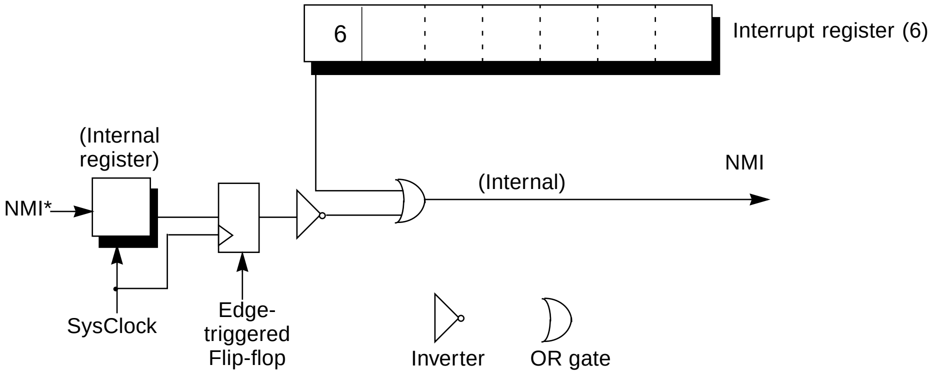


Figure 17-3 V$_R$5000 Nonmaskable Interrupt Signal

Figure 17-4 shows the masking of the V$_R$5000 interrupt signal.

- *Cause* register bits 15:8 (IP7-IP0) are AND-ORed with *Status* register interrupt mask bits 15:8 (IM7-IM0) to mask individual interrupts.
- *Status* register bit 0 is a global Interrupt Enable (IE). It is ANDed with the output of the AND-OR logic to produce the V$_R$5000 interrupt signal.

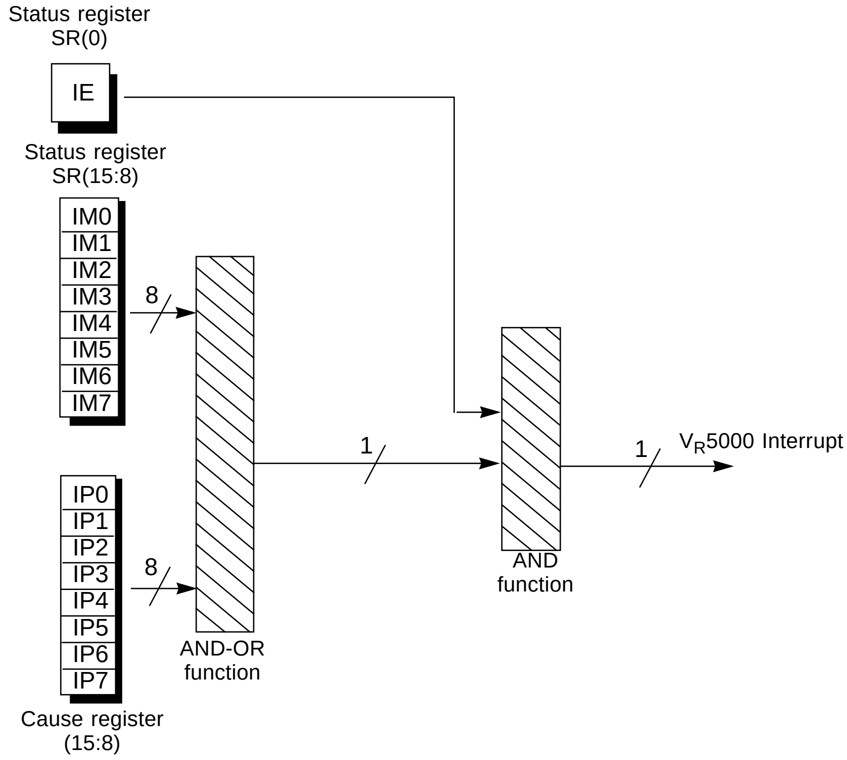


Figure 17-4   Masking of the V_R5000 Interrupt

Chapter 18 Standby Mode Operation

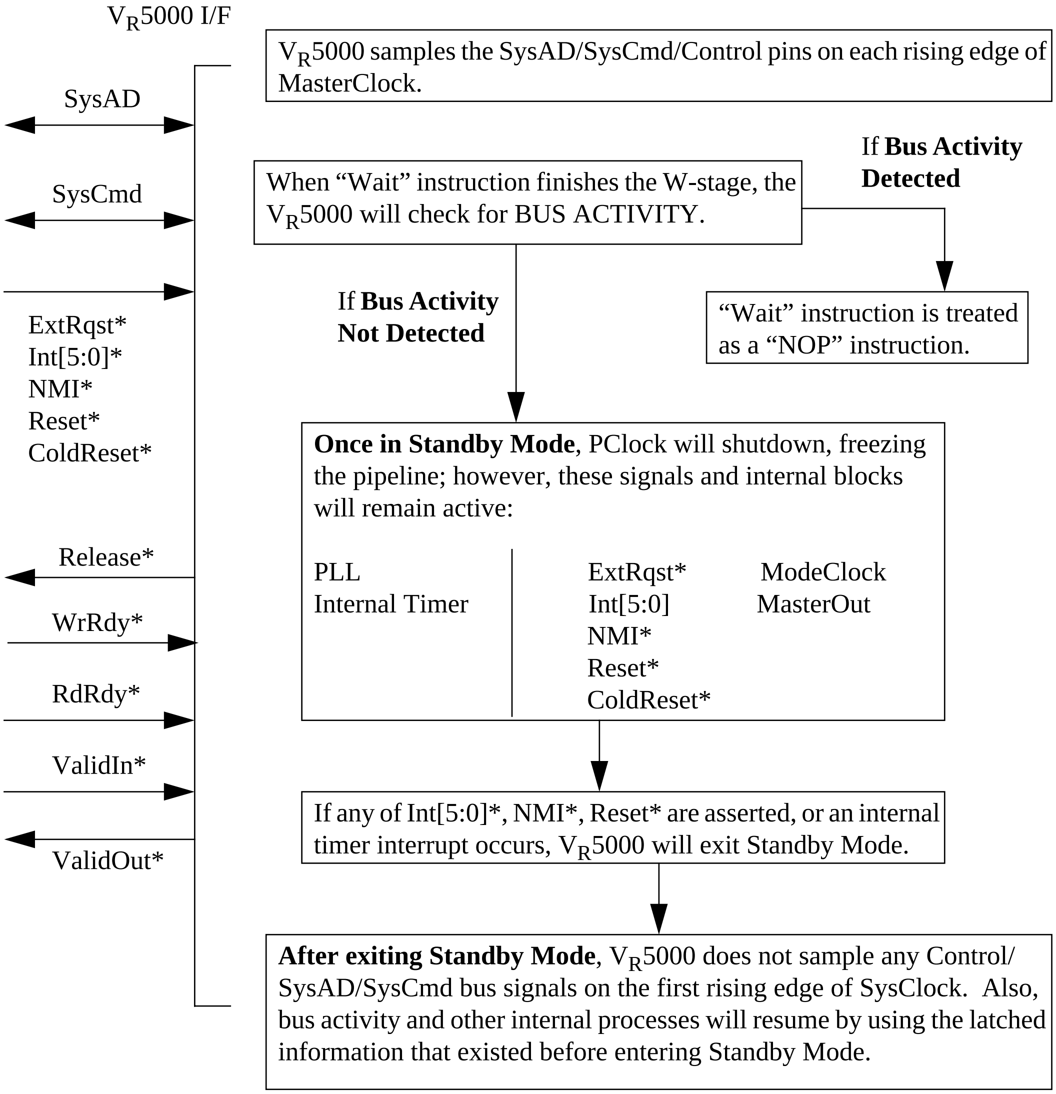
The Standby Mode operation is a means of reducing the internal core's power consumption when the CPU is in a "standby" state. In this chapter, the Standby Mode operation is discussed.

18.1 Entering Standby Mode

To enter Standby Mode, first execute the WAIT instruction. When the WAIT instruction finishes the W pipe-stage, if the SysAD bus is currently idle, the internal clocks will shut down, thus freezing the pipeline. The PLL, internal timer, some of the input pin clocks (**Int[5:0]***, **NMI***, **ExtRqst***, **Reset*** and **ColdReset***), and the output clock (**ModeClock**) will continue to run. If these conditions are not correct when the WAIT instruction finishes the W pipe-stage (i.e., the **SysAD** bus is not idle), the WAIT is treated as a NOP.

When the processor enters Standby Mode, the system interface signals are in their idle state and the processor is the master of the **SysAD** bus. The **Int***, **NMI***, **ExtReq***, **Reset***, and **ColdReset*** signals are monitored for an interrupt or reset condition that signals the end of Standby Mode.

Once the CPU is in Standby Mode, any interrupt, including **ExtRqst*** or **Reset***, will cause the CPU to exit Standby Mode. Figure 18-1 illustrates the Standby Mode Operation.



V_R5000 samples the SysAD/SysCmd/Control pins on each rising edge of MasterClock.

When “Wait” instruction finishes the W-stage, the V_R5000 will check for BUS ACTIVITY.

“Wait” instruction is treated as a “NOP” instruction.

Once in Standby Mode, PClock will shutdown, freezing the pipeline; however, these signals and internal blocks will remain active:

PLL
Internal Timer

ExtRqst*
Int[5:0]
NMI*
Reset*
ColdReset*

ModeClock
MasterOut

If any of Int[5:0]*, NMI*, Reset* are asserted, or an internal timer interrupt occurs, V_R5000 will exit Standby Mode.

After exiting Standby Mode, V_R5000 does not sample any Control/SysAD/SysCmd bus signals on the first rising edge of SysClock. Also, bus activity and other internal processes will resume by using the latched information that existed before entering Standby Mode.

Note: During Standby Mode, all control signals for the CPU must be deasserted or put into the appropriate state, and all input signals, except **Int[5:0]***, **Reset***, **ColdReset*** and **ExtRqst***, must remain unchanged.

Figure 18-1 Standby Mode Operation

Chapter 19 PLL Analog Power Filtering

For noisy module environments a filter circuit of the following form is recommended as shown in Figure 19-1.

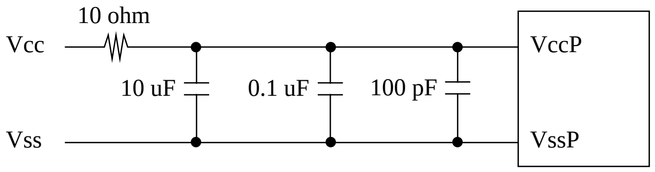


Figure 19-1 PLL Filter Circuit (1)

Because the optimum values of filter elements differ depending on the application and noise environment of the system, the above values are given for reference only. Find the optimum values for users' application through trial and error. A choke element (inductor) may be used instead of the resistor used as a power filter.

In the case that the processor's behavior is unstable with the above filter circuit, as shown in Figure 19-1, please insert a resistor (e.g. 10 ohm) between V_{SS} and V_{SS}P, as shown in Figure 19-2. Please make a full evaluation on your board to insert the resistor.

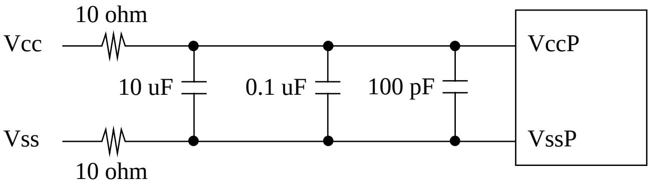


Figure 19-2 PLL Filter Circuit (2)

Chapter 20 V_R5000 Instruction Hazards

20.1 Introduction

This chapter identifies the V_R5000 Instruction Hazards. Certain combinations of instructions are not permitted because the results of executing such combinations are unpredictable in combination with some events, such as pipeline delays, cache misses, interrupts, and exceptions.

Most hazards result from instructions modifying and reading state in different pipeline stages. Such hazards are defined between pairs of instructions, not on a single instruction in isolation. Other hazards are associated with restartability of instructions in the presence of exceptions.

For the following code hazards, the behavior is undefined and unpredictable.

20.2 List of Instruction Hazards

- Any instruction that would modify PageMask or EntryHi or EntryLo0 or EntryLo1 or Random CP0 Registers should not be followed by a TLBWR instruction. There should be at least two integer instructions between the register modification and the TLBWR instruction.
- Any instruction that would modify PageMask or EntryHi or EntryLo0 or EntryLo1 or Index CP0 Registers should not be followed by a TLBWI instruction. There should be at least two integer instructions between the register modification and the TLBWI instruction.
- Any instruction that would modify the Index CP0 Register or the contents of the JTLB should not be followed by a TLBR instruction. There should be at least two integer instructions between the register modification and the TLBR instruction.
- Any instruction that would modify the PageMask or EntryHi or CPO Registers or the contents of the JTLB should not be followed by a TLBP instruction. There should be at least two integer instructions between the register modification and the TLBP instruction.
- Any instruction that would modify the EPC or ErrorEPC or Status CPO Registers should not be followed by an ERET instruction. There should be at least two integer instructions between the register modification and the ERET instruction.
- A Branch or Jump instruction is not allowed to be in the delay-slot of another Branch/Jump instruction. This sequence is illegal in the MIPs architecture.
- The two instructions preceding any DIV, DIVU, DDIV, DDIVU, MULT, MULTU, DMULT or DMULTU instructions should not read the HI or LO registers. There should be at least two integer instructions between the register read and the register modification.
- Any instruction that would modify Count Register should not be followed by any instruction that would read Count Register when the Boot Mode Serial bit 18 is 0. There should be at least two integer instructions between the register modification and the register read.

Appendix A Cycle Counts for V_R5000 Cache Operations

A.1 Cycle Counts for V_R5000 Cache Misses

A.1.1 Mnemonics

To describe processor sequences that inlude a memory access, the number of cycles must be calculated based on the system response to a memory access. Such sequences will be described with equations based on the following mnemonics:

- SYSDIV: The number of processor cycles per system cycle, ranges from 2 - 8.
- ML: Number of system cycles of memory latency defined as the number of cycles the **SysAD** bus is driven by the external agent before the first doubleword of data appears.
- DD: Number of system cycles required to return the block of data, defined to be the number of cycles beginning when the first doubleword of data appears on the **SysAD** bus and ending when the last double word of data appears on the **SysAD** bus inclusive.
- {0 to (SYSDIV - 1)}: In many equations this term is used. It has a value (number of cycles) between 0 and (SYSDIV - 1) depending on the alignment of the execution of the cache miss or cache op with the system clock.

A.1.2 DCache Misses

Caveats to DCache Misses:

1) All Cycle counts are in processor cycles.

2) DCache misses have lower priority than write backs, external requests, and ICache misses. If the write back buffer contains unwritten data when a dcache miss occurs, the write back buffer will be retired before the handling of the dcache miss is begun. Instruction cache misses are given priority over data cache misses. If an icache miss occurs at the same time as a dcache miss, the icache miss will be handled first. External requests will be completed before beginning the handling of a dcache miss.

3) For all data cache misses handling of the returning cache miss data must wait for the store buffer and response buffer to empty (if they are filled) and for dirty data (if present) to be moved from the dcache to the write back buffer. It is possible that if all of the above occur, and the dcache miss hits in the secondary cache, the first doubleword of data will return before the data cache is available. In this case the first doubleword of data will hold in the response buffer for one or two cycles which will add to the latency of the dcache miss.

4) In handling a dcache miss a write back may be required which will fill the write back buffer. Write backs can affect subsequent cache misses since they will stall until the write back buffer is written back to memory.

5) All cycle counts are best case assuming no interference from the mechanisms described above.

The following equations yield the number of stall cycles for data cache misses under the specified circumstances.

Secondary cache hit:

Number_Of_Cycles_For_DCache_Miss_Secondary_Cache_Hit =

1 + {0 to (SYSDIV - 1)} + (3 x SYSDIV) + 2

Secondary cache miss:

Number_Of_Cycles_For DCache_Miss_Secondary _Cache_Miss =

1 + {0 to (SYSDIV - 1)} + (2 x SYSDIV) + (ML x SYSDIV) + (1 x SYSDIV) + 2

Note: Memory Latency (ML) has a minimum of 3 to allow for the secondary cache check.

A.1.3 ICache Misses

Caveats to ICache Misses

1) All cycle counts are in processor cycles.

2) ICache misses have lower priority than write backs and external requests. If the write back buffer contains unwritten data when an icache miss occurs, the write back buffer will be retired before the handling of the icache miss is begun. External requests will be completed before beginning the handling of an icache miss.

3) All cycle counts are best case assuming no interference from the mechanisms described above.

The following equations yield the number of stall cycles for instruction cache misses under the specified circumstances.

Secondary cache hit:

Number _Of_Cycles_For_ICache_Miss_Secondary_Cache_Hit =

1 + {0 to (SYSDIV - 1)} + (6 x SYSDIV) + 3

Secondary cache miss:

Number_Of_Cycles_For_ICache_Miss_Secondary_Cache_Miss =

1 + {0 to (SYSDIV - 1)} + (2 x SYSDIV) + (ML x SYSDIV) + (DD x SYSDIV) + 3

Note: Memory Latency (ML) has a minimum of 3 to allow for the secondary cache check.

A.2 Cycle Counts for V_R5000 Cache Operations

Caveats to Cache Operations

1) All cycle counts are in processor cycles.

2) All cache ops have lower priority than cache misses, write backs and external requests. If the write back buffer contains unwritten data when a cache op is executed, the write back buffer will be retired before the cache op is begun. If an instruction

cache miss occurs at the same time as a cache op is executed, the instruction cache miss will be handled first. Cache ops are mutually exclusive with respect to data cache misses. External requests will be completed before beginning a cache op.

3) For all data cache ops the cache op machine waits for the store buffer and response buffer to empty before beginning the cache op. This can add 3 cycles to any data cache op if there is data in the response buffer or store buffer. The response buffer contains data from the last data cache miss that has not yet been written to the data cache. The store buffer contains delayed store data waiting to be written to the data cache.

4) Cache ops of the form xxxx_Writeback_xxxx may perform a write back which will fill the write back buffer. Write backs can affect subsequent cache ops since they will stall until the write back buffer is written back to memory. Cache ops which fill the write back buffer are noted in the following tables.

5) All cycle counts are best case assuming no interference from the mechanisms described above.

Table A-1   Primary Data Cache Operations

Code	Name	Number of Cycles
0	Index_Writeback_Invalidate_D	10 Cycles if the cache line is clean. 12 Cycles if the cache line is dirty. (Write back)
1	Index_Load_Tag_D	7 Cycles
2	Index_Store_Tag_D	8 Cycles
3	Create_Dirty_Exclusive_D	10 Cycles for a cache hit. 13 Cycles for a cache miss if the cache line is clean. 15 Cycles for a cache miss if the cache line is dirty. (Writeback)
4	Hit_Invalidate_D	7 Cycles for a cache miss. 9 Cycles for a cache hit.
5	Hit_Writeback_Invalidate_D	7 Cycles for a cache miss. 12 Cycles for a cache hit if the cache line is clean. 14 Cycles for a cache hit if the cache line is dirty. (Writeback)
6	Hit_Writeback_D	7 Cycles for a cache miss. 10 Cycles for a cache hit if the cache line is clean. 14 Cycles for a cache hit if the cache line is dirty. (Writeback)

Table A-2 Primary Instruction Cache Operations

Code	Name	Number of Cycles
0	Index_Invalidate_I	7 Cycles.
1	Index_Load_Tag_I	7 Cycles.
2	Index_Store_Tag_I	8 Cycles.
3	NA	
4	Hit_Invalidate_I	7 Cycles for a cache miss. 9 Cycles for a cache hit.
5	Fill_I	This equation yields the number of processor cycles for a Fill_I cache op: Number_Of_Cycles_For_A_Fill_I_Cacheop = 10 + {0 to (SYSDIV -1)} + (2 x SYSDIV) + (ML x SYSDIV) + (DD x SYSDIV).
6	Hit_Writeback_I	7 Cycles for a cache miss. 20 Cycles for a cache hit. (Writeback)

Table A-3 Secondary Cache Operations

Code	Name	Number of Cycles
0	Flash_Invalidate_S	This equation yields the number of processor cycles for a Flash_Invalidate_S cache op: Number_Of_Cycles_For_Flash_Invalidate_S_Cacheop = 3 + {0 to (SYSDIV - 1)} + (1 x SYSDIV) + 3
1	Index_Load_Tag_S	This equation yields the number of processor cycles for an Index_Load_Tag_S cache op: Number_Of_Cycles_For_Index_Load_Tag_S = 3 + {0 to (SYSDIV -1)} + (4 x SYSDIV) + 3
2	Index_Store_Tag_S	This equation yields the number of processor cycles for an Index_Store_Tag_S cache op: Number_Of_Cycles_For_Index_Store_Tag_S = 3 + {0 to (SYSDIV - 1)} + (1 x SYSDIV) + 3
3	NA	
4	NA	
5	Page_Invalidate_S	This equation yields the number of processor cycles for a Page_Invalidate_S cache op: Number_Of_Cycles_For_Page_Invalidate_S = 3 + {0 to (SYSDIV -1)} + (128 x SYSDIV) + 3
6	NA	

Appendix B Subblock Order

A block of data elements (whether bytes, halfwords, words, or doublewords) can be retrieved from storage in two ways: in sequential order, or using a subblock order. This appendix describes these retrieval methods, with an emphasis on subblock ordering.

Sequential ordering retrieves the data elements of a block in serial, or sequential, order.

Figure B-1 shows a sequential order in which doubleword 0 is taken first and doubleword 3 is taken last.

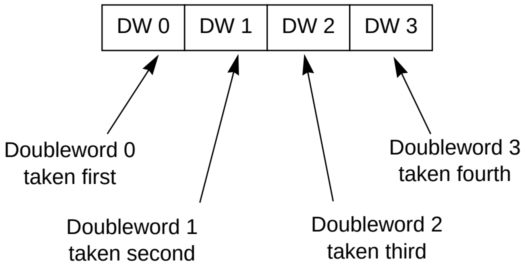


Figure B-1   Retrieving a Data Block in Sequential Order

Subblock ordering allows the system to define the order in which the data elements are retrieved. The smallest data element of a block transfer for the V_R5000 is a doubleword, and Figure B-2 shows the retrieval of a block of data that consists of 4 doublewords, in which DW2 is taken first.

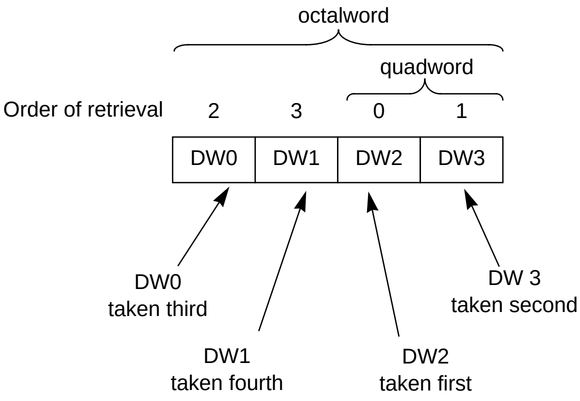


Figure B-2 Retrieving a Data in a Subblock Order

Using the subblock ordering shown in Figure B-2, the doubleword at the target address is retrieved first (DW2), followed by the remaining doubleword (DW3) in this quadword.

It may be easier way to understand subblock ordering by taking a look at the method used for generating the address of each doubleword as it is retrieved. The subblock ordering logic generates this address by executing a bit-wise exclusive-OR (XOR) of the starting block address with the output of a binary counter that increments with each doubleword, starting at doubleword zero (00_2).

Using this scheme, Table B-1 through Table B-3 list the subblock ordering of doublewords for an 8-word block, based on three different starting-block addresses: 10_2, 11_2, and 01_2. The subblock ordering is generated by an XOR of the subblock address (either 10_2, 11_2, and 01_2) with the binary count of the doubleword (00_2 through 11_2). Thus, the third doubleword retrieved from a block of data with a starting address of 10_2 is found by taking the XOR of address 10_2 with the binary count of DW2, 10_2. The result is 00_2, or DW0.

The remaining tables illustrate this method of subblock ordering, using various address permutations.

Table B-1 Subblock Ordering Sequence: Address 10_2

Cycle	Starting Block Address	Binary Count	Double Word Retrieved
1	10	00	10
2	10	01	11
3	10	10	00
4	10	11	01

Table B-2 Subblock Ordering Sequence: Address 11_2

Cycle	Starting Block Address	Binary Count	Double Word Retrieved
1	11	00	11
2	11	01	10
3	11	10	01
4	11	11	00

Table B-3 Subblock Ordering Sequence: Address 01_2

Cycle	Starting Block Address	Binary Count	Double Word Retrieved
1	01	00	01
2	01	01	00
3	01	10	11
4	01	11	10

Appendix C Driver Strength Control

The speed of the V_R5000 output drivers is statically controlled at boot time. This appendix discusses the output buffer strength control mechanism in the V_R5000 processor.

Two of the boot time mode bits are used to control the strength of the output buffer. These are boot mode bit 13 and 14.

The output driver strength can be from 100% (fastest) to 50% (slowest), based on the value of boot mode bits 13 and 14. Table C-1 shows the encoding for these boot mode bits and the selected driver strength.

Table C-1 Output Driver Strength

Boot Mode Bits 14 13	Driver Strength
1 0	100%
1 1	83%
0 0	67%
0 1	50%

Appendix D Differences between V_R5000 and V_R5000A

Parameter	V_R5000	V_R5000A
Maximum internal operating frequency	150/180/200 MHz	250/266 MHz
Multiplication ratio for clock interface (input vs. internal)	2, 3, 4, 5, 6, 7, 8	2, 2.5**Note**, 3, 4, 5, 6, 7, 8
Supply voltage	3.3V±5%	Core: 2.4V±0.1V (100 to 235 MHz) 2.5V±5% (236 to 250 MHz) 2.6V±0.1V (251 to 266 MHz) I/O : 3.3V±5%
Package	•223-pin ceramic PGA •272-pin plastic BGA (cavity down advanced type)	272-pin plastic BGA (cavity down advanced type)

Note Selectable only when SysClock = 100MHz

Appendix E Differences between V_R5000 and V_R4310

Item		V_R5000	V_R4310
Operation Frequency	Internal	200 MHz MAX.	167 MHz MAX.
	External	100 MHz MAX.	83.3 MHz MAX.
Pipeline		2-way superscalar 5-stage pipeline	5-stage pipeline
Cache	On-chip Primary Instruction Cache	32 KB (2-way set)	16 KB (direct map)
	On-chip Primary Data Cache	32 KB (2-way set)	8 KB (direct map)
	Secondary Cache Interface	Incorporated (direct map)	N/A
	Data Protection	Byte parity	N/A
System Bus	Write Data Transfer Rate	9 types (DD, DDxDDx, DDxxDDxx, DxDx, DDxxxDDxxx, DDxxxxDDxxxx, DxxDxx, DDxxxxxxDDxxxxxx, DxxxDxxx)	2 types (DD, DxxDxx)
	SysAD Bus Used after Last D Cycle	Unused for trailing x cycles	Maintains last D cycle value

Item		V_R5000	V_R4310
Boot Mode Setting		Serial data input from ModeIn pin	Specific by DivMode (2:0)
Integer Operating Unit		MIPS I, II, III, IV instruction set	MIPS I, II, III instruction set
JTAG Interface		N/A	Incorporated
SyncIn - SyncOut Path		N/A	Available
Clock Interface	PClock Divisor	2, 3, 4, 5, 6, 7, or 8	1.5, 2, 2.5, 3, 4, 5, or 6
	System Bus Clock Divisor	2, 3, 4, 5, 6, 7, or 8	1.5, 2, 2.5, 3, 4, 5, or 6
	Clock Output	N/A	TClock
Power Control Mode		Standby mode (freezing pipeline)	N/A
PRId Register		Imp = 0x23	Imp = 0x0B

Appendix F V_R5000 Restrictions

- Any load-linked memory reference that hits in the DTLB will cause the LLAddr register to hold the virtual address of that reference instead of the physical address.
- C0_CacheErr[2] does not report Virtual Address [14] of the parity error location. This bit is always read as zero.
- If a pipeline cancelling logic (e.g. cache error, bus error) occurs after the V_R5000 detects a non-maskable interrupt (NMI) by the V_R5000 starts the NMI handling, the NMI will be cancelled and only the pipeline cancelling logic will be handled.
  If an NMI cancellation occurred, make NMI* inactive once and then make it active again after the NMI cancellation.
- An LL or LLD instruction targeting 64-bit Kernel xkphys address space issues a 4-byte uncached read request or 8-byte uncached read request respectively. If the targeted primary data cache line for an LL/LLD instruction is dirty, the cache data is ignored and an uncached load from memory is executed, and consequently the consistency of data is not guaranteed.
  Therefore, write back the line from the primary data cache to memory before the execution of an LL/LLD instruction targeting xkphys address space.
  Example of a program is as follows.

example:

```
cache     Hit_writeback_d, offset(base)
ll        rt, offset(base)
  :
sc        rt, offset(base)
```

Appendix G Index

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: ____________________

Document number: ____________________ Page number: ____________________

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑