NEC

User's Manual

VR4100 Series™

64-/32-bit Microprocessor

Architecture

Target Device

μPD30121 (VR4121™)
μPD30122 (VR4122™)
μPD30131 (VR4131™)
μPD30181 (VR4181™)
μPD30181A, 30181AY (VR4181A™)

Document No. U15509EJ2V0UM00 (2nd edition)
Date Published June 2002 NS CP(K)

Printed in Japan

[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 V_{DD} 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.

Purchase of NEC I^2C components conveys a license under the Philips I^2C Patent Rights to use these components in an I^2C system, provided that the system conforms to the I^2C Standard Specification as defined by Philips.

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 April, 2002. 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 do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

- **Sucursal en España**
  Madrid, Spain
  Tel: 091-504 27 87
  Fax: 091-504 28 60

- **Succursale Française**
  Vélizy-Villacoublay, France
  Tel: 01-30-67 58 00
  Fax: 01-30-67 58 99

- **Filiale Italiana**
  Milano, Italy
  Tel: 02-66 75 41
  Fax: 02-66 75 42 99

- **Branch The Netherlands**
  Eindhoven, The Netherlands
  Tel: 040-244 58 45
  Fax: 040-244 45 80

- **Branch Sweden**
  Taeby, Sweden
  Tel: 08-63 80 820
  Fax: 08-63 80 388

- **United Kingdom Branch**
  Milton Keynes, UK
  Tel: 01908-691-133
  Fax: 01908-670-290

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 Shanghai, Ltd.
Shanghai, P.R. China
Tel: 021-6841-1138
Fax: 021-6841-1137

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

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

J02.4

PREFACE

Readers

This manual targets users who intend to understand the functions of the VR4100 Series, the RISC microprocessors, and to design application systems using them.

Purpose

This manual introduces the architecture of the VR4100 Series to users, following the organization described below.

Organization

Two manuals are available for the VR4100 Series: Architecture User's Manual (this manual) and Hardware User's Manual of each product.

Architecture User's Manual	Hardware User's Manual
• Pipeline operation • Cache organization and memory management system • Exception processing • Interrupts • Instruction set	• Pin functions • Physical address space • Function of Coprocessor 0 • Initialization interface • Peripheral units

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.

In this manual, the following products are referred to as the VR4100 Series. Descriptions that differ between these products are explained individually, and common parts are explained as for the VR4100 Series.

VR4121 (μPD30121)
VR4122 (μPD30122)
VR4131 (μPD30131)
VR4181 (μPD30181)
VR4181A (μPD30181A, 30181AY)

To learn in detail about the function of a specific instruction,

→ Read **CHAPTER 2 CPU INSTRUCTION SET SUMMARY**, **CHAPTER 3 MIPS16 INSTRUCTION SET**, **CHAPTER 9 CPU INSTRUCTION SET DETAILS**, and **CHAPTER 10 MIPS16 INSTRUCTION SET FORMAT**.

To learn about the overall functions of the VR4100 Series,

→ Read this manual in sequential order.

To learn about hardware functions,

→ Refer to **Hardware User's Manual** which is separately available.

To learn about electrical specifications,

→ Refer to **Data Sheet** which is separately available.

Conventions

Data significance:	Higher on left and lower on right
Active low:	XXX# (trailing # after pin and signal names)
Note:	Description of item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	binary/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

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

Document name	Document number
VR4100 Series Architecture User's Manual	This manual
VR4121 User's Manual	U13569E
μPD30121 (VR4121) Data Sheet	U14691E
VR4122 User's Manual	U14327E
μPD30122 (VR4122) Data Sheet	U16219E
VR4131 Hardware User's Manual	U15350E
μPD30131 (VR4131) Data Sheet	To be prepared
VR4181 Hardware User's Manual	U14272E
μPD30181 (VR4181) Data Sheet	U14273E
VR4181A Hardware User's Manual	To be prepared
μPD30181A, 30181AY (VR4181A) Data Sheet	To be prepared
VR Series ™ Programming Guide Application Note	U10710E

CONTENTS

LIST OF FIGURES (1/3)

LIST OF FIGURES (2/3)

LIST OF FIGURES (3/3)

LIST OF TABLES (1/2)

LIST OF TABLES (2/2)

CHAPTER 1 INTRODUCTION

This chapter gives an outline of the VR4121 (μPD30121), the VR4122 (μPD30122), the VR4131 (μPD30131), the VR4181 (μPD30181), and the VR4181A (μPD30181A, 30181AY), which are 64-/32-bit RISC microprocessors. In this manual, these products are referred to as the VR4100 Series.

1.1 Features

The VR4100 Series, which is a part of the RISC microprocessor VR Series, is a group of products developed for PDAs. The VR Series is high-performance 64-/32-bit microprocessors employing the RISC (reduced instruction set computer) architecture developed by MIPS™ manufactured by NEC.

The VR4100 Series accommodates the ultra low power consumption CPU core provided with cache memory, a high-speed product-sum operation unit, and an address management unit. The VR4100 Series also has interface units for the peripheral circuits required for battery-driven portable information equipment (refer to **Hardware User's Manual** of each product for details about on-chip peripheral functions).

The features of the VR4100 Series are described below.

- Employs 64-bit RISC core as a CPU
  Possible to operate in 32-bit mode
- Optimized instruction pipeline
- On-chip cache memory
- Employs write-back cache
  Reduces store operations using system bus
- Physical address space: 32 bits
  Virtual address space: 40 bits
- Translation lookaside buffer (TLB) with 32-double entries
- Instruction set: MIPS III (however, the FPU, LL, LLD, SC, and SCD instructions are removed), MIPS16
- Supports high-speed product-sum operation instructions
- Effective power management features, which include the four modes of Fullspeed, Standby, Suspend, and Hibernate
- On-chip PLL and clock generator
- Variable on-chip peripheral functions ideal for potable information equipment

The functions of the VR4100 Series are listed as follows.

Table 1-1. Comparison of Functions of VR4100 Series

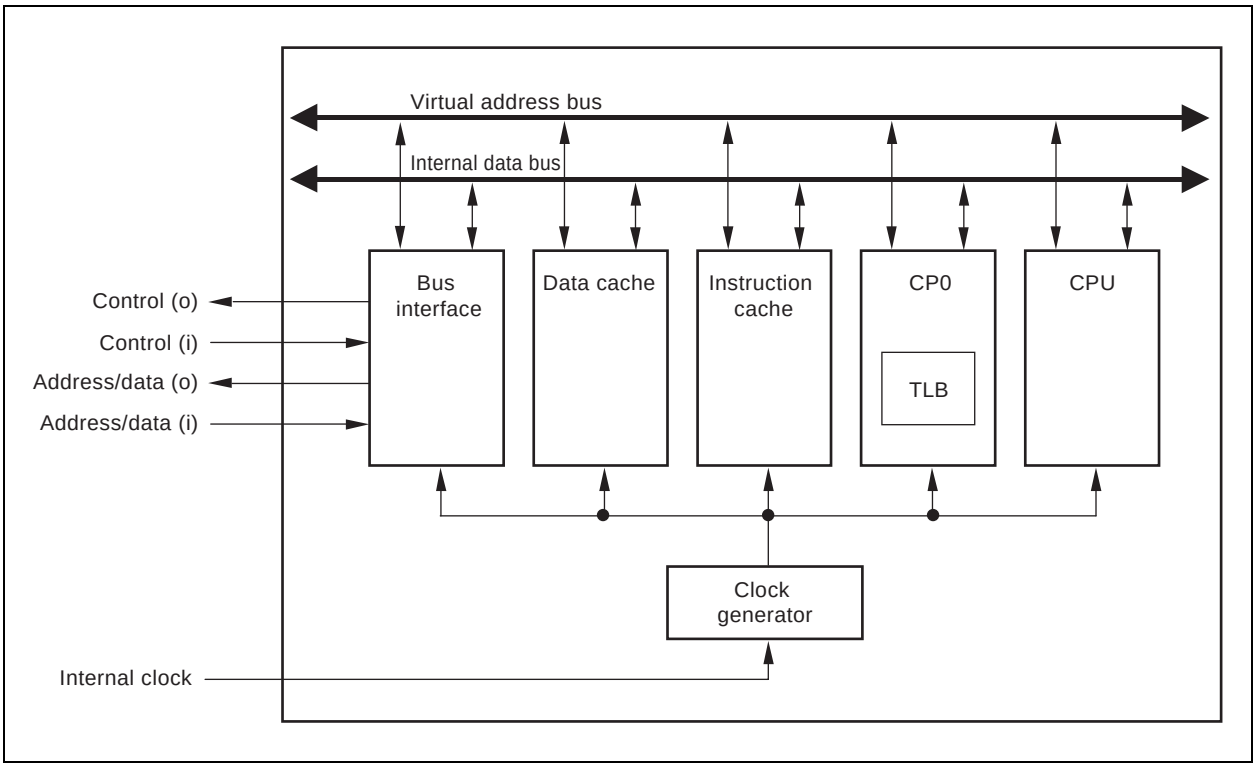
Item	VR4121	VR4122	VR4131	VR4181	VR4181A
Part number	μPD30121	μPD30122	μPD30131	μPD30181	μPD30181A, 30181AY
CPU core	VR4120™ core		VR4130™ core	VR4110™ core	VR4120 core
Instruction set	MIPS I, II, III + high-speed product-sum (32-bit) + MIPS16			MIPS I, II, III + high-speed product-sum (16-bit) + MIPS16	MIPS I, II, III + high-speed product-sum (32-bit) + MIPS16
Pipeline	5-/6-stage pipeline		2-way superscalar 6-/7-stage pipeline	5-stage pipeline	5-/6-stage pipeline
On-chip cache memory	• Instruction: 16KB • Data: 8KB • Direct map	• Instruction: 32KB • Data: 16KB • Direct map	• Instruction: 16KB • Data: 16KB • 2-way set-associative • With line lock function	• Instruction: 4KB • Data: 4KB • Direct map	• Instruction: 8KB • Data: 8KB • Direct map
On-chip peripheral functions	• Memory controller • Extension bus interface (ISA) • LCD interface • Touch panel interface • Keyboard interface • Communication interface (UART, CSI, IrDA (SIR, MIR, FIR)) • Modem interface • Audio interface • LED controller • DMA controller • Timer, counter • Watchdog timer • General-purpose port • Clock generator • Power management unit • A/D converter • D/A converter	• Memory controller • Extension bus interface (ISA, PCI) • Communication interface (UART, CSI, IrDA (SIR, MIR, FIR)) • LED controller • Timer, counter • General-purpose port • Clock generator • Power management unit		• Memory controller • Extension bus interface (ISA) • LCD interface • Touch panel interface • Keyboard interface • Communication interface (UART, CSI, IrDA (SIR)) • CompactFlash interface • Audio interface • LED controller • DMA controller • Timer, counter • Watchdog timer • General-purpose port • Clock generator • Power management unit • A/D converter • D/A converter	• Memory controller • Extension bus interface (ISA) • LCD interface • Touch panel interface • Keyboard interface • Communication interface (UART, CSI, I^2C, IrDA (SIR)) • CompactFlash interface • AC97/I^2S audio interface • DMA controller • USB host/function controller • PWM generator • Timer, counter • Watchdog timer • General-purpose port • Clock generator • Power management unit • A/D converter • D/A converter
Other functions	–	• On-chip branch prediction function • On-chip hardware debug function		–	• On-chip branch prediction function • On-chip hardware debug function

1.2 CPU Core

Figure 1-1 shows the internal block diagram of the CPU core.

In addition to the conventional high-performance integer operation units, this CPU core has a full-associative format translation lookaside buffer (TLB), which has 32 entries that provide mapping to 2-page pairs for one entry. Moreover, it also has instruction and data caches, and a bus interface.

Figure 1-1. CPU Core Internal Block Diagram



(1) CPU

CPU is a block that performs integer calculations. This block includes a 64-bit integer data path, and product-sum operator.

(2) Coprocessor 0 (CP0)

CP0 incorporates a memory management unit (MMU) and exception handling function. The MMU checks whether there is an access between different memory segments (user, supervisor, and kernel) by executing address conversion. The translation lookaside buffer (TLB) converts virtual addresses to physical addresses.

(3) Instruction cache

The instruction cache employs virtual index and physical tag formats. It is managed with direct mapping format in the VR4121, VR4122, VR4181, and VR4181A, or with 2-way set-associative format in the VR4131.

(4) Data cache

The data cache employs virtual index, physical tag, and writeback formats. It is managed with direct mapping format in the VR4121, VR4122, VR4181, and VR4181A, or with 2-way set-associative format in the VR4131.

(5) CPU bus interface

The bus interface controls data transmission/reception between the CPU core and peripheral units. The bus interface consists of two 32-bit multiplexed address/data buses (one for input, and the other for output), clock signals, interrupt request signals, and various other control signals.

(6) Clock generator

The clock generator processes clock inputs and supplies them to internal units.

1.2.1 CPU registers

The CPU core has thirty-two 64-bit general-purpose registers (GPR).

In addition, it provides the following special registers:

- PC: Program counter (64 bits)
- HI register: Contains the integer multiply and divide higher doubleword result (64 bits)
- LO register: Contains the integer multiply and divide lower doubleword result (64 bits)

Two of the general-purpose registers are assigned the following functions:

- r0 is fixed to 0, 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 register when a zero value is needed.
- r31 is the link register used by link instructions such as JAL (jump and link) instructions. This register can be used for other instructions. However, be careful that use of the register by a link instruction will not coincide with use of the register for other operations.

The register group is provided within the CP0 (system control coprocessor), to process exceptions and to manage addresses.

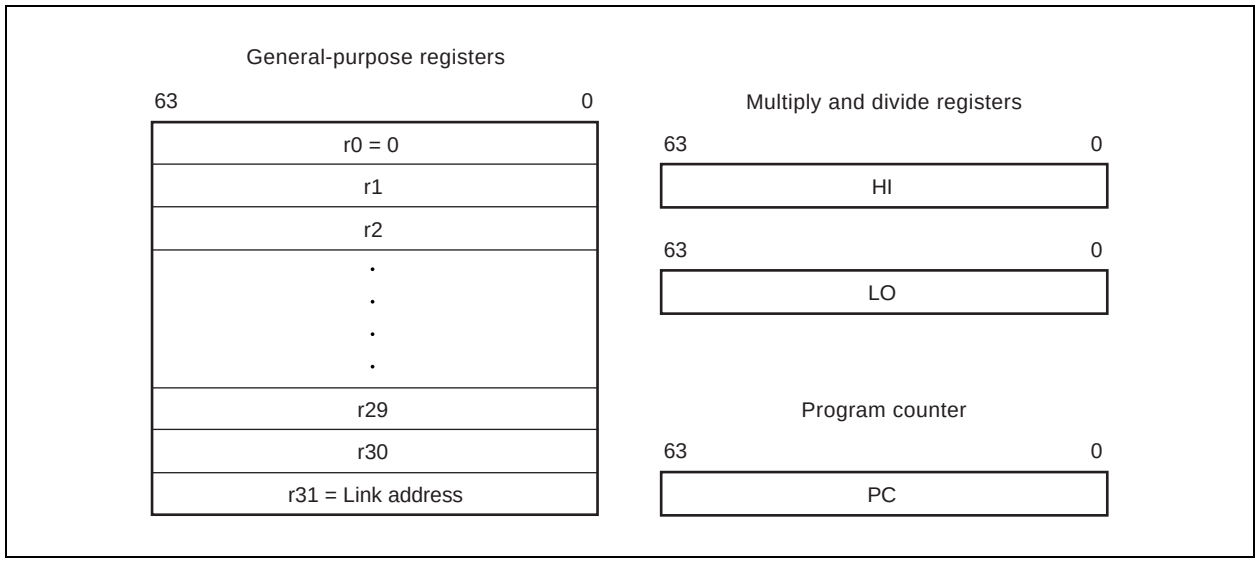
CPU registers can operate as either 32-bit or 64-bit registers, depending on the processor operation mode.

The operation of the CPU register differs depending on what instructions are executed: 32-bit instructions or MIPS16 instructions. For details, refer to **CHAPTER 3 MIPS16 INSTRUCTION SET**.

The VR4100 Series processors have no program status word (PSW) register as such; this is covered by the Status and Cause registers incorporated within the system control coprocessor (CP0). For details of CP0 registers, refer to **Table 1-2 CP0 Registers**.

Figure 1-2 shows the CPU registers.

Figure 1-2. CPU Registers



1.2.2 Coprocessors

MIPS ISA defines 4 types of coprocessors (CP0 to CP3).

- CP0 translates virtual addresses to physical addresses, switches the operating mode (Kernel, Supervisor, or User mode), and manages exceptions. It also controls the cache subsystem to analyze a cause and to return from the error state.
- CP1 is reserved for floating-point instructions.
- CP2 is reserved for future definition by MIPS.
- CP3 is no longer defined. CP3 instructions are reserved for future extensions.

The VR4100 Series implements the CP0 only.

1.2.3 System control coprocessor (CP0)

CP0 translates virtual addresses to physical addresses, switches the operating mode, controls the cache memory, and manages exceptions. For detailed descriptions of these functions, refer to **CHAPTER 5 MEMORY MANAGEMENT SYSTEM** and **CHAPTER 6 EXCEPTION PROCESSING**.

CP0 has thirty-two registers that have corresponding register number. The register number is used as an operand of instructions to specify a CP0 register to be accessed. Table 1-2 shows simple descriptions of each register.

Table 1-2. CP0 Registers

Register Number	Register Name	Usage	Description
0	Index	Memory management	Programmable pointer to TLB array
1	Random	Memory management	Pseudo-random pointer to TLB array (read only)
2	EntryLo0	Memory management	Lower half of TLB entry for even VPN
3	EntryLo1	Memory management	Lower half of TLB entry for odd VPN
4	Context	Exception processing	Pointer to virtual PTE table in 32-bit mode
5	PageMask	Memory management	Page size specification
6	Wired	Memory management	Number of wired TLB entries
7	–	–	Reserved for future use
8	BadVAddr	Exception processing	Virtual address where the most recent error occurred
9	Count	Exception processing	Timer count
10	EntryHi	Memory management	Upper half of TLB entry (including ASID)
11	Compare	Exception processing	Timer compare value
12	Status	Exception processing	Operation status
13	Cause	Exception processing	Cause of last exception
14	EPC	Exception processing	Exception program counter
15	PRId	Memory management	Processor revision identifier
16	Config	Memory management	Memory mode system specification
17	LLAddr[Note1]	Memory management	Physical address for diagnostic purpose
18	WatchLo	Exception processing	Memory reference trap address lower bits
19	WatchHi	Exception processing	Memory reference trap address higher bits
20	Xcontext	Exception processing	Pointer to virtual PTE table in 64-bit mode
21 to 25	–	–	Reserved for future use
26	Parity Error[Note2]	Exception processing	Cache parity bits
27	Cache Error[Note2]	Exception processing	Index and status of cache error
28	TagLo	Memory management	Cache tag register (low)
29	TagHi	Memory management	Cache tag register (high)
30	ErrorEPC	Exception processing	Error exception program counter
31	–	–	Reserved for future use

Notes 1. This register is defined to maintain compatibility with the VR4000™ and VR4400™. The contents of this register are meaningless in the normal operation.

2. This register is defined to maintain compatibility with the VR4100™. This register is not used in the normal operation.

Caution When accessing the CP0 registers, some instructions require consideration of the interval time until the next instruction is executed, because there is a delay from when the contents of the CP0 register change to when this change is reflected in the CPU operation. This time lag is called a CP0 hazard. For details, refer to CHAPTER 11 COPROCESSOR 0 HAZARDS.

1.2.4 Floating-point unit (FPU)

The VR4100 Series does not support the floating-point unit (FPU). A coprocessor unusable exception will occur if any FPU instructions are executed. If necessary, FPU instructions should be emulated by software in an exception handler.

1.2.5 Cache memory

The VR4100 Series incorporates instruction and data caches, which are independent of each other. This configuration enables high-performance pipeline operations. Both caches have a 64-bit data bus, enabling a one-clock access. These buses can be accessed in parallel.

The caches are managed with direct mapping format in the VR4121, VR4122, VR4181, and VR4181A, or with 2-way set-associative format in the VR4131. The data cache of the VR4131 has also the line lock function.

A detailed description of caches is given in **CHAPETER 7 CACHE MEMORY**.

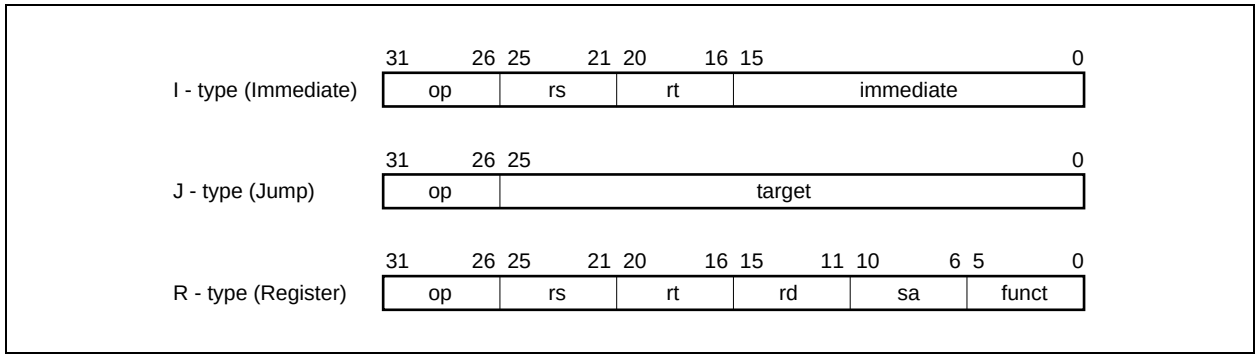
1.3 CPU Instruction Set Overview

There are two types of CPU instructions: 32-bit length instructions (MIPS III) and 16-bit length instructions (MIPS16). Use of the MIPS16 instructions is enabled or disabled by setting MIPS16EN pin during a reset.

(1) MIPS III instructions

All the CPU instructions are 32-bit length when executing MIPS III instructions, and they are classified into three instruction formats as shown in Figure 1-3: immediate (I type), jump (J type), and register (R type). The fields of each instruction format are described in **CHAPTER 2 CPU INSTRUCTION SET SUMMARY**.

Figure 1-3. CPU Instruction Formats (32-bit Length Instruction)



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

(a) Load and store instructions move data between the memory and the 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.
(b) Computational instructions perform arithmetic, logical, shift, and multiply and divide operations on values in registers. They include R-type (in which both the operands and the result are stored in registers) and I-type (in which one operand is a 16-bit signed immediate value) formats.
(c) Jump and branch instructions change the control flow of a program. Jumps are made either to an absolute address formed by combining a 26-bit target address with the higher bits of the program counter (J-type format) or register-specified address (R-type format). The format of the branch instructions is I type. Branches have 16-bit offsets relative to the program counter. JAL instructions save their return address in register 31.
(d) System control coprocessor (CP0) instructions perform operations on CP0 registers to control the memory-management and exception-handling facilities of the processor.
(e) Special instructions perform system calls and breakpoint exceptions, or cause a branch to the general exception-handling vector based upon the result of a comparison. These instructions occur in both R-type and I-type formats.

For the operation of each instruction, refer to **CHAPTER 2 CPU INSTRUCTION SET SUMMARY** and **CHAPTER 9 CPU INSTRUCTION SET DETAILS.**

(2) Additional instructions

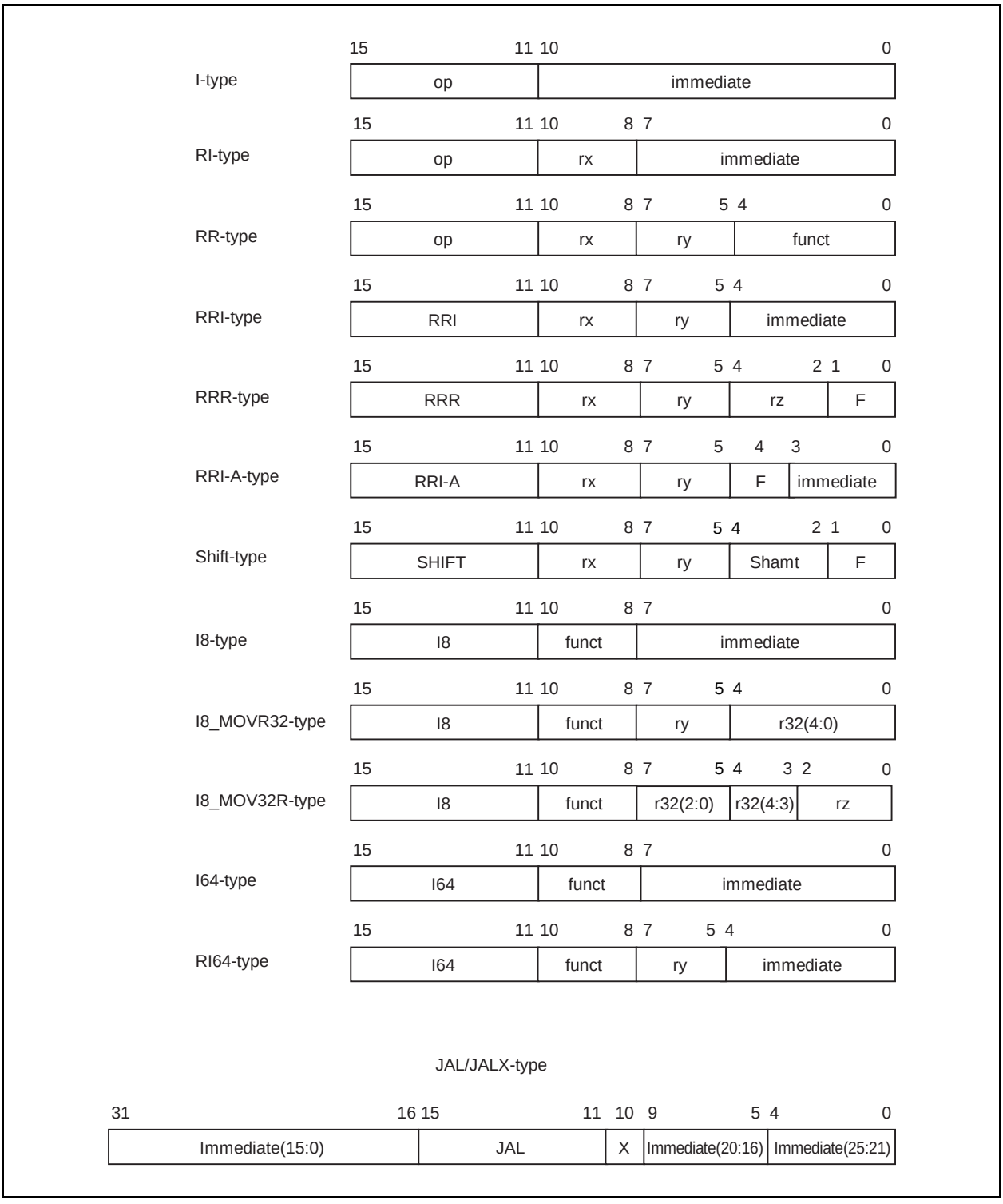
All the sum-of-products instructions and power mode instructions are 32-bit length.

(3) MIPS16 instructions

All the CPU instructions except for JAL and JALX are 16-bit length when executing MIPS16 instructions, and they are classified into thirteen instruction formats as shown in Figure 1-4.

The fields of each instruction format are described in **CHAPTER 3 MIPS 16 INSTRUCTION SET.**

Figure 1-4. CPU Instruction Formats (16-bit Length Instruction)



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

(a) Load and store instructions move data between memory and general-purpose registers. They include RRI, RI, I8, and RI64 types.
(b) Computational instructions perform arithmetic, logical, shift, and multiply and divide operations on values in registers. They include RI-, RRIA, I8, RI64, I64, RR, RRR, I8_MOVR32, and I8_MOV32R types.
(c) Jump and branch instructions change the control flow of a program. They include JAL/JALX, RR, RI, I8, and I types.
(d) Special instructions are SYSCALL, BREAK, and Extend instructions. The SYSCALL and BREAK instructions transfer control to an exception handler. The Extend instruction extends the immediate field of the next instruction. They are RR and I types. When extending the immediate field of the next instruction by using the Extend instruction, one cycle is needed for executing the Extend instruction, and another cycle is needed for executing the next instruction.

For more details of each instruction's operation, refer to **CHAPTER 3 MIPS16 INSTRUCTION SET** and **CHAPTER 10 MIPS16 INSTRUCTION SET FORMAT**.

1.4 Data Formats and Addressing

The VR4100 Series uses the following four data formats:

- Doubleword (64 bits)
- Word (32 bits)
- Halfword (16 bits)
- Byte (8 bits)

In the CPU core, if the data format is any one of halfword, word, or doubleword, the byte ordering can be set as either big endian or little endian. In the VR4131, the setting of BIGENDIAN pin during a reset decides which byte order is used. The VR4121, VR4122, VR4181, and VR4181A only support the little-endian order.

Endianness refers to the location of byte 0 within the multi-byte data structure. Figures 1-5 and 1-6 show the configuration.

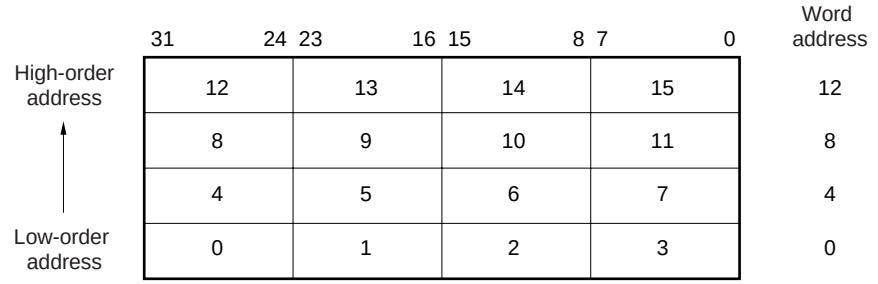
When configured as a big-endian system, byte 0 is always the most-significant (leftmost) byte, which is compatible with MC68000™ and IBM370™ conventions.

When configured as a little-endian system, byte 0 is always the least-significant (rightmost) byte, which is compatible with Pentium™ and DEC VAX™ conventions.

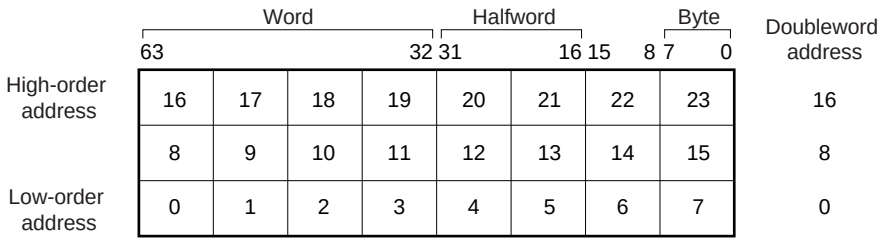
In this manual, bit designations are always little endian.

Figure 1-5. Byte Address in Big-Endian Byte Order

(a) Word data



(b) Doubleword data

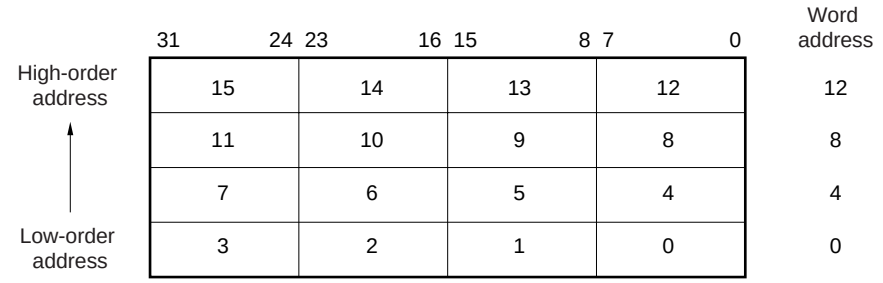


Remarks 1. The highest byte is the lowest address.

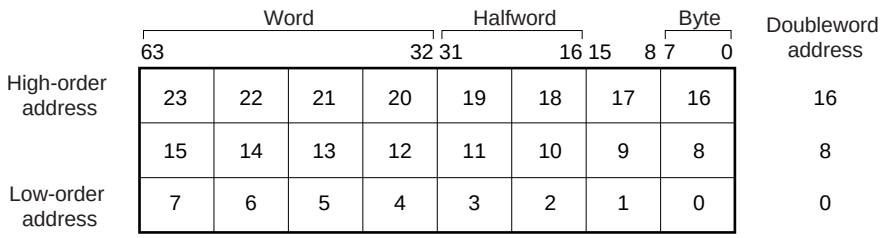
2. The address of word data is specified by the highest byte's address.

Figure 1-6. Byte Address in Little-Endian Byte Order

(a) Word data



(b) Doubleword data



Remarks 1. The lowest byte is the lowest address.

2. The address of word data is specified by the lowest byte's address.

The CPU core uses the following byte boundaries for halfword, word, and doubleword accesses:

- Halfword: An even byte boundary (0, 2, 4...)
- Word: A byte boundary divisible by four (0, 4, 8...)
- Doubleword: A byte boundary divisible by eight (0, 8, 16...)

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

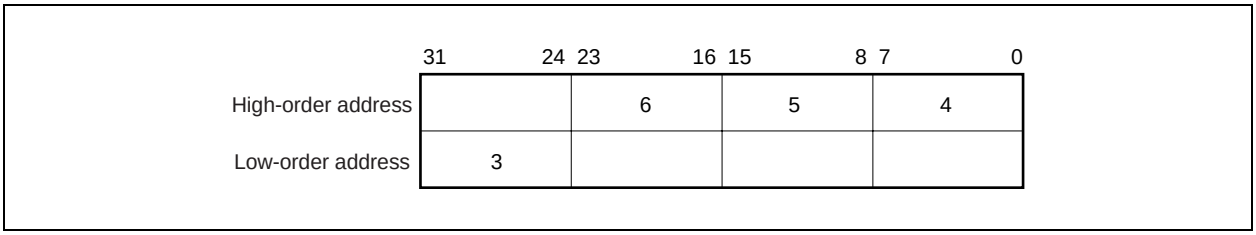
- Word access: LWL, LWR, SWL, SWR
- Doubleword access: LDL, LDR, SDL, SDR

These instructions are used in pairs of L and R.

Accessing misaligned data requires one additional instruction cycle (1 PCycle) over that required for accessing aligned data.

Figure 1-7 shows the access of a misaligned word that has byte address 3.

Figure 1-7. Misaligned Word Accessing (Little-Endian)



Caution In the VR4131, data transfer to the internal I/O (register) space or to the PCI bus is performed with data converted to little endian even during operation in big-endian mode. Therefore, the following restrictions apply for access to these address spaces.

- **Do not perform 3-byte access. When 3-byte access is executed, data is undefined.**
- **When 8-byte access is executed, the order of higher word and lower word is reversed.**
- **Do not use the LWR, LWL, LDR, and LDL instructions. Access by the LWR, LWL, LDR, or LDL instruction causes erroneous data to be loaded.**

1.5 Memory Management System

The VR4100 Series has a 32-bit physical addressing range of 4 GB. However, since it is rare for systems to implement a physical memory space as large as that memory space, the CPU provides a logical expansion of memory space by translating addresses composed in the large virtual address space into available physical memory addresses.

A detailed description of these address spaces is given in **CHAPTER 5 MEMORY MANAGEMENT SYSTEM**.

1.5.1 Translation lookaside buffer (TLB)

Virtual memory mapping is performed using the translation lookaside buffer (TLB). The TLB converts virtual addresses to physical addresses. It runs by a full-associative method and has 32 entries, each mapping a pair of two consecutive pages. The page size is variable between 1 KB and 256 KB, in powers of 4.

(1) Joint TLB (JTLB)

The JTLB holds both instruction and data addresses.

For fast virtual-to-physical address decoding, the VR4100 Series uses a large, fully associative TLB (joint TLB) that translates 64 virtual pages to their corresponding physical addresses. The TLB is organized as 32 pairs of even-odd entries, and maps a virtual address and address space identifier (ASID) into the 4 GB physical address space.

The page size can be configured, on a per-entry basis, to map a page size of 1 KB to 256 KB. A CP0 register stores the size of the page to be mapped, and that size is entered into the TLB when a new entry is written. Thus, operating systems can provide special purpose maps; for example, a typical frame buffer can be memory-mapped using only one TLB entry.

Translating a virtual address to a physical address begins by comparing the virtual address from the processor with the physical 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.

1.5.2 Processor modes

(1) Operating modes

The VR4100 Series has three operating modes, User, Supervisor, and Kernel. The manner in which memory addresses are mapped depends on these operating modes. Refer to **CHAPTER 5 MEMORY MANAGEMENT SYSTEM** for details.

(2) Addressing modes

The VR4100 Series has two addressing modes, 64-bit and 32-bit. The manner in which memory addresses are translated or mapped depends on these operating modes. Refer to **CHAPTER 5 MEMORY MANAGEMENT SYSTEM** for details.

1.6 Instruction Pipeline

The VR4100 Series has a 5- to 7-stage instruction pipeline.

In the VR4121, VR4122, VR4181, and VR4181A, one instruction is issued each cycle under normal circumstances.

The VR4131 employs a 2-way superscalar mechanism so that two instructions can be executed simultaneously.

A detailed description of the pipeline is provided in **CHAPTER 4 PIPELINE**.

1.6.1 Branch prediction

The VR4122, VR4131, and VR4181A have a branch prediction mechanism to speed up branch operations. These processors have a branch prediction table that holds branch instructions whose conditions were satisfied in the past, and the target addresses of the instructions. If an instruction that is the same as the fetched instruction is in this table (hit), execution branches without delay. If the corresponding branch instruction is not in the branch prediction table (miss), the address of that instruction is loaded to the branch prediction table and then execution branches. For the operations when a hit or a miss occurs, refer to **CHAPTER 4 PIPELINE**.

If the BP bit of the Config register of CP0 is cleared, branch prediction is performed. It is not performed if the BP bit is set (1) or in the MIPS16 instruction mode.

1.7 Code Compatibility

The CPU cores of the VR4100 Series are designed in consideration of the program compatibility to other VR-Series processors. However since they have some differences from other processors on their architecture, they cannot necessarily execute all programs that can be executed in other VR-Series processors, and also other VR-Series processors cannot necessarily execute all programs that can be executed in the VR4100 Series.

Matters that should be paid attention to when porting programs between the VR4100 Series and other VR-Series processors are listed below.

- A 16-bit length MIPS16 instruction set is added in the VR4100 Series.
- Multiply-add instructions are added in the VR4100 Series.
- Instructions for power modes (HIBERNATE, STANDBY, SUSPEND) are added in the VR4100 Series to support power modes.
- Operations to lock a cache are added to the CACHE instruction in the VR4131.
- The VR4100 Series does not support floating-point instructions since it has no Floating-Point Unit (FPU).
- The VR4100 Series does not have the LL bit to perform synchronization of multiprocessing. Therefore, it does not support instructions that manipulate the LL bit (LL, LLD, SC, SCD).
- The CP0 hazards of the VR4100 Series are equally or less stringent than those of the VR4000 (see Chapter 11 for details).

For more information about each instruction, refer to Chapters 9 and 3, and user's manuals of each product other than the VR4100 Series.

Instructions supported by each of the VR Series processors are listed below.

Table 1-3. List of Instructions Supported by VR Series Processors

Products / Supported instructions		VR4121 VR4122 VR4181 VR4181A	VR4131	VR4300 TM VR4305 TM VR4310 TM	VR5000 TM VR5000A TM	VR5432 TM VR5500 TM	VR10000TM VR12000TM
MIPS I		A	A	A	A	A	A
MIPS II		A	A	A	A	A	A
MIPS III		A	A	A	A	A	A
	LL bit manipulation	N/A	N/A	A	A	A	A
MIPS IV		N/A	N/A	N/A	A	A	A
MIPS16		A	A	N/A	N/A	N/A	N/A
Multiply-add		A	A	N/A	N/A	A	N/A
Floating-point operation		N/A	N/A	A	A	A	A
Power mode transition		A	A	N/A	A	A (VR5500)	N/A

CHAPTER 2 CPU INSTRUCTION SET SUMMARY

This chapter is an overview of the CPU instruction set; refer to **CHAPTER 9 CPU INSTRUCTION SET DETAILS** for detailed descriptions of individual CPU instructions.

2.1 Instruction Set Architecture

In the MIPS Instruction Set Architecture (ISA), five levels of instruction sets, from MIPS I through MIPS V, are currently defined. An instruction set of larger level number includes that of smaller level number. In other words, a processor implementing the MIPS IV instruction set is able to run MIPS I, MIPS II, or MIPS III binary programs without change.

There are another instruction sets called ASE, Application-Specific Extension, that extend functions for specific applications and MIPS16 is the one currently defined (refer to **CHAPTER 3 MIPS16 INSTRUCTION SET** for details).

The VR4100 Series implements MIPS III and MIPS16 instruction sets except for the following instructions:

(1) Synchronization support instructions

The VR4100 Series does not support a multiprocessor operating environment. Thus the instructions to support synchronization of memory update defined in the MIPS II and MIPS III ISA - the load linked and store conditional instructions - cause reserved instruction exception. The load link (LL) bit is eliminated.

Remark The SYNC instruction is handled as a NOP instruction since all load/store instructions in this processor are executed in program order.

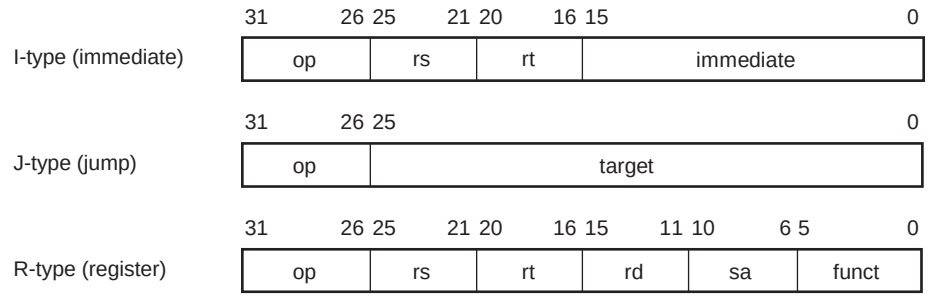
(2) Floating-point operation instructions

The VR4100 Series does not incorporate a floating-point unit (FPU). Thus the FPU instructions cause a coprocessor unusable exception. FPU instructions should be emulated by software in an exception handler if necessary.

2.2 CPU Instruction Formats

Each MIPS III ISA 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) - as shown in Figure 2-1. The use of a small number of instruction formats simplifies instruction decoding, allowing the compiler to synthesize more complicated and less frequently used instruction and addressing modes from these three formats as needed.

Figure 2-1. CPU Instruction Formats



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

2.3 Instructions Added in the VR4100 Series

In the VR4100 Series, instructions such as power mode instructions or product-sum operation instructions, which are suitable for potable information equipment and multimedia field, are added. These instructions are not included in the standard MIPS III instruction set.

2.3.1 Product-sum operation instructions

These instructions add a value in an accumulator to the result of multiplication and store it into a destination register, using the HI register and LO register as an accumulator. A 64-bit accumulator consists of the low-order 32 bits of the HI register as high-order bits and the low-order 32 bits of the LO register as low-order bits. No overflow or no underflow occurs by executing these instructions, and therefore, no exception occurs.

Of product-sum operation instructions, those that perform saturation processing or store data into a general-purpose register by specifying options are called MACC instructions.

Table 2-1. MACC Instructions (for VR4121, VR4122, VR4131, and VR4181A)

Instruction	Definition
MACC	Multiply and Add Accumulate
DMACC	Doubleword Multiply and Add Accumulate

Table 2-2. Product-Sum Operation Instructions (for VR4181)

Instruction	Definition
MADD16	Multiply and Add 16-bit Integer
DMADD16	Doubleowrd Multiply and Add 16-bit Integer

2.3.2 Power mode instructions

These instructions stop the internal clock of the processor and set the processor in a low power consumption mode. Three low power consumption modes are available, each of which can be set by a dedicated instruction.

Table 2-3. Power Mode Instructions

Instruction	Definition
STANDBY	Standby
SUSPEND	Suspend
HIBERNATE	Hibernate

2.4 Instruction Overview

The CPU instructions are classified into five classes. The product-sum operation instructions and power mode instructions added in the VR4100 Series are also included in one of the five classes.

2.4.1 Load and store instructions

Loads and stores are immediate (I-type) instructions that move data between memory and the general-purpose registers. The only addressing mode that load and store instructions directly support is base register plus 16-bit signed immediate offset.

Tables 2-5 and 2-6 list the ISA-defined load/store instructions and extended-ISA instructions, respectively.

(1) Scheduling a load delay slot

A load instruction that does not allow its result to be used by the instruction immediately following is called a delayed load instruction. The instruction slot immediately following this delayed load instruction is referred to as the load delay slot.

In the VR4100 Series, a load instruction can be followed directly by an instruction that accesses a register that is loaded by the load instruction. In this case, however, an interlock occurs for a necessary number of cycles. Any instruction can follow a load instruction, but the load delay slot should be scheduled appropriately for both performance and compatibility with the VR Series microprocessors. For detail, see **CHAPTER 4 PIPELINE**.

(2) Store delay slot

When a store instruction is writing data to a cache, the data cache is kept busy at the DC and WB stages. If an instruction (such as load) that follows directly the store instruction accesses the data cache in the DC stage, a hardware-driven interlock occurs. To overcome this problem, the store delay slot should be scheduled.

Table 2-4. Number of Delay Slot Cycles Necessary for Load and Store Instructions

Instruction	Necessary number of PCycles
Load	1
Store	1

(3) Defining access types

Access type indicates the size of a processor data item to be loaded or stored, set by the load or store instruction opcode. Access types and accessed byte are shown in Figure 2-2.

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

The access type, together with the three low-order bits of the address, defines the bytes accessed within the addressed doubleword (shown in Figure 2-2). Only the combinations shown in Figure 2-2 are permissible; other combinations cause address error exceptions.

Figure 2-2. Byte Specification Related to Load and Store Instructions

Access type (value)	Low-order address bits			Accessed byte (big-endian)								Accessed byte (little-endian)							
	2	1	0	63							0	63							0
Doubleword (7)	0	0	0	0	1	2	3	4	5	6	7	7	6	5	4	3	2	1	0
7-byte (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	
6-byte (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		
5-byte (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				
Triple byte (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							

Remark The big-endian order is supported by the VR4131 only.

Table 2-5. Load/Store Instruction

Instruction	Format and Description	op	base	rt	offset
Load Byte	LB rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The bytes of the memory location specified by the address are sign extended and loaded into register rt.				
Load Byte Unsigned	LBU rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The bytes of the memory location specified by the address are zero extended and loaded into register rt.				
Load Halfword	LH rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The halfword of the memory location specified by the address is sign extended and loaded to register rt.				
Load Halfword Unsigned	LHU rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The halfword of the memory location specified by the address is zero extended and loaded to register rt.				
Load Word	LW rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The word of the memory location specified by the address is sign extended and loaded to register rt. In the 64-bit mode, it is further sign extended to 64 bits.				
Load Word Left	LWL rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the word whose address is specified so that the address-specified byte is at the left-most position of the word. The result of the shift operation is merged with the contents of register rt and loaded to register rt. In the 64-bit mode, it is further sign extended to 64 bits.				
Load Word Right	LWR rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the word whose address is specified so that the address-specified byte is at the right-most position of the word. The result of the shift operation is merged with the contents of register rt and loaded to register rt. In the 64-bit mode, it is further sign extended to 64 bits.				
Store Byte	SB rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The least significant byte of register rt is stored to the memory location specified by the address.				
Store Halfword	SH rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The least significant halfword of register rt is stored to the memory location specified by the address.				
Store Word	SW rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The lower word of register rt is stored to the memory location specified by the address.				
Store Word Left	SWL rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the contents of register rt so that the left-most byte of the word is in the position of the address-specified byte. The result is stored to the lower word in memory.				
Store Word Right	SWR rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the contents of register rt so that the right-most byte of the word is in the position of the address-specified byte. The result is stored to the upper word in memory.				

Table 2-6. Load/Store Instruction (Extended ISA)

Instruction	Format and Description	op	base	rt	offset
Load Doubleword	LD rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The doubleword of the memory location specified by the address are loaded into register rt.				
Load Doubleword Left	LDL rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the double word whose address is specified so that the address-specified byte is at the left-most position of the double word. The result of the shift operation is merged with the contents of register rt and loaded to register rt.				
Load Doubleword Right	LDR rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the double word whose address is specified so that the address-specified byte is at the right-most position of the double word. The result of the shift operation is merged with the contents of register rt and loaded to register rt.				
Load Word Unsigned	LWU rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The word of the memory location specified by the address are zero extended and loaded into register rt				
Store Doubleword	SD rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. The contents of register rt are stored to the memory location specified by the address.				
Store Doubleword Left	SDL rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the right the contents of register rt so that the left-most byte of the double word is in the position of the address-specified byte. The result is stored to the lower doubleword in memory.				
Store Doubleword Right	SDR rt, offset (base) The offset is sign extended and then added to the contents of the register base to form the virtual address. Shifts to the left the contents of register rt so that the right-most byte of the double word is in the position of the address-specified byte. The result is stored to the upper doubleword in memory.				

2.4.2 Computational instructions

Computational instructions perform arithmetic, logical, and shift operations on values in registers. 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 are classified as:

(1) ALU immediate instructions
(2) Three-operand type instructions
(3) Shift instructions
(4) Multiply/divide instructions

In addition, product-sum operation instructions are added in the VR4100 Series.

To maintain data compatibility between the 64- and 32-bit modes, it is necessary to sign-extend 32-bit operands correctly. If the sign extension is not correct, the 32-bit operation result is meaningless.

Table 2-7. ALU Immediate Instruction

Instruction	Format and Description	op	rs	rt	immediate
Add Immediate	ADDI rt, rs, immediate The 16-bit immediate is sign extended and then added to the contents of register rs to form a 32-bit result. The result is stored into register rt. In the 64-bit mode, the operand must be sign extended. An exception occurs on the generation of 2's complement overflow.				
Add Immediate Unsigned	ADDIU rt, rs, immediate The 16-bit immediate is sign extended and then added to the contents of register rs to form a 32-bit result. The result is stored into register rt. In the 64-bit mode, the operand must be sign extended. No exception occurs on the generation of integer overflow.				
Set On Less Than Immediate	SLTI rt, rs, immediate The 16-bit immediate is sign extended and then compared to the contents of register rt treating both operands as signed integers. If rs is less than the immediate, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rt.				
Set On Less Than Immediate Unsigned	SLTIU rt, rs, immediate The 16-bit immediate is sign extended and then compared to the contents of register rt treating both operands as unsigned integers. If rs is less than the immediate, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rt.				
AND Immediate	ANDI rt, rs, immediate The 16-bit immediate is zero extended and then ANDed with the contents of the register. The result is stored into register rt.				
OR Immediate	ORI rt, rs, immediate The 16-bit immediate is zero extended and then ORed with the contents of the register. The result is stored into register rt.				
Exclusive OR Immediate	XORI rt, rs, immediate The 16-bit immediate is zero extended and then Ex-ORed with the contents of the register. The result is stored into register rt.				
Load Upper Immediate	LUI rt, immediate The 16-bit immediate is shifted left by 16 bits to set the lower 16 bits of word to 0. The result is stored into register rt. In the 64-bit mode, the operand must be sign extended.				

Table 2-8. ALU Immediate Instruction (Extended ISA)

Instruction	Format and Description	op	rs	rt	immediate
Doubleword Add Immediate	DADDI rt, rs, immediate The 16-bit immediate is sign extended to 64 bits and then added to the contents of register rs to form a 64-bit result. The result is stored into register rt. An exception occurs on the generation of integer overflow.				
Doubleword Add Immediate Unsigned	DADDIU rt, rs, immediate The 16-bit immediate is sign extended to 64 bits and then added to the contents of register rs to form a 64-bit result. The result is stored into register rt. No exception occurs on the generation of overflow.				

Table 2-9. Three-Operand Type Instruction

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Add	ADD rd, rs, rt The contents of registers rs and rt are added together to form a 32-bit result. The result is stored into register rd. In the 64-bit mode, the operand must be sign extended. An exception occurs on the generation of integer overflow.						
Add Unsigned	ADDU rd, rs, rt The contents of registers rs and rt are added together to form a 32-bit result. The result is stored into register rd. In the 64-bit mode, the operand must be sign extended. No exception occurs on the generation of integer overflow.						
Subtract	SUB rd, rs, rt The contents of register rt are subtracted from the contents of register rs. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. An exception occurs on the generation of integer overflow.						
Subtract Unsigned	SUBU rd, rs, rt The contents of register rt are subtracted from the contents of register rs. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended. No exception occurs on the generation of integer overflow.						
Set On Less Than	SLT rd, rs, rt The contents of registers rs and rt are compared, treating both operands as signed integers. If the contents of register rs is less than that of register rt, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rd.						
Set On Less Than Unsigned	SLTU rd, rs, rt The contents of registers rs and rt are compared treating both operands as unsigned integers. If the contents of register rs is less than that of register rt, the result is set to 1; otherwise, the result is set to 0. The result is stored to register rd.						
AND	AND rd, rt, rs The contents of register rs are logical ANDed with that of general register rt bit-wise. The result is stored to register rd.						
OR	OR rd, rt, rs The contents of register rs are logical ORed with that of general register rt bit-wise. The result is stored to register rd.						
Exclusive OR	XOR rd, rt, rs The contents of register rs are logical Ex-ORed with that of general register rt bit-wise. The result is stored to register rd.						
NOR	NOR rd, rt, rs The contents of register rs are logical NORed with that of general register rt bit-wise. The result is stored to register rd.						

Table 2-10. Three-Operand Type Instruction (Extended ISA)

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Doubleword Add	DADD rd, rt, rs The contents of register rs are added to that of register rt. The 64-bit result is stored into register rd. An exception occurs on the generation of integer overflow.						
Doubleword Add Unsigned	DADDU rd, rt, rs The contents of register rs are added to that of register rt. The 64-bit result is stored into register rd. No exception occurs on the generation of integer overflow.						
Doubleword Subtract	DSUB rd, rt, rs The contents of register rt are subtracted from that of register rs. The 64-bit result is stored into register rd. An exception occurs on the generation of integer overflow.						
Doubleword Subtract Unsigned	DSUBU rd, rt, rs The contents of register rt are subtracted from that of register rs. The 64-bit result is stored into register rd. No exception occurs on the generation of integer overflow.						

Table 2-11. Shift Instruction

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Shift Left Logical	SLL rd, rs, sa The contents of register rt are shifted left by sa bits and zeros are inserted into the emptied lower bits. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.						
Shift Right Logical	SRL rd, rs, sa The contents of register rt are shifted right by sa bits and zeros are inserted into the emptied higher bits. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.						
Shift Right Arithmetic	SRA rd, rt, sa The contents of register rt are shifted right by sa bits and the emptied higher bits are sign extended. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.						
Shift Left Logical Variable	SLLV rd, rt, rs The contents of register rt are shifted left and zeros are inserted into the emptied lower bits. The lower five bits of register rs specify the shift count. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.						
Shift Right Logical Variable	SRLV rd, rt, rs The contents of register rt are shifted right and zeros are inserted into the emptied higher bits. The lower five bits of register rs specify the shift count. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.						
Shift Right Arithmetic Variable	SRAV rd, rt, rs The contents of register rt are shifted right and the emptied higher bits are sign extended. The lower five bits of register rs specify the shift count. The 32-bit result is stored into register rd. In the 64-bit mode, the operand must be sign extended.						

Table 2-12. Shift Instruction (Extended ISA)

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Doubleword Shift Left Logical	DSLL rd, rs, sa The contents of register rt are shifted left by sa bits and zeros are inserted into the emptied lower bits. The 64-bit result is stored into register rd.						
Doubleword Shift Right Logical	DSRL rd, rs, sa The contents of register rt are shifted right by sa bits and zeros are inserted into the emptied higher bits. The 64-bit result is stored into register rd.						
Doubleword Shift Right Arithmetic	DSRA rd, rt, sa The contents of register rt are shifted right by sa bits and the emptied higher bits are sign extended. The 64-bit result is stored into register rd.						
Doubleword Shift Left Logical Variable	DSLLV rd, rt, rs The contents of register rt are shifted left and zeros are inserted into the emptied lower bits. The lower six bits of register rs specify the shift count. The 64-bit result is stored into register rd.						
Doubleword Shift Right Logical Variable	DSRLV rd, rt, rs The contents of register rt are shifted right and zeros are inserted into the emptied higher bits. The lower six bits of register rs specify the shift count. The 64-bit result is stored into register rd.						
Doubleword Shift Right Arithmetic Variable	DSRAV rd, rt, rs The contents of register rt are shifted right and the emptied higher bits are sign extended. The lower six bits of register rs specify the shift count. The 64-bit result is stored into register rd.						
Doubleword Shift Left Logical + 32	DSLL32 rd, rt, sa The contents of register rt are shifted left by 32 + sa bits and zeros are inserted into the emptied lower bits. The 64-bit result is stored into register rd.						
Doubleword Shift Right Logical + 32	DSRL32 rd, rt, sa The contents of register rt are shifted right by 32 + sa bits and zeros are inserted into the emptied higher bits. The 64-bit result is stored into register rd.						
Doubleword Shift Right Arithmetic + 32	DSRA32 rd, rt, sa The contents of register rt are shifted right by 32 + sa bits and the emptied higher bits are sign extended. The 64-bit result is stored into register rd.						

Table 2-13. Multiply/Divide Instructions

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Multiply	MULT rs, rt The contents of registers rt and rs are multiplied, treating both operands as 32-bit signed integers. The 64-bit result is stored into special registers HI and LO. In the 64-bit mode, the operand must be sign extended.						
Multiply Unsigned	MULTU rs, rt The contents of registers rt and rs are multiplied, treating both operands as 32-bit unsigned integers. The 64-bit result is stored into special registers HI and LO. In the 64-bit mode, the operand must be sign extended.						
Divide	DIV rs, rt The contents of register rs are divided by that of register rt, treating both operands as 32-bit signed integers. The 32-bit quotient is stored into special register LO, and the 32-bit remainder is stored into special register HI. In the 64-bit mode, the operand must be sign extended.						
Divide Unsigned	DIVU rs, rt The contents of register rs are divided by that of register rt, treating both operands as 32-bit unsigned integers. The 32-bit quotient is stored into special register LO, and the 32-bit remainder is stored into special register HI. In the 64-bit mode, the operand must be sign extended.						
Move from HI	MFHI rd The contents of special register HI are loaded into register rd.						
Move from LO	MFLO rd The contents of special register LO are loaded into register rd.						
Move to HI	MTHI rs The contents of register rs are loaded into special register HI.						
Move to LO	MTLO rs The contents of register rs are loaded into special register LO.						

Table 2-14. Multiply/Divide Instructions (Extended ISA)

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Doubleword Multiply	DMULT rs, rt The contents of registers rt and rs are multiplied, treating both operands as signed integers. The 128-bit result is stored into special registers HI and LO.						
Doubleword Multiply Unsigned	DMULTU rs, rt The contents of registers rt and rs are multiplied, treating both operands as unsigned integers. The 128-bit result is stored into special registers HI and LO.						
Doubleword Divide	DDIV rs, rt The contents of register rs are divided by that of register rt, treating both operands as signed integers. The 64-bit quotient is stored into special register LO, and the 64-bit remainder is stored into special register HI.						
Doubleword Divide Unsigned	DDIVU rs, rt The contents of register rs are divided by that of register rt, treating both operands as unsigned integers. The 64-bit quotient is stored into special register LO, and the 64-bit remainder is stored into special register HI.						

Table 2-15. Product-Sum Operation Instructions (for VR4121, VR4122, VR4131, and VR4181A)

Instruction	Format and Description	op	rs	rt	rd	funct
Multiply and Add Accumulate	MACC{h}{u}{s} rd, rs, rt The contents of registers rt and rs are multiplied, treating both operands as 32-bit signed integers. The result is added to the combined value of special registers HI and LO. The 64-bit result is stored into special registers HI and LO. If h=0, the same data as that stored in register LO is also stored in register rd; if h=1, the same data as that stored in register HI is also stored in register rd. If u is specified, the operand is treated as unsigned data. If s is specified, registers rs and rd are treated as a 16-bit value (32 bits sign- or zero-extended), and the value obtained by combining registers HI and LO is treated as a 32-bit value (64 bits sign- or zero-extended). Moreover, saturation processing is performed for the operation result in the format specified with u.					
Doubleword Multiply and Add Accumulate	DMACC{h}{u}{s} rd, rs, rt The contents of registers rt and rs are multiplied, treating both operands as 32-bit signed integers. The result is added to value of special register LO. The 64-bit result is stored into special register LO. If h=0, the same data as that stored in register LO is also stored in register rd; if h=1, undefined data is stored in register rd. If u is specified, the operand is treated as unsigned data. If s is specified, registers rs and rd are treated as a 16-bit value (32 bits sign- or zero-extended), and register LO is treated as a 32-bit value (64 bits sign- or zero-extended). Moreover, saturation processing is performed for the operation result in the format specified with u.					

Table 2-16. Product-Sum Operation Instructions (for VR4181)

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Multiply and Add 16-bit Integer	MADD16 rs, rt The contents of registers rt and rs are multiplied, treating both operands as 16-bit signed integers (by sign extending to 64 bits). The result is added to the combined value of special registers HI and LO. The 64-bit result is stored into special registers HI and LO.						
Doubleword Multiply and Add 16-bit Integer	DMADD16 rs, rt The contents of registers rt and rs are multiplied, treating both operands as 16-bit signed integers (by sign extending to 64 bits). The result is added to value of special register LO. The 64-bit result is stored into special register LO.						

MFHI and MFLO instructions after a multiply or divide instruction generate interlocks to delay execution of the next instruction, inhibiting the result from being read until the multiply or divide instruction completes.

Table 2-17 gives the number of processor cycles (PCycles) required to resolve interlock or stall between various multiply or divide instructions and a subsequent MFHI or MFLO instruction.

Table 2-17. Number of Stall Cycles in Multiply and Divide Instructions

Instruction	Number of instruction cycles
MULT	1
MULTU	1
DIV	35
DIVU	35
DMULT	4
DMULTU	4
DDIV	67
DDIVU	67
MACC	0
DMACC	0
MADD16	1
DMADD16	1

2.4.3 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 instruction (this is known as the instruction in the delay slot) always executes while the target instruction is being fetched from memory.

For instructions involving a link (such as JAL and BLTZAL), the return address is saved in register r31.

(1) Overview of jump instructions

Subroutine calls in high-level languages are usually implemented with J or JAL 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 a 32-bit or 64-bit absolute address.

Returns, dispatches, and cross-page jumps are usually implemented with the JR or JALR instructions. Both are R-type instructions that take the 32-bit or 64-bit byte address contained in one of the general-purpose registers.

Table 2-18. Jump Instructions

Instruction	Format and Description	op	target
Jump	J target The contents of 26-bit target address is shifted left by two bits and combined with the high-order four bits of the PC. The program jumps to this calculated address with a delay of one instruction.		
Jump and Link	JAL target The contents of 26-bit target address is shifted left by two bits and combined with the high-order four bits of the PC. The program jumps to this calculated address with a delay of one instruction. The address of the instruction following the delay slot is stored into r31 (link register).		

Instruction	Format and Description	op	target
Jump and Link Exchange	JALX target The contents of 26-bit target address is shifted left by two bits and combined with the high-order four bits of the PC. The program jumps to this calculated address with a delay of one instruction, and then the ISA mode bit is reversed. The address of the instruction following the delay slot is stored into r31 (link register).		

Instruction	Format and Description	op	rs	rt	rd	sa	funct
Jump Register	JR rs The program jumps to the address specified in register rs with a delay of one instruction.						
Jump snd Link Register	JALR rs, rd The program jumps to the address specified in register rs with a delay of one instruction. The address of the instruction following the delay slot is stored into rd.						

(2) Overview of branch instructions

A branch instruction has a PC-related signed 16-bit offset.

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

Calculation of the target address is performed at the RF stage and the EX stage of the instruction. The target instruction of the branch is fetched at the EX stage of the branch instruction.

If the branch condition does not meet in executing a Likely instruction, the instruction in its delay slot is nullified. For all other branch instructions, the instruction in its delay slot is unconditionally executed.

Table 2-19. Branch Instructions (1/2)

Instruction	Format and Description	op	rs	rt	offset
Branch on Equal	BEQ rs, rt, offset If the contents of register rs are equal to that of register rt, the program branches to the target address.				
Branch on Not Equal	BNE rs, rt, offset If the contents of register rs are not equal to that of register rt, the program branches to the target address.				
Branch on Less Than or Equal to Zero	BLEZ rs, offset If the contents of register rs are less than or equal to zero, the program branches to the target address.				
Branch on Greater Than Zero	BGTZ rs, offset If the contents of register rs are greater than zero, the program branches to the target address.				

Instruction	Format and Description	REGIMM	rs	sub	offset
Branch on Less Than Zero	BLTZ rs, offset If the contents of register rs are less than zero, the program branches to the target address.				
Branch on Greater Than or Equal to Zero	BGEZ rs, offset If the contents of register rs are greater than or equal to zero, the program branches to the target address.				
Branch on Less Than Zero and Link	BLTZAL rs, offset The address of the instruction that follows delay slot is stored to register r31 (link register). If the contents of register rs are less than zero, the program branches to the target address.				
Branch on Greater Than or Equal to Zero and Link	BGEZAL rs, offset The address of the instruction that follows delay slot is stored to register r31 (link register). If the contents of register rs are greater than or equal to zero, the program branches to the target address.				

Remark sub: Sub-operation code

Table 2-19. Branch Instructions (2/2)

Instruction	Format and Description	COP0	BC	br	offset
Branch on Coprocessor 0 True	BC0T offset Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate out the branch target address. If the conditional signal of the coprocessor 0 is true, the program branches to the target address with one-instruction delay.				
Branch on Coprocessor 0 False	BC0F offset Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate out the branch target address. If the conditional signal of the coprocessor 0 is false, the program branches to the target address with one-instruction delay.				

Remark BC: BC sub-operation code
br: branch condition identifier

Table 2-20. Branch Instructions (Extended ISA) (1/2)

Instruction	Format and Description	op	rs	rt	offset
Branch on Equal Likely	BEQL rs, rt, offset If the contents of register rs are equal to that of register rt, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				
Branch on Not Equal Likely	BNEL rs, rt, offset If the contents of register rs are not equal to that of register rt, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				
Branch on Less Than or Equal to Zero Likely	BLEZL rs, offset If the contents of register rs are less than or equal to zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				
Branch on Greater Than Zero	BGTZL rs, offset If the contents of register rs are greater than zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				

Table 2-20. Branch Instructions (Extended ISA) (2/2)

Instruction	Format and Description	REGIMM	rs	sub	offset
Branch on Less Than Zero Likely	BLTZL rs, offset If the contents of register rs are less than zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				
Branch on Greater Than or Equal to Zero Likely	BGEZL rs, offset If the contents of register rs are greater than or equal to zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				
Branch on Less Than Zero and Link Likely	BLTZALL rs, offset The address of the instruction that follows delay slot is stored to register r31 (link register). If the contents of register rs are less than zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				
Branch on Greater Than or Equal to Zero and Link Likely	BGEZALL rs, offset The address of the instruction that follows delay slot is stored to register r31 (link register). If the contents of register rs are greater than or equal to zero, the program branches to the target address. If the branch condition is not met, the instruction in the delay slot is discarded.				

Remark sub: Sub-operation code

Instruction	Format and Description	COP0	BC	br	offset
Branch on Coprocessor 0 True Likely	BC0TL offset Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate out the branch target address. If the conditional signal of the coprocessor 0 is true, the program branches to the target address with one-instruction delay. If the branch condition is not met, the instruction in the delay slot is discarded.				
Branch on Coprocessor 0 False Likely	BC0FL offset Adds the 16-bit offset (shifted left by two bits and sign extended to 32 bits) to the address of the instruction in the delay slot to calculate out the branch target address. If the conditional signal of the coprocessor 0 is false, the program branches to the target address with one-instruction delay. If the branch condition is not met, the instruction in the delay slot is discarded.				

Remark BC: BC sub-operation code
br: branch condition identifier

2.4.4 Special instructions

Special instructions generate software exceptions. Their formats are R-type (Syscall, Break). The Trap instruction is available only for the products that support the MIPS III instruction set or later. All the other instructions are available for all VR Series.

Table 2-21. Special Instructions

Instruction	Format and Description	SPECIAL	rs	rt	rd	sa	funct
Synchronize	SYNC Completes the load/store instruction executing in the current pipeline before the next load/store instruction starts execution.						
System Call	SYSCALL Generates a system call exception, and then transits control to the exception handling program.						
Breakpoint	BREAK Generates a break point exception, and then transits control to the exception handling program.						

Remark SYNC instruction is handled as a NOP instruction in the VR4100 Series.

Table 2-22. Special Instructions (Extended ISA) (1/2)

Instruction	Format and Description	SPECIAL	rs	rt	rd	sa	funct
Trap If Greater Than or Equal	TGE rs, rt The contents of register rs are compared with that of register rt, treating both operands as signed integers. If the contents of register rs are greater than or equal to that of register rt, an exception occurs.						
Trap If Greater Than or Equal Unsigned	TGEU rs, rt The contents of register rs are compared with that of register rt, treating both operands as unsigned integers. If the contents of register rs are greater than or equal to that of register rt, an exception occurs.						
Trap If Less Than	TLT rs, rt The contents of register rs are compared with that of register rt, treating both operands as signed integers. If the contents of register rs are less than that of register rt, an exception occurs.						
Trap If Less Than Unsigned	TLTU rs, rt The contents of register rs are compared with that of register rt, treating both operands as unsigned integers. If the contents of register rs are less than that of register rt, an exception occurs.						
Trap If Equal	TEQ rs, rt If the contents of registers rs and rt are equal, an exception occurs.						
Trap If Not Equal	TNE rs, rt If the contents of registers rs and rt are not equal, an exception occurs.						

Table 2-22. Special Instructions (Extended ISA) (2/2)

Instruction	Format and Description	REGIMM	rs	sub	immediate
Trap If Greater Than or Equal Immediate	TGEI rs, immediate The contents of register rs are compared with 16-bit sign-extended immediate data, treating both operands as signed integers. If the contents of register rs are greater than or equal to 16-bit sign-extended immediate data, an exception occurs.				
Trap If Greater Than or Equal Immediate Unsigned	TGEIU rs, immediate The contents of register rs are compared with 16-bit zero-extended immediate data, treating both operands as unsigned integers. If the contents of register rs are greater than or equal to 16-bit sign-extended immediate data, an exception occurs.				
Trap If Less Than Immediate	TLTI rs, immediate The contents of register rs are compared with 16-bit sign-extended immediate data, treating both operands as signed integers. If the contents of register rs are less than 16-bit sign-extended immediate data, an exception occurs.				
Trap If Less Than Immediate Unsigned	TLTIU rs, immediate The contents of register rs are compared with 16-bit zero-extended immediate data, treating both operands as unsigned integers. If the contents of register rs are less than 16-bit sign-extended immediate data, an exception occurs.				
Trap If Equal Immediate	TEQI rs, immediate If the contents of register rs and immediate data are equal, an exception occurs.				
Trap If Not Equal Immediate	TNEI rs, immediate If the contents of register rs and immediate data are not equal, an exception occurs.				

Remark sub: Sub-operation code

2.4.5 System control coprocessor (CP0) instructions

System control coprocessor (CP0) instructions perform operations specifically on the CP0 registers to manipulate the memory management and exception handling facilities of the processor.

The power mode instructions added in the VR4100 Series are included in this instruction group.

Table 2-23. System Control Coprocessor (CP0) Instructions (1/2)

Instruction	Format and Description	COP0	sub	rt	rd	0
Move to System Control Coprocessor	MTC0 rt, rd The word data of general-purpose register rt in the CPU are loaded into general-purpose register rd in the CP0.					
Move from System Control Coprocessor	MFC0 rt, rd The word data of general-purpose register rd in the CP0 are loaded into general-purpose register rt in the CPU.					
Doubleword Move to System Control Coprocessor 0	DMTC0 rt, rd The doubleword data of general-purpose register rt in the CPU are loaded into general-purpose register rd in the CP0.					
Doubleword Move from System Control Coprocessor 0	DMFC0 rt, rd The doubleword data of general-purpose register rd in the CP0 are loaded into general-purpose register rt in the CPU.					

Remark sub: Sub-operation code

Table 2-23. System Control Coprocessor (CP0) Instructions (2/2)

Instruction	Format and Description	COP0	CO	funct
Read Indexed TLB Entry	TLBR The TLB entry indexed by the Index register is loaded into the EntryHi, EntryLo0, EntryLo1, or PageMask register.			
Write Indexed TLB Entry	TLBWI The contents of the EntryHi, EntryLo0, EntryLo1, or PageMask register are loaded into the TLB entry indexed by the Index register.			
Write Random TLB Entry	TLBWR The contents of the EntryHi, EntryLo0, EntryLo1, or PageMask register are loaded into the TLB entry indexed by the Random register.			
Probe TLB For Matching Entry	TLBP The address of the TLB entry that matches with the contents of EntryHi register is loaded into the Index register.			
Return From Exception	ERET The program returns from exception, interrupt, or error trap.			

Remark CO: Sub-operation identifier

Instruction	Format and Description	COP0	CO	funct
STANDBY	STANDBY The processor's operating mode is transited from Fullspeed mode to Standby mode.			
SUSPEND	SUSPEND The processor's operating mode is transited from Fullspeed mode to Suspend mode.			
HIBERNATE	HIBERNATE The processor's operating mode is transited from Fullspeed mode to Hibernate mode.			

Remark CO: Sub-operation identifier

Instruction	Format and Description	CACHE	base	op	offset
Cache Operation	Cache op, offset (base) The 16-bit offset is sign extended to 32 bits and added to the contents of the register base, to form virtual address. This virtual address is translated to physical address with TLB. For this physical address, cache operation that is indicated by 5-bit sub-opcode is performed.				

CHAPTER 3 MIPS16 INSTRUCTION SET

3.1 Outline

If the MIPS16 ASE (Application-Specific Extension), which is an expanded function for MIPS ISA (Instruction Set Architecture), is used, system costs can be considerably reduced by lowering the memory capacity requirement of embedded hardware. MIPS16 is an instruction set that uses the 16-bit instruction length, and is compatible with MIPS I, II, III, IV, and V[Note] instruction sets in any combination. Moreover, existing 32-bit instruction length binary data can be executed with MIPS16 without change.

Note The VR4100 Series currently supports the MIPS I, II, and III instruction sets.

MIPS16 instruction set is enabled or disabled in the VR4100 Series according to the state of MIPS16EN pin during a reset.

3.2 Features

- 16-bit length instruction format
- Reduces memory capacity requirements to lower overall system cost
- MIPS16 instructions can be used with MIPS instruction binary
- Compatibility with MIPS I, II, III, IV, and V instruction sets
- Used with switching between MIPS16 instruction length mode and 32-bit MIPS instruction length mode.
- Supports 8-bit, 16-bit, 32-bit, and 64-bit data formats
- Provides 8 general-purpose registers and special registers
- Improved code generation efficiency using special 16-bit dedicated instructions

3.3 Register Set

Tables 3-1 and 3-2 show the MIPS16 register sets. These register sets form part of the register sets that can be accessed in 32-bit instruction length mode. MIPS16 instructions can directly access 8 of the 32 registers that can be used in the 32-bit instruction length mode.

In addition to these 8 general-purpose registers, the special instructions of MIPS16 reference the stack pointer register (sp), return address register (ra), condition code register (t8), and program counter (pc). sp and ra are mapped by fixing to the general-purpose registers in the 32-bit instruction length mode.

MIPS16 has 2 move instructions that are used in addressing 32 general-purpose registers.

Table 3-1. General-purpose Registers

MIPS16 register encoding	32-bit MIPS register encoding	Symbol	Comment
0	16	s0	General-purpose register
1	17	s1	General-purpose register
2	2	v0	General-purpose register
3	3	v1	General-purpose register
4	4	a0	General-purpose register
5	5	a1	General-purpose register
6	6	a2	General-purpose register
7	7	a3	General-purpose register
N/A	24	t8	MIPS16 condition code register. BTEQZ, BTNEZ, CMP, CMPI, SLT, SLTU, SLTI, and SLTIU instructions are implicitly referenced.
N/A	29	sp	Stack pointer register
N/A	31	ra	Return address register

Remarks 1. The symbols are the general assembler symbols.

2. The MIPS register encoding numbers 0 to 7 correspond to the MIPS16 binary encoding of the registers, and are used to show the relationship between this encoding and the MIPS registers. The numbers 0 to 7 are not used to reference registers, except within binary MIPS16 instructions. Registers are referenced from the assembler using the MIPS name ($16, $17, $2, etc.) or the symbol name (s0, s1, v0, etc.). For example, when register number 17 is accessed with the register file, the programmer references either $17 or s1 even if the MIPS16 encoding of this register is 001.

3. The general-purpose registers not shown in this table cannot be accessed with a MIPS16 instruction set other than the Move instruction. The Move instruction of MIPS16 can access all 32 general-purpose registers.

4. To reference the MIPS16 condition code registers with this manual, either T, t8, or $24 has to be used, depending on the case. These three names reference the same physical register.

Table 3-2. Special Registers

Symbol	Description
PC	Program counter. The PC-relative Add instruction and Load instruction can access this register.
HI	The upper word of the multiply or divide result is inserted
LO	The lower word of the multiply or divide result is inserted

3.4 ISA Mode

MIPS16 instruction set supports procedure calling, and returns from the MIPS16 instruction mode or the 32-bit instruction length mode to the MIPS16 instruction mode or the 32-bit instruction length mode.

- The JAL instruction supports calling to the same ISA.
- The JALX instruction supports calling that inverses ISA.
- The JALR instruction supports calling to either ISA.
- The JR instruction supports also returning to either ISA.

MIPS16 instruction set also supports a return operation from exception processing.

- The ERET instruction, which is defined only in 32-bit instruction length mode, supports returning to ISA when an exception has not occurred.

The ISA mode bit defines the instruction length mode to be executed. If the ISA mode bit is 0, the processor executes only 32-bit instructions. If the ISA mode bit is 1, the processor executes only MIPS16 instructions.

3.4.1 Changing ISA mode bit by software

Only the JALX, JR, and JALR instructions change the ISA mode bit between the MIPS16 instruction mode and the 32-bit instruction length mode. The ISA mode bit cannot be directly overwritten by software. The JALX changes the ISA mode bit to select another ISA mode. The JR instruction and JALR instruction load the ISA mode bit from bit 0 of the general-purpose register that holds the target address. Bit 0 is not a part of the target address. Bit 0 of the target address is always 0, and no address exception is generated.

Moreover, the JAL, JALR, and JALX instructions save the ISA mode bit to bit 0 of the general-purpose register that acquires the return address. The contents of this general-purpose register are later used by the JR and JALR instruction for return and restoration of the ISA mode.

3.4.2 Changing ISA mode bit by exception

Even if an exception occurs, the ISA mode does not change. When an exception occurs, the ISA mode bit is cleared to 0 so that the exception is serviced with 32-bit code. Then the ISA mode status before the exception occurred is saved to the least significant bit of the EPC register or the ErrorEPC register. During return from an exception, the ISA mode before the exception occurred is returned to by executing the JR or ERET instruction with the contents of this register. Moreover, the ISA mode bit is cleared to 0 after cold reset and soft reset of the CPU core, and the 32-bit instruction length mode returns to its initial state.

3.4.3 Enabling change ISA mode bit

Changing the ISA mode bit is valid only when MIPS16EN pin is set to active during the RTC reset, and the MIPS16 instruction mode is enabled. The operation of the JALX, JALR, JR, and ERET instructions in the 32-bit instruction mode, differs depending on whether the MIPS16 instruction mode is enabled or prohibited. If the MIPS16 instruction mode is prohibited, the JALX instruction generates a reserved instruction exception. The JR and JALR instructions generate an address exception when bit 0 of the source register is 1. The ERET instruction generates an address exception when bit 0 of the EPC or ErrorEPC register is 1. If the MIPS16 instruction mode is enabled, the JALX instruction executes JAL, and the ISA mode bit is inverted. The JR and JALR instructions load the ISA mode from bit 0 of the source register. The ERET instruction loads the ISA mode from bit 0 of the EPC or ErrorEPC register. Bit 0 of the target address is always 0, and no address exception is generated even when bit 0 of the source register is 1.

3.5 Types of Instructions

This section describes the different types of instructions, and indicates the MIPS16 instructions included in each group.

Instructions are divided into the following types.

Load and Store instructions : Move data between memory and the general-purpose registers.
Computational instructions : Perform arithmetic operations, logical operations, and shift operations on values in registers.
Jump and Branch instructions: Change the control flow of a program.
Special instructions : SYSCALL, BREAK, and Extend instructions. SYSCALL and BREAK transfer control to an exception handler. Extend enlarges the immediate field of the next instruction. Instructions that can be extended with Extend are indicated as **Note 1** in **Table 3-3 MIPS16 Instruction Set Outline**.

Table 3-3. MIPS16 Instruction Set Outline (1/2)

Op	Description	Op	Description
Load and Store instructions		Multiply/Divide instructions	
LB[Note 1]	Load Byte	MULT	Multiply
LBU[Note 1]	Load Byte Unsigned	MULTU	Multiply Unsigned
LH[Note 1]	Load Halfword	DIV	Divide
LHU[Note 1]	Load Halfword Unsigned	DIVU	Divide Unsigned
LW[Note 1]	Load Word	MFHI	Move From HI
LWU[Notes 1, 2]	Load Word Unsigned	MFLO	Move From LO
LD[Notes 1, 2]	Load Doubleword	DMULT[Note 2]	Doubleword Multiply
SB[Note 1]	Store Byte	DMULTU[Note 2]	Doubleword Multiply Unsigned
SH[Note 1]	Store Halfword	DDIV[Note 2]	Doubleword Divide
SW[Note 1]	Store Word	DDIVU[Note 2]	Doubleword Divide Unsigned
SD[Notes 1, 2]	Store Doubleword		

Notes 1. Extendable instruction. For details, see **3.8.2 Extend instruction**.
2. Can be used in 64-bit mode and 32-bit Kernel mode.

Table 3-3. MIPS16 Instruction Set Outline (2/2)

Op	Description	Op	Description
Arithmetic instructions: ALU immediate instructions		Jump/Branch instructions	
LI[Note 1]	Load Immediate	JAL	Jump and Link
ADDIU[Note 1]	Add Immediate Unsigned	JALX	Jump and Link Exchange
DADDIU[Notes 1, 2]	Doubleword Add Immediate Unsigned	JR	Jump Register
SLTI[Note 1]	Set on Less Than Immediate	JALR	Jump and Link Register
SLTIU[Note 1]	Set on Less Than Immediate Unsigned	BEQZ[Note 1]	Branch on Equal to Zero
CMPI[Note 1]	Compare Immediate	BNEZ[Note 1]	Branch on Not Equal to Zero
		BTEQZ[Note 1]	Branch on T Equal to Zero
Arithmetic instructions: 2/3 operand register instructions		BTNEZ[Note 1]	Branch on T Not Equal to Zero
ADDU	Add Unsigned	B[Note 1]	Branch Unconditional
SUBU	Subtract Unsigned		
DADDU[Note 2]	Doubleword Add Unsigned	Shift instructions	
DSUBU[Note 2]	Doubleword Subtract Unsigned	SLL[Note 1]	Shift Left Logical
SLT	Set on Less Than	SRL[Note 1]	Shift Right Logical
SLTU	Set on Less Than Unsigned	SRA[Note 1]	Shift Right Arithmetic
CMP	Compare	SLLV	Shift Left Logical Variable
NEG	Negate	SRLV	Shift Right Logical Variable
AND	AND	SRAV	Shift Right Arithmetic Variable
OR	OR	DSLL[Notes 1, 2]	Doubleword Shift Left Logical
XOR	Exclusive OR	DSRL[Notes 1, 2]	Doubleword Shift Right Logical
NOT	Not	DSRA[Notes 1, 2]	Doubleword Shift Right Arithmetic
MOVE	Move	DSLLV[Note 2]	Doubleword Shift Left Logical Variable
		DSRLV[Note 2]	Doubleword Shift Right Logical Variable
Special instructions		DSRAV[Note 2]	Doubleword Shift Right Arithmetic Variable
EXTEND	Extend		
BREAK	Breakpoint		
SYCALL	System Call		

Notes 1. Extendable instruction. For details, see **3.8.2 Extend instruction**.

2. Can be used in 64-bit mode and 32-bit Kernel mode.

3.6 Instruction Format

The MIPS16 instruction set has a length of 16 bits and is located at the half-word boundary. One part of Jump instructions and instructions for which the Extend instruction extends immediate become 32 bits in length, but crossing the word boundary does not represent a problem.

The instruction format is shown below. Variable subfields are indicated with lower case letters (rx, ry, rz, immediate, etc.).

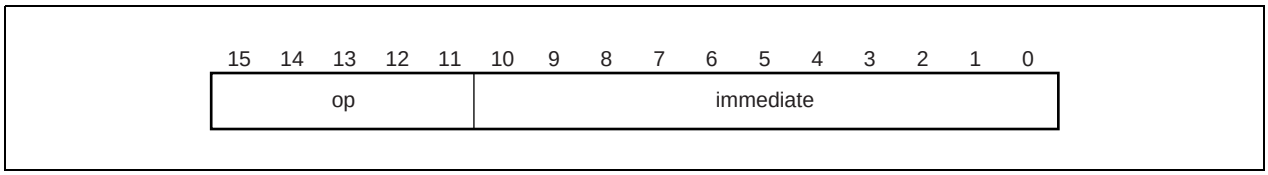
In the case of special functions, constants are input to the two instruction subfields op and funct. These values are indicated by upper case mnemonics. For example, in the case of the Load Byte instruction, op is LB, and in the case of the Add instruction, op is SPECIAL, and function is ADD.

The constants of the fields used in the instruction formats are shown below.

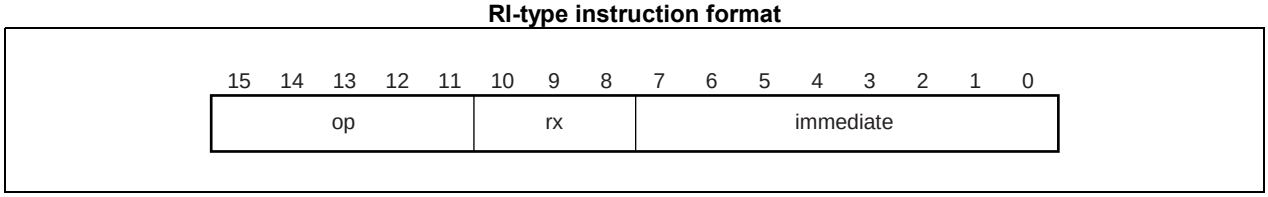
Table 3-4. Field Definition

Field	Definition
op	5-bit major operation code
rx	3-bit source/destination register specification
ry	3-bit source/destination register specification
immediate or imm	4-bit, 5-bit, 8-bit, or 11-bit immediate value, branch displacement, or address displacement
rz	3-bit source/destination register specification
Funct or F	Function field

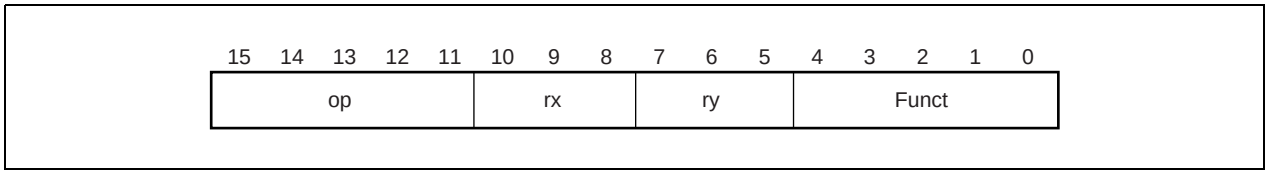
I-type (immediate) instruction format



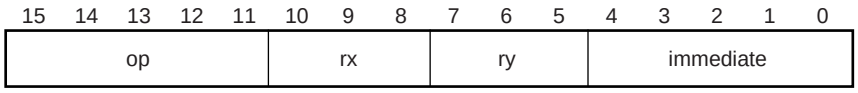
RI-type instruction format



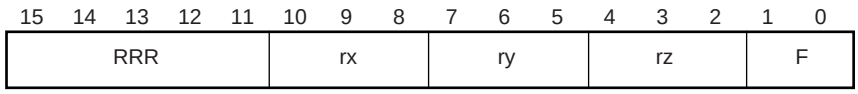
RR-type instruction format



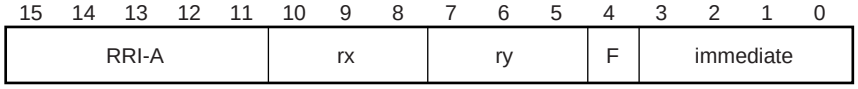
RRI-type instruction format



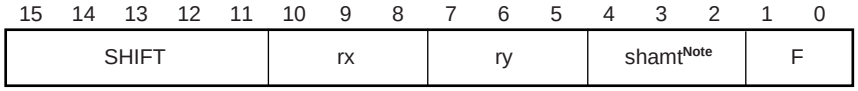
RRR-type instruction format



RRI-A type instruction format

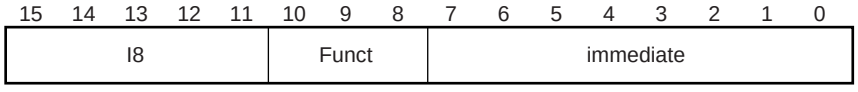


SHIFT instruction format

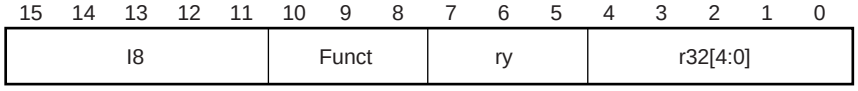


Note The 3-bit shamt field can encode shift count numbers from 0 to 7. 0-bit shift (NOP) cannot be executed. 0 is regarded as shift count 8.

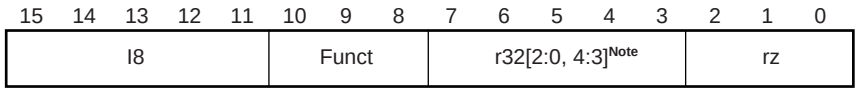
I8-type instruction format



I8_MOVR32 instruction format (used only with MOVR32 instruction)

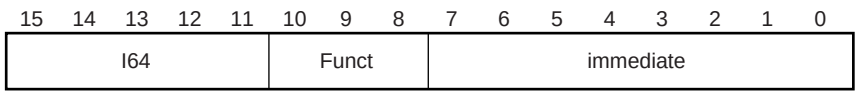


I8_MOV32R instruction format (used only with MOV32R instruction)

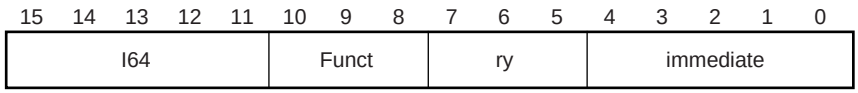


Note The r32 field uses special bit encoding. For example, encoding of $7 (00111) is 11100 in the r32 field.

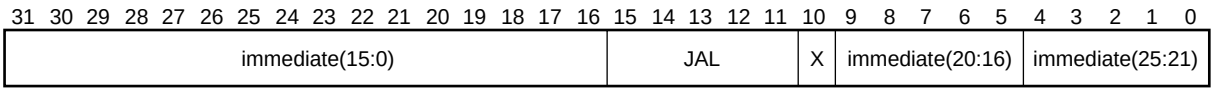
I64-type instruction format



RI64-type instruction format



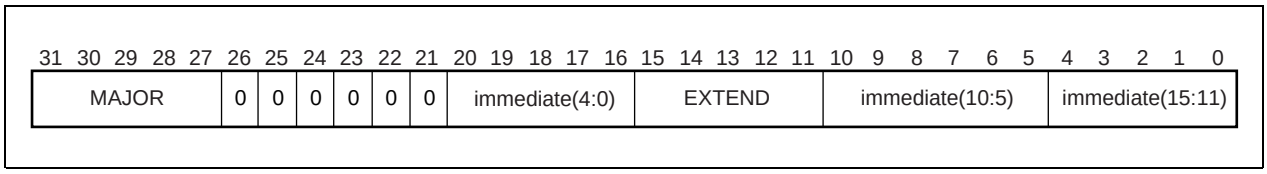
JAL and JALX instruction format



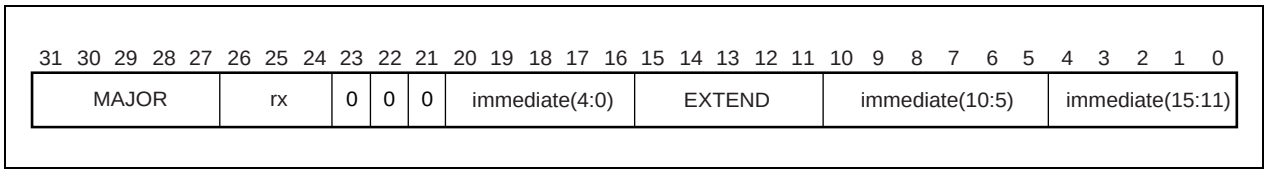
JAL in case of X = 0 instruction

JALX in case of X = 1 instruction

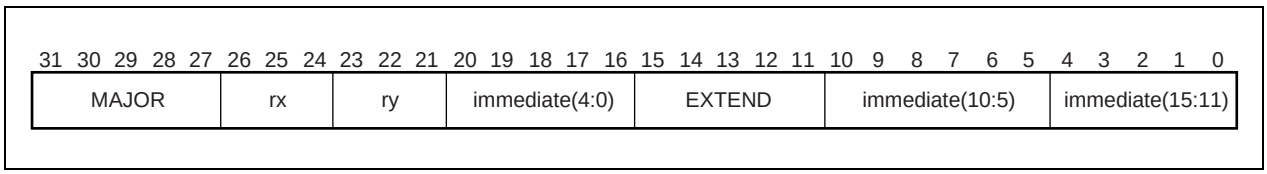
EXT-I instruction format



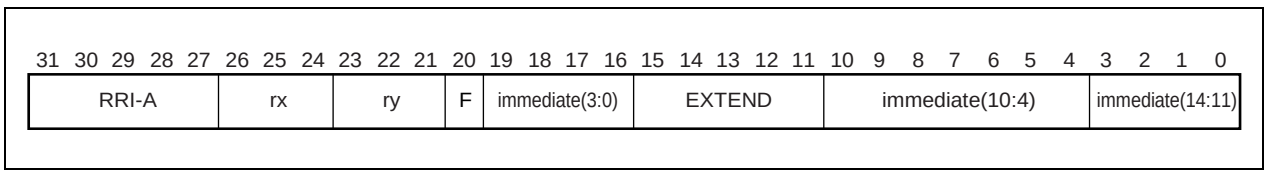
EXT-RI instruction format



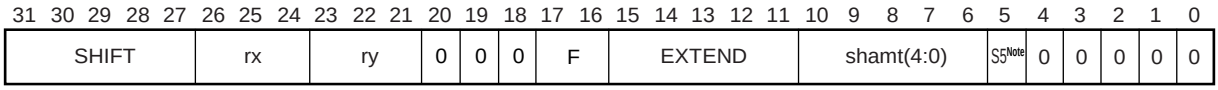
EXT-RRI instruction format



EXT-RRI-A instruction format

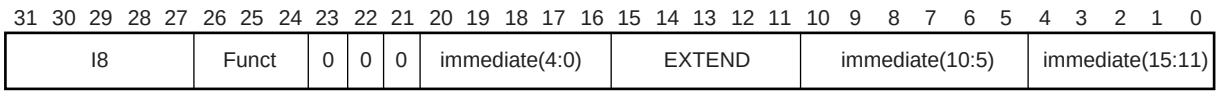


EXT-SHIFT instruction format

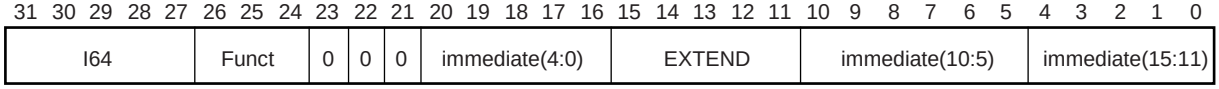


Note Only in the case of DSLL, the S5 bit is the most significant bit of the 6-bit shift count field (shamt). In the case of all 32-bit extended shifts, S5 must be 0. For a normal shift instruction, the display of shift count 0 is considered as shift count 8, but the extended shift instruction does not perform such mapping changes. Therefore, 0-bit shift using the extended format is possible.

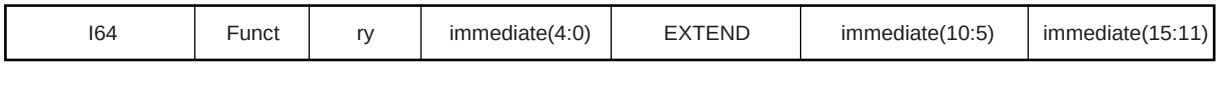
EXT-I8 instruction format



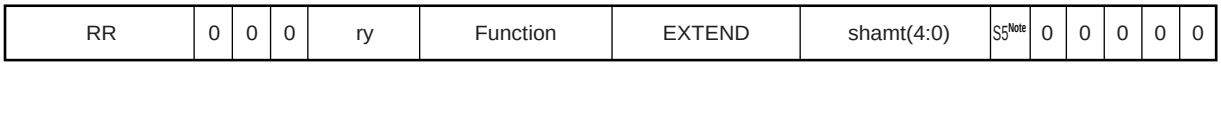
EXT-I64 instruction format



EXT-RI64 instruction format



EXT-SHIFT64 instruction format



Note The S5 bit is the most significant bit of the 6-bit shift count field (shamt). In the case of a normal shift instruction, the display of shift count 0 is considered as shift count 8, but the extended shift instruction does not perform such mapping changes.
Therefore, 0-bit shift using the extended format is possible.

3.7 MIPS16 Operation Code Bit Encoding

This section describes encoding for major and minor opcode. Table 3-5 shows bit encoding of the MIPS16 major operation code. Tables 3-6 to 3-11 show bit encoding of the minor operation code. The italic operation codes in the tables are instructions for the extended ISA.

Table 3-5. Bit Encoding of Major Operation Code (op)

Instruction bits	Instruction bits [13:11]							
[15:14]	000	001	010	011	100	101	110	111
00	addiusp[Note 1]	addiupc[Note 2]	b	jal(x)[Note 3]	beqz	bnez	SHIFT	*ld*
01	RRI-A	addiu8[Note 4]	slti	sltiu	I8	li	cmpi	*sd*
10	lb	lh	lwsp	lw	lbu	lhu	lwpc	*lwu*
11	sb	sh	swsp	sw	RRR	RR	extend	*l64*

Notes 1. addiusp : addiu rx, sp, immediate
2. addiupc : addiu rx, pc, immediate
3. jal(x) : jal instruction and jalx instruction
4. addiu8 : aadiu rx, immediate

Table 3-6. RR Minor Operation Code (RR-Type Instruction)

Instruction bits	Instruction bits [2:0]							
[4:3]	000	001	010	011	100	101	110	111
00	j(al)r[Note 1]	*	slt	sltu	sllv	break	srlv	srav
01	*dsrl*[Note 2]	syscall	cmp	neg	and	or	xor	not
10	Mfhi	*	mflo	*dsra*[Note 2]	*dsllv*	*	*dsrlv*	*dsrav*
11	mult	Multu	div	divu	*dmult*	*dmultu*	*ddiv*	*ddivu*

Notes 1. J(al)r: jr rx instruction (ry = 000)
jr ra instruction (ry = 001, rx = 000)
jalr ra, rx instruction (ry = 010)
2. dsrl and dsra use the rx register field to encode the shift count (8-digit shift for 0). In the case of the extended version of these two instructions, the EXT-SHIFT64 format is used. Only these two RR instructions can be extended.

Remarks The symbols in the figures have the following meaning.
* : Execution of operation code with an asterisk on the current VR4100 Series causes a reserved instruction exception to be generated. This code is reserved for future extension.

Table 3-7. RRR Minor Operation Code (RRR-Type Instruction)

Instruction bits [1:0]			
00	01	10	11
daddu	addu	*dsubu*	subu

Table 3-8. RRI-A Minor Operation Code (RRI-Type ADD Instruction)

Instruction bit [4]	
0	1
addiu[Note 1]	daddiu[Note 2]

Notes 1. addiu : addiu ry, rx, immediate
2. daddiu : daddiu ry, rx immediate

Table 3-9. SHIFT Minor Operation Code (SHIFT-Type Instruction)

Instruction bits [1:0]			
00	01	10	11
sll	*Dsll*	srl	sra

Table 3-10. I8 Minor Operation Code (I8-Type Instruction)

Instruction bits [10:8]							
000	001	010	011	100	101	110	111
bteqz	btnez	swrasp[Note 1]	adjsp[Note 2]	*	mov32r[Note 3]	*	movr32[Note 4]

Notes 1. swrasp : sw ra, immediate(sp)
2. adjsp : addiu sp, immediate
3. mov32r : move r32, rz
4. movr32 : move ry, r32

Remark The symbols used in the figures have the following meaning.
* : Execution of operation code with an asterisk on the current VR4100 Series causes a reserved instruction exception to be generated. This code is reserved for future extension.

Table 3-11. I64 Minor Operation Code (64-bit Only, I64-Type Instruction)

Instruction bits [10:8]							
000	001	010	011	100	101	110	111
ldsp[Note 1]	*sdsp*[Note 2]	*sdrasp*[Note 3]	*dadjsp*[Note 4]	*ldpc*[Note 5]	*daddiu5*[Note 6]	*dadiupc*[Note 7]	*dadiusp*[Note 8]

Notes 1. ldsp : ld ry, immediate
2. sdsp : sd ry, immediate
3. sdrasp : sd ra, immediate
4. dadjsp : daddiu sp, immediate
5. ldpc : ld ry, immediate
6. daddiu5 : daddiu ry, immediate
7. dadiupc : daddiu ry, pc, immediate
8. dadiusp : daddiu ry, sp, immediate

3.8 Outline of Instructions

This section describes the assembler syntax and defines each instruction. Instructions can be divided into the following four types.

- Load and Store instructions
- Computational instructions
- Jump and Branch instructions
- Special instructions

3.8.1 PC-relative instructions

PC-relative instructions is the instruction format first defined among the MIPS16 instruction set. MIPS16 supports both extension and non-extension through the Extend instruction for four PC-relative instructions.

Load Word	LW rx, offset(pc)
Load Doubleword	LD ry, offset(pc)
Add Immediate Unsigned	ADDIU rx, pc, immediate
Doubleword Add Immediate Unsigned	DADDIU ry, pc, immediate

All these instructions calculate the PC value of a PC-relative instruction or the PC value of the instruction immediately preceding as the base address. The address calculation base using various function combinations is shown next.

Table 3-12. Base PC Address Setting

Instruction	Base PC value
Non-extension PC-relative instructions not located in Jump delay slot	PC of instruction
Extension PC-relative instruction	PC of Extend instruction
Non-extension PC-relative instruction in Jump delay slot of JR or JALR	PC of JR instruction or JALR instruction
Non-extension PC-relative instruction in Jump delay slot of JAL or JALX	PC of initial halfword of JAL or JALX[Note]

Note Because the JAL and JALX instruction length is 32 bits.

The PC value used as the base for address calculation for the PC-relative instruction outlines shown in tables 3-14 and 3-15 is called base PC value. The base PC value is defined so as to be equivalent to the exception program counter (EPC) value related to the PC-relative instruction.

3.8.2 Extend instruction

The Extend instruction can extend the immediate fields of MIPS16 instructions, which have fewer immediate fields than equivalent 32-bit MIPS instructions. The Extend instruction must always precede (by one instruction) the instruction whose immediate field you want to extend. Every extended instruction consumes four bytes in program memory instead of two bytes (two bytes for Extend and two bytes for the instruction being extended), and it can cross a word boundary.

For example, the MIPS16 instruction

```
LW  ry, offset (rx)
```

contains a five-bit immediate. The immediate expands to 16 bits (000000000 || offset || 00) before execution in the pipeline. This allows 32 different offset values of 0, 4, 8, and up through 124. Once extended, this instruction can hold any of the normal 65,536 values in the range –32768 through 32767.

Shift instructions are extended to 5-bit unsigned immediate values. All other immediate instructions expand to either signed or unsigned 16-bit immediate values. The only exceptions are

```
ADDIU  ry, rx, immediate
DADDIU  ry, rx, immediate
```

which can be extended only to a 15-bit signed immediate.

There is only one restriction. Extended instructions should not be placed in jump delay slots. Otherwise, the results are unpredictable because the pipeline would attempt to execute one half the instruction.

Table 3-13 lists the MIPS16 extendable instructions, the size of their immediate, and how much each immediate can be extended when preceded with the Extend instruction.

For the instruction format of the Extend instruction, see **3.6 Instruction Format**.

Table 3-13. Extendable MIPS16 Instructions

MIPS16 Instruction	MIPS16 Immediate	Instruction Format	Extended Immediate	Instruction Format
Load Byte	5	RRI	16	EXT-RRI
Load Byte Unsigned	5	RRI	16	EXT-RRI
Load Halfword	5	RRI	16	EXT-RRI
Load Halfword Unsigned	5	RRI	16	EXT-RRI
Load Word	5 8	RRI RI	16 16	EXT-RRI EXT-RI
Load Word Unsigned	5	RRI	16	EXT-RRI
Load Doubleword	5	RRI	16	EXT-RRI
Store Byte	5	RRI	16	EXT-RRI
Store Halfword	5	RRI	16	EXT-RRI
Store Word	5 (Other) 8 (SW rx, offset(sp)) 8 (SW ra, offset(sp))	RRI RI I8	16 16 16	EXT-RRI EXT-RI EXT-I8
Store Doubleword	5 (SD ry, offset(rx)) 8 (Other)	RRI I64	16 16	EXT-RRI EXT-I64
Load Immediate	8	RI	16	EXT-RI
Add Immediate Unsigned	4 (ADDIU ry, rx, imm) 8 (ADDIU sp, imm) 8 (Other)	RRI-A I8 RI	15 16 16	EXT-RRI-A EXT-I8 EXT-RI
Doubleword Add Immediate Unsigned	4 (DADDIU ry, rx, imm) 5 (DADDIU ry, pc, imm) 8 (Other)	RRI-A RI64 I64	15 16 16	EXT-RRI-A EXT-RI64 EXT-I64
Set on Less Than Immediate	8	RI	16	EXT-RI
Set on Less Than Immediate Unsigned	8	RI	16	EXT-RI
Compare Immediate	8	RI	16	EXT-RI
Shift Left Logical	3	SHIFT	5	EXT-SHIFT
Shift Right Logical	3	SHIFT	5	EXT-SHIFT
Shift Right Arithmetic	3	SHIFT	5	EXT-SHIFT
Doubleword Shift Left Logical	3	SHIFT	6	EXT-SHIFT
Doubleword Shift Right Logical	3	RR	6	EXT- SHIFT64
Doubleword Shift Right Arithmetic	3	RR	6	EXT- SHIFT64
Branch on Equal to Zero	8	RI	16	EXT-RI
Branch on Not Equal to Zero	8	RI	16	EXT-RI
Branch on T Equal to Zero	8	I8	16	EXT-I8
Branch on T Not Equal to Zero	8	I8	16	EXT-I8
Branch Unconditional	11	I	16	EXT-I

3.8.3 Delay slots

MIPS16 instructions normally execute in one cycle. However, some instructions have special requirements that must be met to assure optimum instruction flow. The instructions include All Load, Branch, and Multiply/Divide instructions.

(1) Load delay slots

MIPS16 operates with delayed loads. This is similar to the method used by 32-bit length instruction sets. If another instruction references the load destination register before the load operation is completed, one cycle occurs automatically. To assure the best performance, the compiler should always schedule load delay slots as early as possible.

(2) Branch delay slots not supported

Unlike for 32-bit length instructions, there are no branch delay slots for branch instructions in MIPS16. If a branch is taken, the instruction that immediately follows the branch (instruction corresponding to 32-bit length instruction's delay slot) is cancelled. There are no restrictions on the instruction that follows a branch instruction, and such instruction is executed only when a branch is not taken. Branches, jumps, and extended instructions are permitted in the instruction slot after a branch.

(3) Jump delay slots

With MIPS16, there is a delay of one cycle after each jump instruction. The processor executes any instruction in the jump delay slot before it executes the jump target instruction. Two restrictions apply to any instruction placed in the jump delay slot:

1. Do not specify a branch or jump in the delay slot.
2. Do not specify an extended instruction (32 bits) in the delay slot. Doing so will make the results unpredictable.

(4) Multiply and divide scheduling

Multiply and divide latency depends on the hardware implementation. If an MFLO or MFHI instruction references the Multiply or Divide result registers before the result is ready, the pipeline stalls until the operation is complete and the result is available. However, to assure the best performance, the compiler should always schedule Multiply and Divide instructions as early as possible.

MIPS16 requires that all MFHI and MFLO instructions be followed by two instructions that do not write to the HI or LO registers. Otherwise, the data read by MFLO or MFHI will be undefined. The Extend instruction is counted singly as one instruction.

3.8.4 Instruction details

(1) Load and store instructions

Load and Store instructions move data between memory and the general-purpose registers. The only addressing mode that is supported is the mode for adding immediate offset to the base register.

Table 3-14. Load and Store Instructions (1/3)

Instruction	Format and Description
Load Byte	LB ry, offset (rx) The 5-bit immediate is zero extended and then added to the contents of general-purpose register rx to form the virtual address. The bytes of the memory location specified by the address are sign extended and loaded into general-purpose register ry.
Load Byte Unsigned	LBU ry, offset (rx) The 5-bit immediate is zero extended and then added to the contents of general-purpose register rx to form the virtual address. The bytes of the memory location specified by the address are zero extended and loaded into general-purpose register ry
Load Halfword	LH ry, offset (rx) The 5-bit immediate is shifted left one bit, zero extended, and then added to the contents of general-purpose register rx to form the virtual address. The halfword of the memory location specified by the address is sign extended and loaded to general-purpose register ry. If the least significant bit of the address is not 0, an address error exception is generated.
Load Halfword Unsigned	LHU ry, offset (rx) The 5-bit immediate is shifted left one bit, zero extended, and then added to the contents of general-purpose register rx to form the virtual address. The halfword of the memory location specified by the address is zero extended and loaded to general-purpose register ry. If the least significant bit of the address is not 0, an address error exception is generated.
Load Word	LW ry, offset (rx) The 5-bit immediate is shifted left two bits, zero extended, and then added to the contents of general-purpose register rx to form the virtual address. The word of the memory location specified by the address is loaded to general-purpose register ry. In the 64-bit mode, it is further sign extended to 64 bits. If either of the lower two bits is not 0, an address error exception is generated.
	LW rx, offset (pc) The two lower bits of the BasePC value associated with the instruction are cleared to form the masked BasePC value. The 8-bit immediate is shifted left two bits, zero extended, and then added to the masked BasePC to form the virtual address. The contents of the word at the memory location specified by the address are loaded to general-purpose register rx. In the 64-bit mode, it is further sign extended to 64 bits.
	LW rx, offset (sp). The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of general-purpose register sp to form the virtual address. The contents of the word at the memory location specified by the address are loaded to general-purpose register rx. In the 64-bit mode, it is further sign extended to 64 bits. If either of the two lower bits of the address is 0, an address error exception is generated.

Table 3-14. Load and Store Instructions (2/3)

Instruction	Format and Description
Load Word Unsigned	LWU ry, offset (rx) The 5-bit immediate is shifted left two bits, zero extended to 64 bits, and then added to the contents of general-purpose register rx to form the virtual address. The word of the memory location specified by the address is zero extended and loaded to general-purpose register ry. If either of the two lower bits of the address is not 0, an address error exception is generated. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Load Doubleword	LD ry, offset (rx) The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general-purpose register rx to form the virtual address. The 64-bit doubleword of the memory location specified by the address is loaded to general-purpose register ry. If any of the lower three bits of the address is not 0, an address error exception is generated. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	LD ry, offset (pc) The lower three bits of the base PC value related to the instruction are cleared to form the masked BasePC value. The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the masked BasePC to form the virtual address. The 64-bit doubleword at the memory location specified by the address is loaded to general-purpose register ry. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	LD ry, offset (sp) The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and added to the contents of general-purpose register sp to form the virtual address. The 64-bit doubleword at the memory location specified by the address is loaded to general-purpose register ry. If any of the three lower bits of the address is not 0, an address error exception is generated. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.

Table 3-14. Load and Store Instructions (3/3)

Instruction	Format and Description
Store Byte	SB ry, offset (rx) The 5-bit immediate is zero extended and then added to the contents of general-purpose register rx to form the virtual address. The least significant byte of general-purpose register ry is stored to the memory location specified by the address.
Store Halfword	SH ry, offset (rx) The 5-bit immediate is shifted left one bit, zero extended, and then added to the contents of general-purpose register rx to form the virtual address. The lower halfword of general-purpose register ry is stored to the memory location specified by the address. If the least significant bit of the address is not 0, an address error exception is generated.
Store Word	SW ry, offset (rx) The 5-bit immediate is shifted left two bits, zero extended, and then added to the contents of general-purpose register rx to form a virtual address. The contents of general-purpose register ry are stored to the memory location specified by the address. If either of the two lower bits of the address is not 0, an address error exception is generated.
	SW rx, offset (sp) The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of general-purpose register sp to form the virtual address. The contents of general-purpose register rx are stored to the memory location specified by the address. If either of the two lower bits of the address is not 0, and address error exception is generated.
	SW ra, offset (sp) The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of general-purpose register sp to form the virtual address. The contents of general-purpose register ra are stored to the memory location specified by the address. If either of the two lower bits of the address is not 0, an address error exception is generated.
Store Doubleword	SD ry, offset (rx) The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general-purpose register rx to form the virtual address. The 64 bits of general-purpose register ry are stored to the memory location specified by the address. If any of the lower three bits of the address is not 0, an address error exception is generated. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	SD ry, offset (sp) The 5-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general-purpose register sp to form the virtual address. The 64 bits of general-purpose register ry are stored to the memory location specified by the address. If any of the lower three bits of the address is not 0, an address error exception is generated. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	SD ra, offset (sp). The 8-bit immediate is shifted left three bits, zero extended to 64 bits, and then added to the contents of general-purpose register sp to form the virtual address. The 64 bits of general-purpose register ra are stored to the memory location specified by the memory. If any of the three lower bits of the address is not 0, an address error exception is generated. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.

(2) Computational instructions

Computational instructions perform arithmetic, logical, and shift operations on values in registers. There are four categories of Computational instructions: ALU Immediate, Two/Three-Operand Register-Type, Shift, and Multiply/Divide.

Table 3-15. ALU Immediate Instructions (1/2)

Instruction	Format and Description
Load Immediate	LI rx, immediate The 8-bit immediate is zero extended and loaded to general-purpose register rx.
Add Immediate Unsigned	ADDIU ry, rx, immediate The 4-bit immediate is sign extended and then added to the contents of general-purpose register rx to form a 32-bit result. The result is placed into general-purpose register ry. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value.
	ADDIU rx, immediate The 8-bit immediate is sign extended and then added to the contents of general-purpose register rx to form a 32-bit result. The result is placed into general-purpose register rx. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value.
	ADDIU sp, immediate The 8-bit immediate is shifted left three bits, sign extended, and then added to the contents of general-purpose register sp to form a 32-bit result. The result is placed into general-purpose register sp. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value.
	ADDIU rx, pc, immediate The two lower bits of the BasePC value associated with the instruction are cleared to form the masked BasePC value. The 8-bit immediate is shifted left two bits, zero extended, and then added to the masked BasePC value to form the virtual address. This address is placed into general-purpose register rx. No integer overflow exception occurs under any circumstances.
	ADDIU rx, sp, immediate The 8-bit immediate is shifted left two bits, zero extended, and then added to the contents of register sp to form a 32-bit result. The result is placed into general-purpose register rx. No integer overflow exception occurs under any circumstance. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value.

Table 3-15. ALU Immediate Instructions (2/2)

Instruction	Format and Description
Doubleword Add Immediate Unsigned	DADDIU ry, rx, immediate The 4-bit immediate is sign extended to 64 bits, and then added to the contents of register rx to form a 64-bit result. The result is placed into general-purpose register ry. No integer overflow exception occurs under any circumstances. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	DADDIU ry, immediate The 5-bit immediate is sign extended to 64 bits, and then added to the contents of register ry to form a 64-bit result. The result is placed into general-purpose register ry. No integer overflow exception occurs under any circumstances. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	DADDIU sp, immediate The 8-bit immediate is shifted left three bits, sign extended to 64 bits, and then added to the contents of register sp to form a 64-bit result. The result is placed into general-purpose register sp. No integer overflow exception occurs under any circumstances. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	DADDIU ry, pc, immediate The two lower bits of the BasePC value associated with the instruction are cleared to form the masked BasePC value. The 5-bit immediate is shifted left two bits, zero extended, and added to the masked BasePC value to form the virtual address. This address is placed into general-purpose register ry. No integer overflow exception occurs under any circumstances. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
	DADDIU ry, sp, immediate The 5-bit immediate is shifted left two bits, zero extended to 64 bits, and then added to the contents of register sp to form a 64-bit result. This result is placed into register ry. No integer overflow exception occurs under any circumstances. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Set on Less Than Immediate	SLTI rx, immediate The 8-bit immediate is zero extended and subtracted from the contents of general-purpose register rx. Considering both quantities as signed integers, if rx is less than the zero-extended immediate, the result is set to 1; otherwise, the result is set to 0. The result is placed into register T ($24).
Set on Less Than Immediate Unsigned	SLTIU rx, immediate The 8-bit immediate is zero extended and subtracted from the contents of general-purpose register rx. Considering both quantities as signed integers, if rx is less than the zero-extended immediate, the result is set to 1; otherwise, the result is set to 0. The result is placed into register T ($24).
Compare Immediate	CMPI rx, immediate The 8-bit immediate is zero extended and exclusive ORed in 1-bit units with the contents of general-purpose register rx. The result is placed into register T ($24).

Table 3-16. Two-/Three-Operand Register Type (1/2)

Instruction	Format and Description
Add Unsigned	ADDU rz, rx, ry The contents of general-purpose registers rx and ry are added together to form a 32-bit result. The result is placed into general-purpose register rz. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value.
Subtract Unsigned	SUBU rz, rx, ry The contents of general-purpose register ry are subtracted from the contents of general-purpose register rx. The 32-bit result is placed into general-purpose register rz. No integer overflow exception occurs under any circumstances. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value.
Doubleword Add Unsigned	DADDU rz, rx, ry The contents of general-purpose register ry are added to the contents of general-purpose register rx. The 64-bit result is placed into register rz. No integer overflow exception occurs under any circumstances. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Subtract Unsigned	DSUBU rz, rx, ry The contents of general-purpose register ry are subtracted from the contents of general-purpose register rx. The 64-bit result is placed into general-purpose register rz. No integer overflow exception occurs under any circumstances. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Set on Less Than	SLT rx, ry The contents of general-purpose register ry are subtracted from the contents of general-purpose register rx. Considering both quantities as signed integers, if the contents of rx are less than the contents of ry, the result is set to 1; otherwise, the result is set to 0. The result is placed into register T ($24). No integer overflow exception occurs. The comparison is valid even if the subtraction overflows.
Set on Less Than Unsigned	SLTU rx, ry The contents of general-purpose register ry are subtracted from the contents of general-purpose register rx. Considering both quantities as unsigned integers, if the contents of rx are less than the contents of ry, the result is set to 1; otherwise, the result it set to 0. The result is place in register T ($24). No integer overflow exception occurs. The comparison is valid even if the subtraction overflows.

Table 3-16. Two-/Three-Operand Register Type (2/2)

Instruction	Format and Description
Compare	CMP rx, ry The contents of general-purpose register ry are Exclusive-ORed with the contents of general-purpose register rx. The result is placed into register T ($24).
Negate	NEG rx, ry The contents of general-purpose register ry are subtracted from zero to form a 32-bit result. The result is placed in general-purpose register rx.
AND	AND rx, ry The contents of general-purpose register ry are logical ANDed with the contents of general-purpose register rx in 1-bit units. The result is placed in general-purpose register rx.
OR	OR rx, ry The contents of general-purpose register ry are logical ORed with the contents of general-purpose register ry. The result is placed in general-purpose register rx.
Exclusive OR	XOR rx, ry The contents of general-purpose register ry are Exclusive-ORed with the contents of general-purpose register rx in 1-bit units. The result is placed in general-purpose register rx.
NOT	NOT rx, ry The contents of general-purpose register ry are inverted in 1-bit units and placed in general-purpose register rx.
Move	MOVE ry, r32 The contents of general-purpose register r32 are moved to general-purpose register ry. R32 can specify any one of the 32 general-purpose registers.
	MOVE r32, rz The contents of general-purpose register rz are moved to general-purpose register r32. r32 can specify any one of the 32 general-purpose registers

Table 3-17. Shift Instructions (1/2)

Instruction	Format and Description
Shift Left Logical	SLL rx, ry, immediate The 32-bit contents of general-purpose register ry are shifted left and zeros are inserted into the emptied low-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The result is placed in general-purpose register rx. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result.
Shift Right Logical	SLR rx, ry, immediate The 32-bit contents of general-purpose register ry are shifted right, and zeros are inserted into the emptied high-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The result is placed in general-purpose register rx. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result.
Shift Right Arithmetic	SRA rx, ry, immediate The 32-bit contents of general-purpose register ry are shifted right and the emptied high-order bits are sign extended. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result.
Shift Left Logical Variable	SLLV ry, rx The 32-bit contents of general-purpose register ry are shifted left, and zeros are inserted into the emptied low-order bits. The five low-order bits of general-purpose register rx specify the shift count. The result is placed in general-purpose register ry. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result.
Shift Right Logical Variable	SRLV ry, rx The 32-bit contents of general-purpose register ry are shifted right, and the emptied high-order bits are sign extended. The five lower-order bits of general-purpose register rx specify the shift count. The register is placed in general-purpose register ry. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result.
Shift Right Arithmetic Variable	SRAV ry, rx The 32-bit contents of general-purpose register ry are shifted right, and the emptied high-order bits are sign extended. The five low-order bits of general-purpose register rx specify the shift count. The result is placed in general-purpose register ry. In the 64-bit mode, the value that is formed by sign-extending shifted 32-bit value is stored as the result.

Table 3-17. Shift Instructions (2/2)

Instruction	Format and Description
Doubleword Shift Left Logical	DSLL rx, ry, immediate The 64-bit doubleword contents of general-purpose register ry are shifted left, and zeros are inserted into the emptied low-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. The 64-bit result is placed in general-purpose register rx. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Shift Right Logical	DSRL ry, immediate The 64-bit doubleword contents of general-purpose register ry are shifted right, and zeros are inserted into the emptied high-order bits. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Shift Right Arithmetic	DSRA ry, immediate The 64-bit doubleword contents of general-purpose register ry are shifted right, and the emptied high-order bits are sign extended. The 3-bit immediate specifies the shift count. A shift count of 0 is interpreted as a shift count of 8. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Shift Left Logical Variable	DSLLV ry, rx The 64-bit doubleword contents of general-purpose register ry are shifted left, and zeros are inserted into the emptied low-order bits. The six low-order bits of general-purpose register rx specify the shift count. The result is placed in general-purpose register ry. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Shift Right Logical Variable	DSRLV ry, rx The 64-bit doubleword contents of general-purpose register ry are shifted right, and zeros are inserted into the emptied high-order bits. The six low-order bits of general-purpose register rx specify the shift count. The result is placed in general-purpose register ry. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Shift Right Arithmetic Variable	DSRAV ry, rx The 64-bit doubleword contents of general-purpose register ry are shifted right, and the emptied high-order bits are sign extended. The six low-order bits of general-purpose register rx specify the shift count. The result is placed in general-purpose register ry. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.

Table 3-18. Multiply/Divide Instructions (1/2)

Instruction	Format and Description
Multiply	MULT rx, ry The contents of general-purpose registers rx and ry are multiplied, treating both operands as 32-bit two's complement values. No integer overflow exception occurs. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. The low-order 32-bit word of the result are placed in special register LO, and the high-order 32-bit word is placed in special register HI. In the 64-bit mode, each result is sign extended and then stored. If either of the two immediately preceding instructions is MFHI or MFLO, their transfer instruction execution result becomes undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions, and the MULT instruction.
Multiply Unsigned	MULTU rx, ry The contents of general-purpose registers rx and ry are multiplied, treating both operands as 32-bit unsigned values. No integer overflow exception occurs. In the 64-bit mode, the operand must be a 64-bit value formed by sign-extending a 32-bit value. The low-order 32-bit word of the result is placed in special register LO, and the high-order 32-bit word is placed in special register HI. In the 64-bit mode, each result is sign extended and stored. If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the MULTU instruction.
Divide	DIV rx, ry The contents of general-purpose register rx are divided by the contents of general-purpose register ry, treating both operands as 32-bit two's complement values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 32-bit quotient is placed in special register LO, and the 32-bit remainder is placed in special register HI. In the 64-bit mode, the result is sign extended. Normally, this instruction is executed after instructions checking for division by zero and overflow. If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DIV instruction.
Divide Unsigned	DIVU rx, ry The contents of general-purpose register rx are divided by the contents of general-purpose register ry, treating both operands as unsigned values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 32-bit quotient is placed in special register LO, and the 32-bit remainder is placed in special register HI. In the 64-bit mode, the result is sign extended. Normally, this instruction is executed after instructions checking for division by zero. If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DIVU instruction.
Move from HI	MFHI rx The contents of special register HI are loaded into general-purpose register rx. To ensure correct operation when an interrupt occurs, do not use an instruction that changes the HI register (MULT, MULTU, DIV, DIVU, DMULT, DMULTU, DDIV, DDIVU) for the two instructions after the MFHI instruction.

Table 3-18. Multiply/Divide Instructions (2/2)

Instruction	Format and Description
Move from LO	MFLO rx The contents of special register LO are loaded into general-purpose register rx. To ensure correct operation when an interrupt occurs, do not use an instruction that changes the HI register (MULT, MULTU, DIV, DIVU, DMULT, DMULTU, DDIV, DDIVU) for the two instructions after the MFLO instruction.
Doubleword Multiply	DMULT rx, ry The 64-bit contents of general-purpose register rx and ry are multiplied, treating both operands as two's complement values. No integer overflow exception occurs. The low-order 64 bits of the result are placed in special register LO, and the high-order 64 bits are placed in special register HI. If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DMULT instruction. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Multiply Unsigned	DMULTU rx, ry The 64-bit contents of general-purpose registers rx and ry are multiplied, treating both operands as unsigned values. No integer overflow exception occurs. The low-order 64 bits of the result are placed in special register LO, and the high-order 64 bits of the result are placed in special register HI. If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DMULTU instruction. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword divide	DDIV rx, ry The 64-bit contents of general-purpose registers rx are divided by the contents of general-purpose register ry, treating both operands as two's complement values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 64-bit quotient is placed in special register LO, and the 64-bit remainder is placed in special register HI. Normally, this instruction is executed after instructions checking for division by zero and overflow. If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DDIV instruction. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.
Doubleword Divide Unsigned	DDIVU rx, ry The 64-bit contents of general-purpose register rx are divided by the contents of general-purpose register ry, treating both operands as unsigned values. No integer overflow exception occurs. The result when the divisor is 0 is undefined. The 64-bit quotient is placed in special register LO, and the 64-bit remainder is placed in special register HI. Normally, this instruction is executed after an instruction checking for division by zero. If either of the two immediately preceding instructions is MFHI or MFLO, the result of execution of these transfer instructions is undefined. To obtain the correct result, insert two or more other instructions between the MFHI, MFLO instructions and the DDIVU instruction. This operation is defined in the 64-bit mode and the 32-bit kernel mode. When this instruction is executed in the 32-bit user/supervisor mode, a reserved instruction exception is generated.

(3) Jump and branch instructions

Jump and Branch instructions change the control flow of a program.

All Jump instructions occur with a one-instruction delay. That is, the instruction immediately following the jump is always executed.

Branch instructions do not have a delay slot. If a branch is taken, the instruction immediately following the branch is never executed. If the branch is not taken, the instruction immediately following the branch is always executed.

Table 3-19 shows the MIPS16 Jump and Branch instructions.

Table 3-19. Jump and Branch Instructions (1/2)

Instruction	Format and Description
Jump and Link	JAL target The 26-bit target address is shifted left two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction immediately following the delay slot is placed in register ra. The ISA Mode bit is left unchanged. The value stored in ra bit 0 will reflect the current ISA Mode bit.
Jump and Link Exchange	JALX target The 26-bit target address is shifted left two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction immediately following the delay slot is placed in register ra. The ISA Mode bit is inverted with a delay of one instruction. The value stored in ra bit 0 will reflect the ISA Mode bit before execution of the Jump execution.
Jump Register	JR rx The program unconditionally jumps to the address specified in general-purpose register rx, with a delay of one instruction. The instruction sets the ISA Mode bit to the value in rx bit 0. If the Jump target address is in the MIPS16 instruction length mode, no address exception occurs when bit 0 of the source register is 1 because bit 0 of the target address is 0 so that the instruction is located at the halfword boundary. If the 32-bit length instruction mode is changed, an address exception occurs when the jump target address is fetched if the two low-order bits of the target address are not 0.
	JR ra The program unconditionally jumps to the address specified in register ra, with a delay of one instruction. The instruction sets the ISA Mode bit to the value in ra bit 0. If the Jump target address is in the MIPS16 instruction length mode, no address exception occurs when bit 0 of the source register is 1 because bit 0 of the target address is 0 so that the instruction is located at the halfword boundary. If the 32-bit length instruction mode is changed, an address exception occurs when the jump target address is fetched if the two low-order bits of the target address are not 0.
Jump and Link Register	JALR ra, rx The program unconditionally jumps to the address contained in register rx, with a delay of one instruction. This instruction sets the ISA Mode bit to the value in rx bit 0. The address of the instruction immediately following the delay slot is placed in register ra. The value stored in ra bit 0 will reflect the ISA mode bit before the jump execution is executed. If the Jump target address is in the MIPS16 instruction length mode, no address exception occurs when bit 0 of the source register is 1 because bit 0 of the target address is 0 so that the instruction is located at the halfword boundary. If the 32-bit length instruction mode is changed, an address exception occurs when the jump target address is fetched if the two low-order bits of the target address are not 0.

Table 3-19. Jump and Branch Instructions (2/2)

Instruction	Format and Description
Branch on Equal to Zero	BEQZ rx, immediate The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of general-purpose register rx are equal to zero, the program branches to the target address. No delay slot is generated.
Branch on Not Equal to Zero	BNEZ rx, immediate The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of general-purpose register rx are not equal to zero, the program branches to the target address. No delay slot is generated.
Branch on T Equal to Zero	BTEQZ immediate The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of special register T ($24) are not equal to zero, the program branches to the target address. No delay slot is generated.
Branch on T Not Equal to Zero	BTNEZ immediate The 8-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. If the contents of special register T ($24) are not equal to zero, the program branches to the target address. No delay slot is generated.
Branch Unconditional	B immediate The 11-bit immediate is shifted left one bit, sign extended, and then added to the address of the instruction after the branch to form the target address. The program branches to the target address unconditionally.

(4) Special instructions

Special instructions unconditionally perform branching to general exception vectors. Special instructions are of the R type. Table 3-20 shows three special instructions.

Table 3-20. Special Instructions

Instruction	Format and Description
Breakpoint	BREAK immediate A breakpoint trap occurs, immediately and unconditionally transferring control to the exception handler. By using a 6-bit code area, parameters can be sent to the exception handler. If the exception handler uses this parameter, the contents of memory including instructions must be loaded as data.
Extend	EXTEND immediate The 11-bit immediate is combined with the immediate in the next instruction to form a larger immediate equivalent to 32-bit MIPS. The Extend instruction must always precede (by one instruction) the instruction whose immediate field you want to extend. Every extended instruction consumes four bytes in program memory instead of two bytes (two bytes for Extend and two bytes for the instruction being extended), and it can cross a word boundary. (For details, see **3.8.2 Extend instruction.**)
System Call	SYSCALL A system call trap occurs, immediately and unconditionally transferring control to the exception handler.

CHAPTER 4 PIPELINE

This chapter describes the basic operation of the VR4100 Series processor pipeline, which includes descriptions of the delay slots (instructions that follow a branch or load instruction in the pipeline), and interrupts to the pipeline flow caused by interlocks and exceptions.

4.1 Pipeline Stages

In the VR Series, an instruction execution system called a pipeline is adopted. In the pipeline, instruction execution processing is delimited into several stages. Instruction execution is complete when each stage is passed. When processing of one instruction in one stage of the pipeline is complete, the next instruction enters that stage. When the pipeline is full, it means that instructions equaling the number of pipeline stages are being executed simultaneously.

The pipeline clock is called the PClock. Each cycle of the PClock is called a PCycle. Instructions are read in synchronization with the PClock. Each stage of the pipeline is executed in one PCycle. Therefore, executing an instruction requires as many PCycles as the number of pipeline stages. When the required data has not been cached and must instead be fetched from the main memory, the execution requires more cycles than the number of pipeline stages.

4.1.1 VR4121, VR4122, VR4181A

The pipeline of the VR4121, VR4122, or VR4181A has five stages in the MIPS III (32-bit length) instruction mode, or six stages in the MIPS16 (16-bit length) instruction mode.

The name and meanings of each stage are as follows.

- IF - Instruction cache fetch
- IT - Instruction translation (in MIPS16 instruction mode only)
- RF - Register fetch
- EX - Execution
- DC - Data cache fetch
- WB - Writeback

Figure 4-1. Pipeline Stages (VR4121, VR4122, VR4181A)

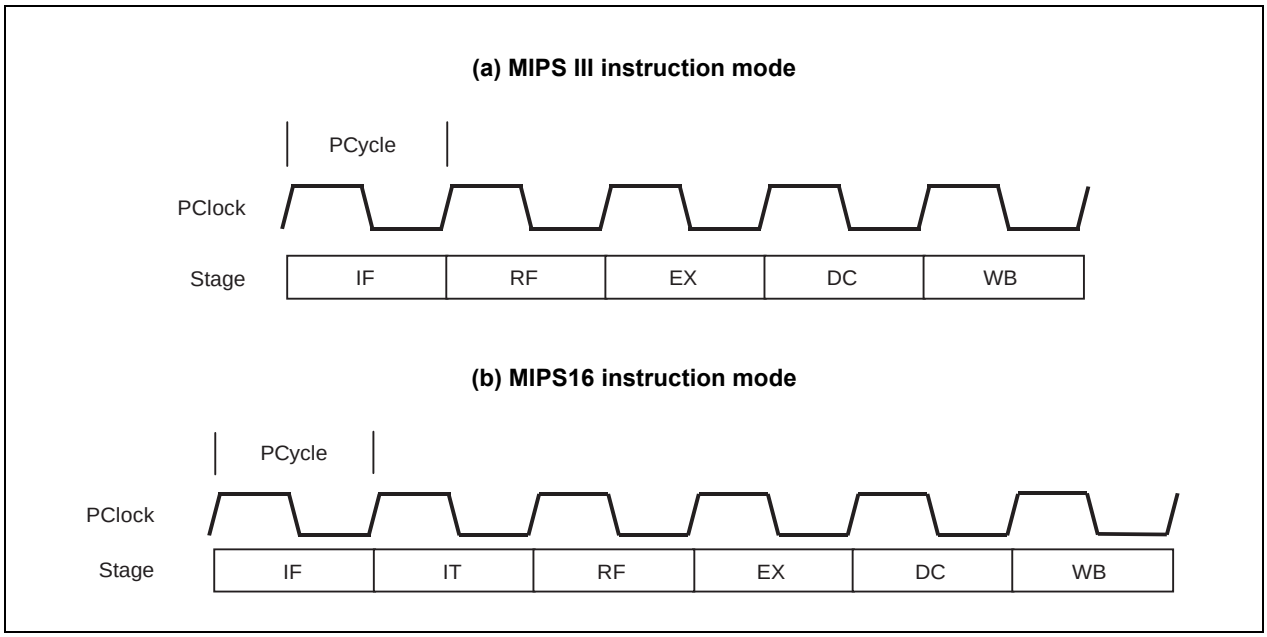
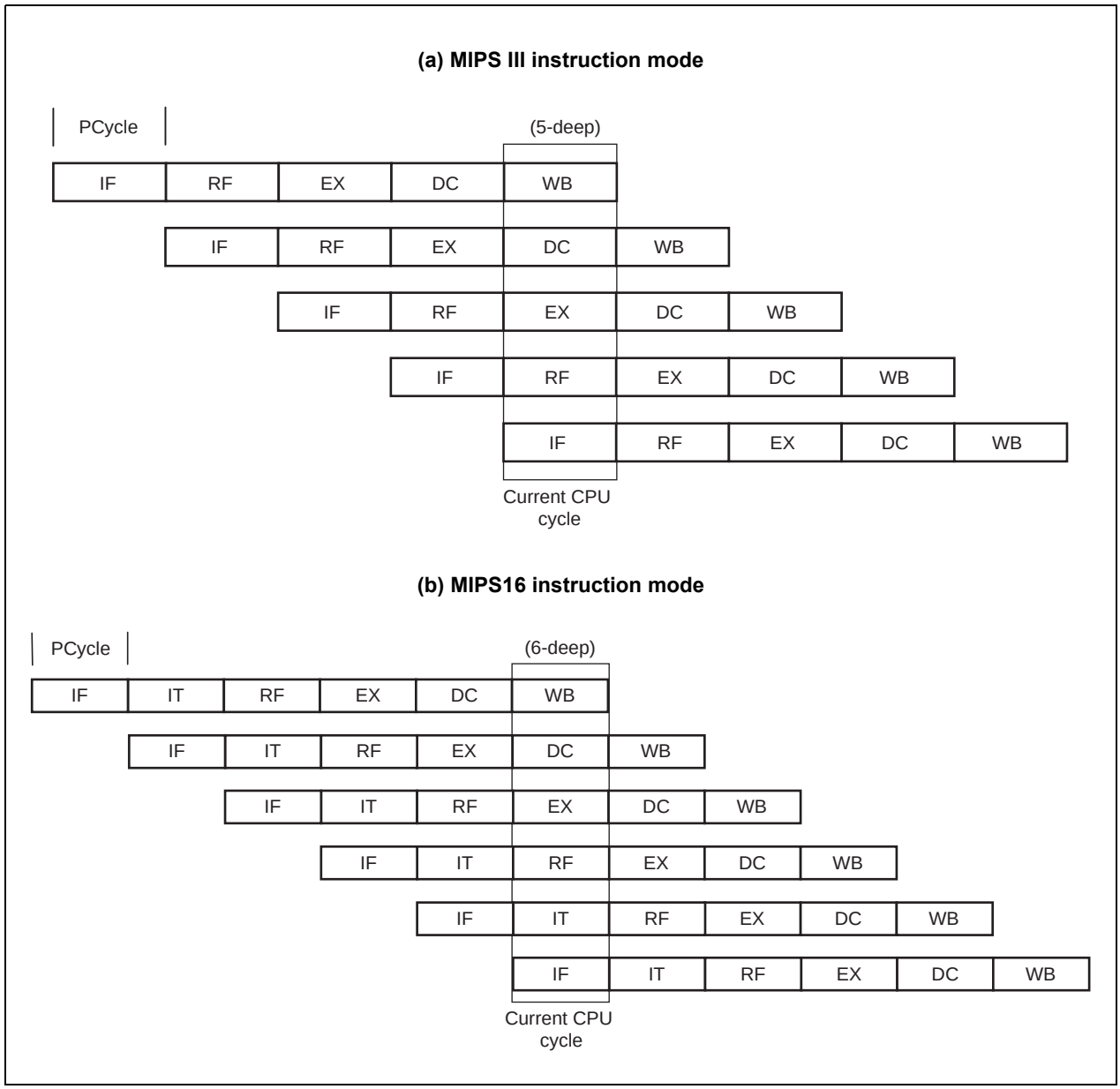


Figure 4-2 shows instruction execution in the pipeline. In this figure, a row indicates the execution process of each instruction, and a column indicates the processes executed simultaneously.

Figure 4-2. Instruction Execution in the Pipeline (VR4121, VR4122, VR4181A)



4.1.2 VR4131

The pipeline of the VR4131 employs the 2-way superscalar mechanism that can execute two instructions each in the same stage. Each pipeline has six stages in the MIPS III (32-bit length) instruction mode, or seven stages in the MIPS16 (16-bit length) instruction mode.

The name and meanings of each stage are as follows.

- IF - Instruction cache fetch
- IT - Instruction translation (in MIPS16 instruction mode only)
- RF - Register fetch
- EX - Execution
- DC1 - Data cache fetch
- DC2 - Data read
- WB - Writeback

Figure 4-3. Pipeline Stages (VR4131)

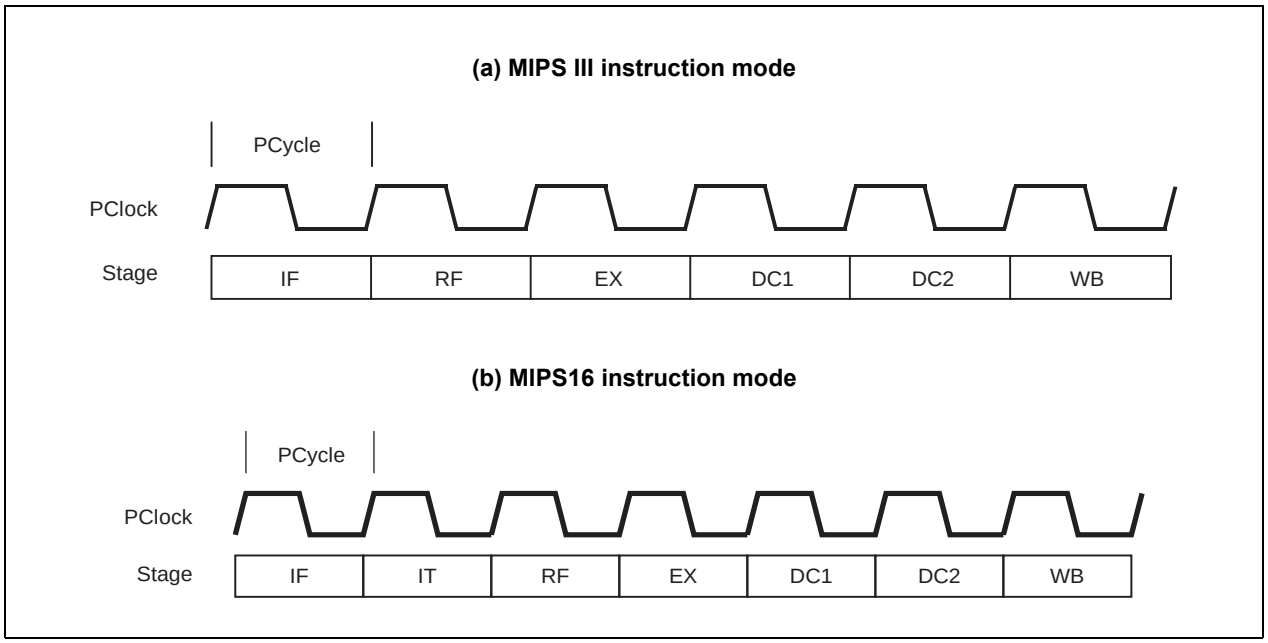
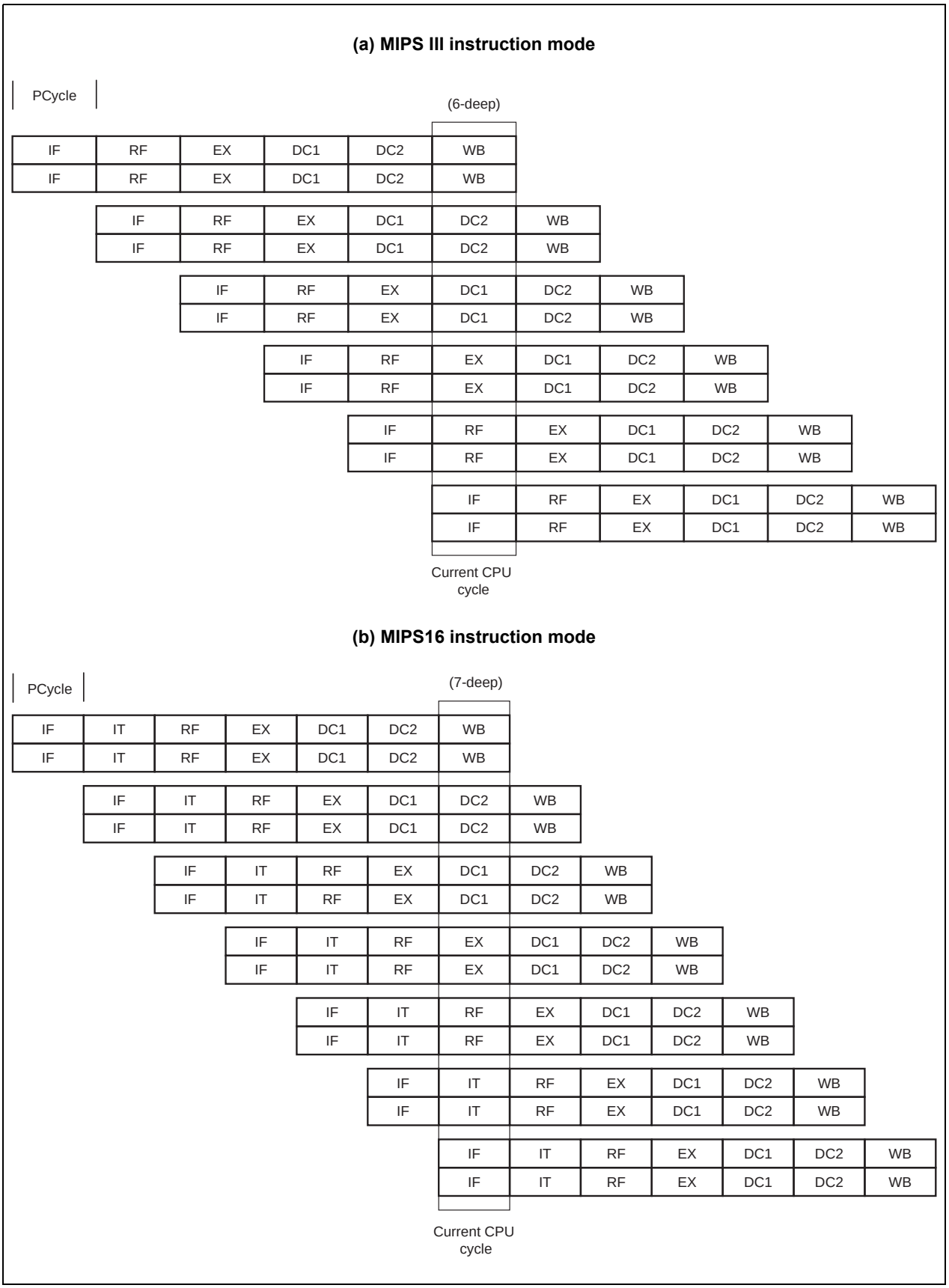


Figure 4-4 shows instruction execution in the pipeline. In this figure, a row indicates the execution process of each instruction, and a column indicates the processes executed simultaneously.

Figure 4-4. Instruction Execution in the Pipeline (VR4131)



4.1.3 VR4181

The pipeline of the VR4181 has five stages regardless the instruction set modes. Each stage has two phases: Φ1 and Φ2.

The name and meanings of each stage are as follows.

- IF - Instruction cache fetch
- RF - Register fetch
- EX - Execution
- DC - Data cache fetch
- WB - Write back

Figure 4-5. Pipeline Stages (VR4181)

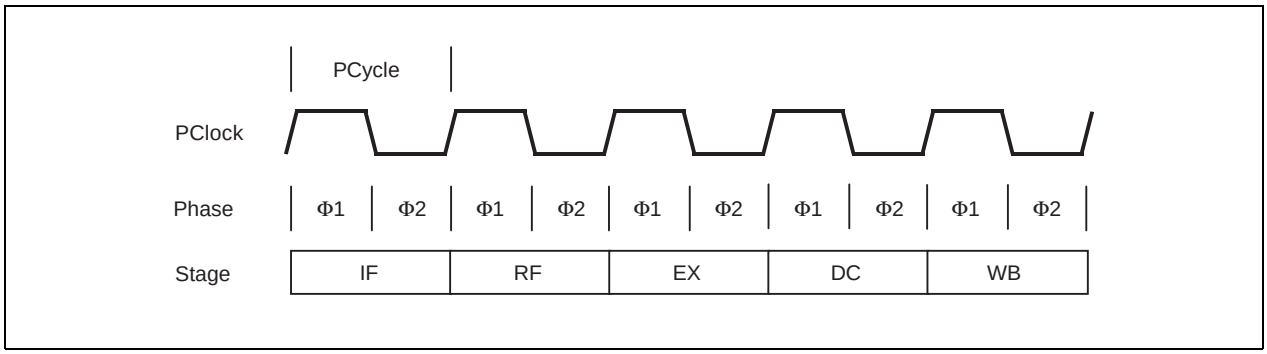
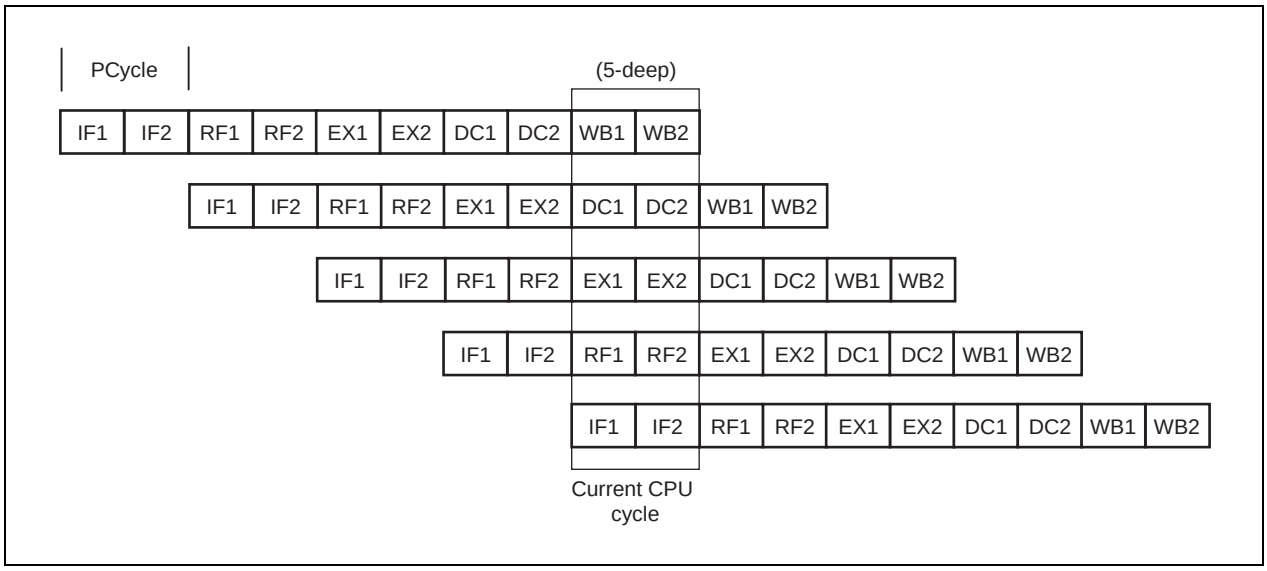


Figure 4-6 shows instruction execution in the pipeline. In this figure, a row indicates the execution process of each instruction, and a column indicates the processes executed simultaneously.

Figure 4-6. Instruction Execution in the Pipeline (VR4181)



4.2 Branch Delay

During a VR4100 Series' pipeline operation, a branch delay occurs when:

- Target address is calculated by a Jump instruction
- Branch condition of branch instruction is met and then logical operation starts for branch-destination comparison

The instruction location immediately following a Jump/Branch instruction is referred to as the branch delay slot.

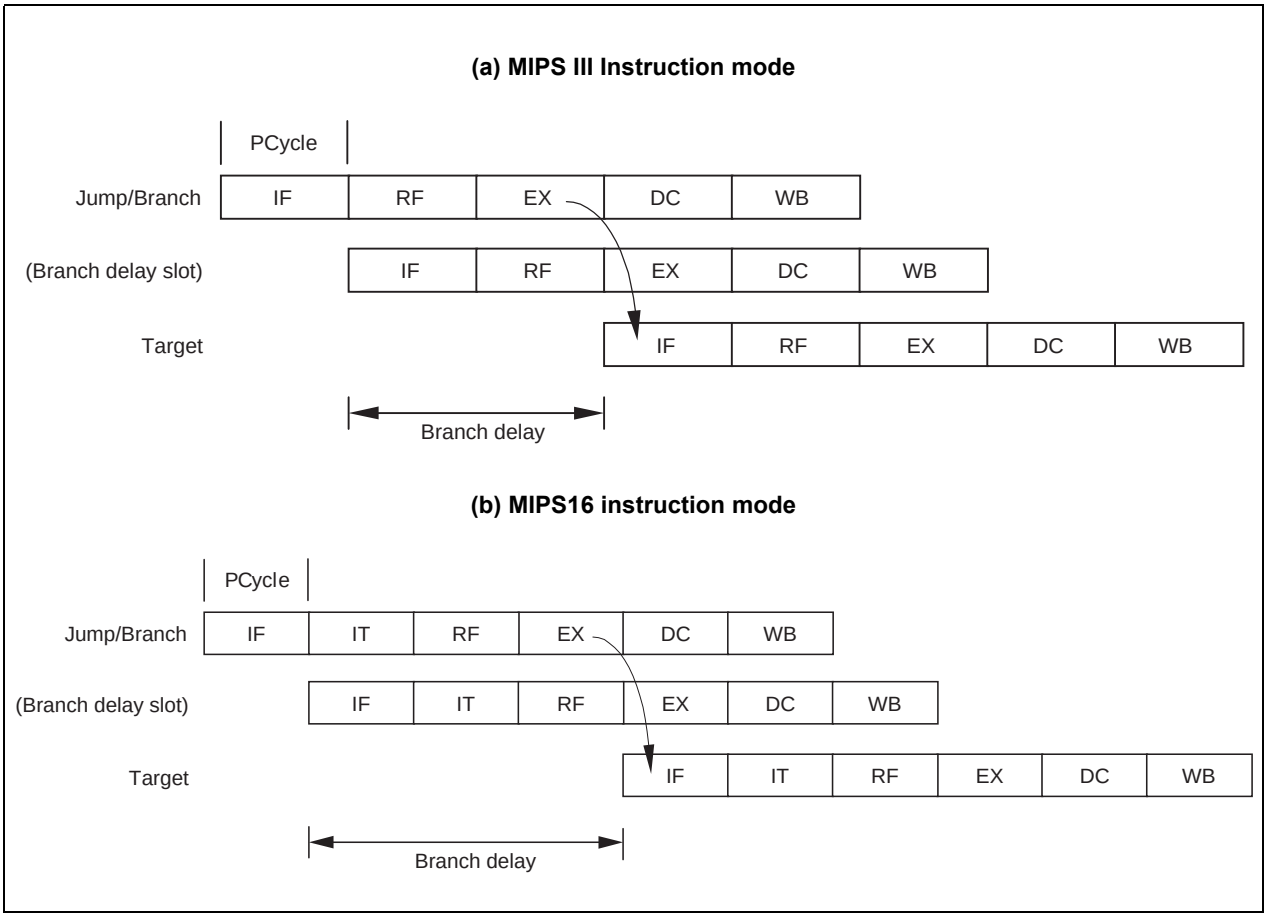
4.2.1 VR4121, VR4122, VR4181A

The instruction address generated at the EX stage in the Jump/Branch instruction is available in the IF stage two instructions later.

In the VR4121, VR4122, and VR4181A, two cycles of branch delay occurs during MIPS III (32-bit length) instruction mode, or three cycles during MIPS16 (16-bit length) instruction mode, when a branch condition is met. An instruction in the branch delay slot is executed during MIPS III instruction mode (except for Branch Likely instructions), though it is discarded during MIPS16 instruction mode.

Figure 4-7 illustrates the branch delay and the location of the branch delay slot.

Figure 4-7. Branch Delay (VR4121, VR4122, VR4181A)



4.2.2 VR4131

The instruction address prefetched at the RF stage in the Jump/Branch instruction is available in the IF stage two instructions later.

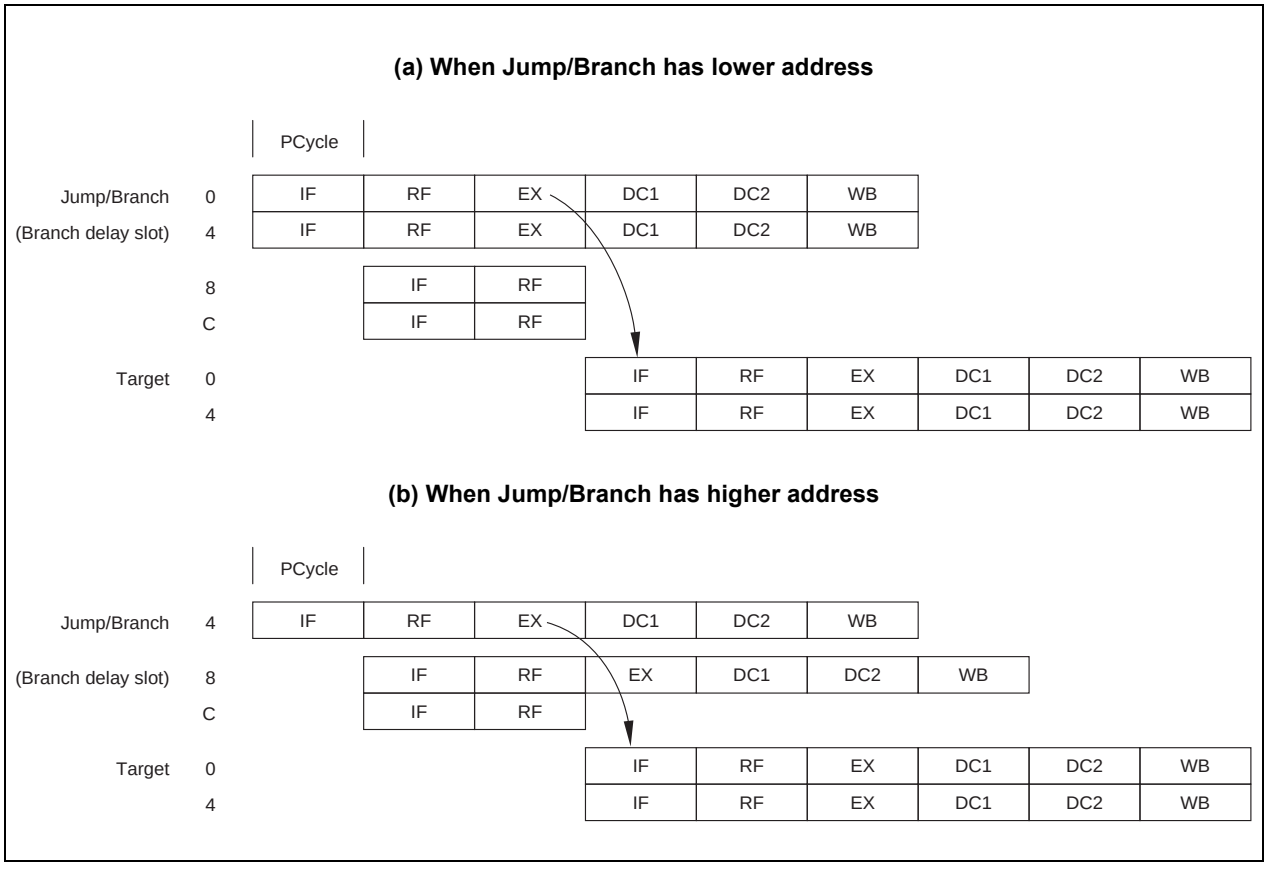
Since the VR4131 employs the 2-way superscalar mechanism, the manipulation of succeeding instructions differs depending that the address of a Jump/Branch instruction is higher or not than that of the instruction in the other way when it is fetched.

(1) MIPS III instruction mode

In the VR4131, two cycles of branch delay occurs when a branch condition is met. An instruction in the branch delay slot is executed (except for Branch Likely instructions).

Figure 4-8 illustrates the branch delay and the location of the branch delay slot.

Figure 4-8. Branch Delay (VR4131, MIPS III Instruction Mode)

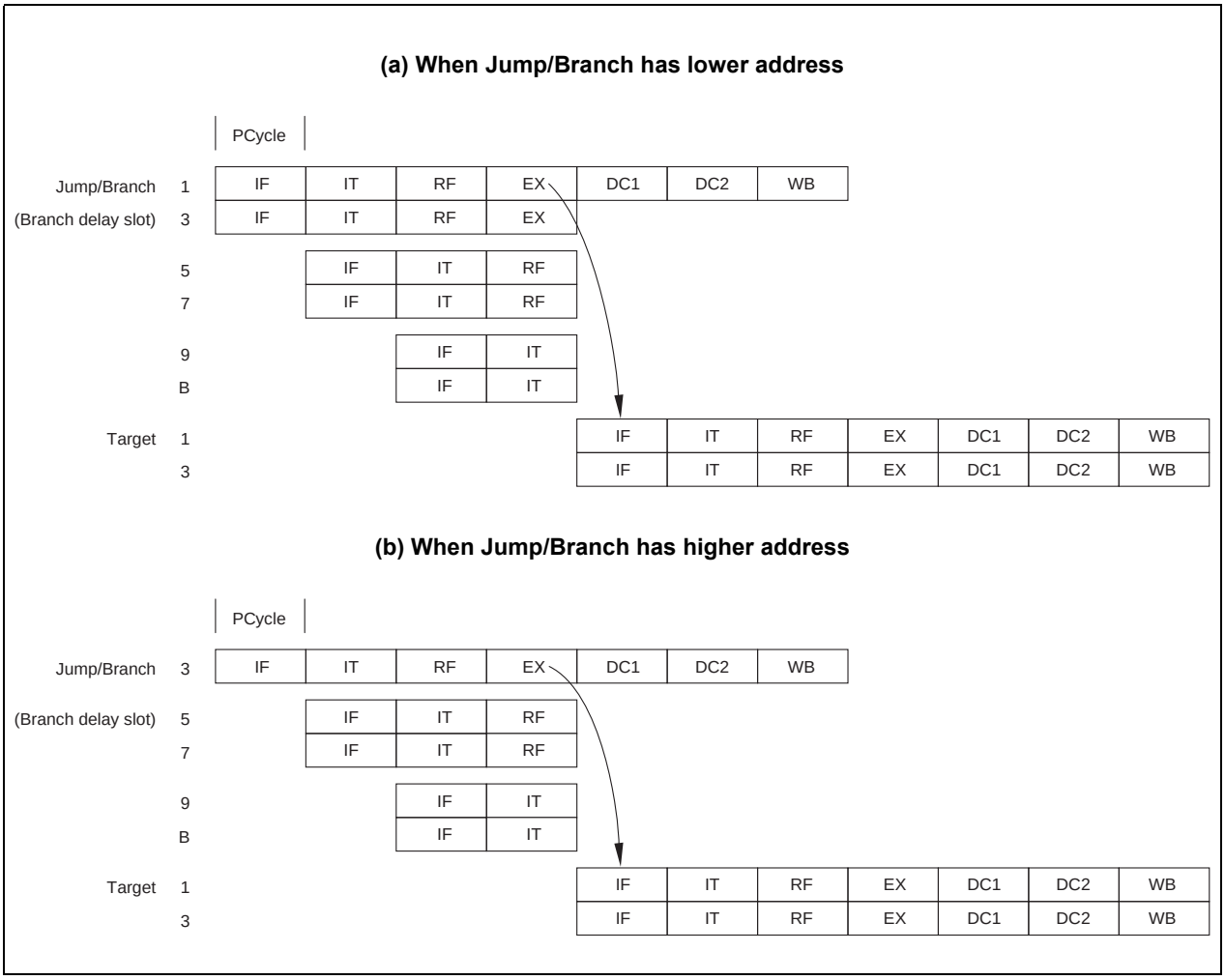


(2) MIPS16 instruction mode

In the VR4131, three cycles of branch delay occurs when a branch condition is met. An instruction in the branch delay slot is discarded.

Figure 4-9 illustrates the branch delay and the location of the branch delay slot.

Figure 4-9. Branch Delay (VR4131, MIPS16 Instruction Mode)



4.2.3 VR4181

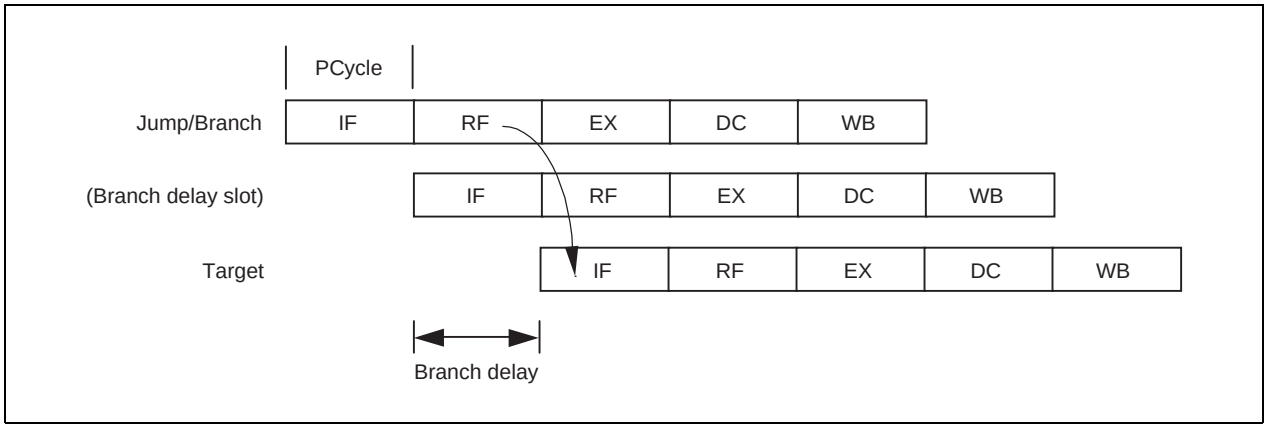
The instruction address generated at the RF stage in the Jump/Branch instruction are available in the IF stage, two instructions later.

In the VR4181, one cycle of branch delay occurs when a branch condition is met in MIPS III instruction mode. An instruction in the branch delay slot is executed (except for Branch Likely instructions).

No branch delay due to a branch instruction occurs in MIPS16 instruction mode. When a branch condition is met, the instruction representing a delay slot is discarded.

Figure 4-10 illustrates the branch delay and the location of the branch delay slot.

Figure 4-10. Branch Delay (VR4181)



4.3 Branch Prediction

The VR4122, VR4131, and VR4181A have a branch prediction mechanism to speed up branch instruction processing.

The VR4122, VR4131, and VR4181A have a full-associative virtual address cache called a branch prediction table. This table holds the history of the branches that have been satisfied recently, using the address of the Branch instruction as a tag and the branch destination address as data.

The VR4122, VR4131, and VR4181A reference the branch prediction table when they fetch a Branch instruction. If the same Branch instruction is in the table (hit), they branch to the branch destination address in the table rather than calculating the branch destination address. If the corresponding Branch instruction is not in the table (miss), they recalculate the branch destination address. If the condition of a missed Branch instruction is satisfied, that Branch instruction and the address of the branch destination are stored in the branch prediction table. New history is written over the entry stored earliest (LRU (least recently used) algorithm).

The branch prediction table of the VR4122 and VR4181A can hold four entries, and that of the VR4131 can hold eight entries.

Whether the branch prediction mechanism is to be used can be specified by using the BP bit of the Config register of CP0. Branch prediction is executed when the BP bit is cleared to 0; it is not executed when the bit is set to 1. The BP bit is cleared to 0 by default.

Branch prediction is not executed in the MIPS16 instruction mode and debug mode. The BP bit is automatically set to 1.

Because the branch prediction table is a virtual address cache, it is invalid if the contents of a physical address corresponding to a virtual address change. When performing an operation that rewrites the text area (such as changing the bank or downloading), therefore, either disable branch prediction (by setting the BP bit to 1) or clear the history of the branch prediction table immediately before. Clear the history regardless of whether the VR4122, VR4131, or VR4181A operates in the virtual address mode. The VR4122, VR4131, and VR4181A clear the history of the branch prediction table in the following cases.

- Writing to EntryHi register
- Writing to Config register (VR4131 only)
- Execution of TLBWI instruction
- Execution of TLBWR instruction
- Execution of TLBR instruction

4.3.1 VR4122, VR4181A

The VR4122 and VR4181A reference the branch prediction table in the IF stage of a Branch instruction. If a hit occurs when the branch condition is decoded in the RF stage, the instruction at the branch destination address output from the branch prediction table is fetched.

When the branch condition is checked in the EX stage and it has been ascertained that a branch is to occur, the pipeline processing of the instruction at the branch destination continues. If it has been found that a branch is not to occur, the processing of the instruction at the branch destination is stopped, and the next instruction in the branch delay slot is fetched in the DC stage.

If it is found that the condition of a Branch instruction missed in the branch prediction table is satisfied and that a branch is to occur, the branch prediction table is updated in the DC stage.

The figure below illustrates the pipeline operation when branch prediction is performed.

Figure 4-11. Pipeline on Branch Prediction (VR4122, VR4181A) (1/2)

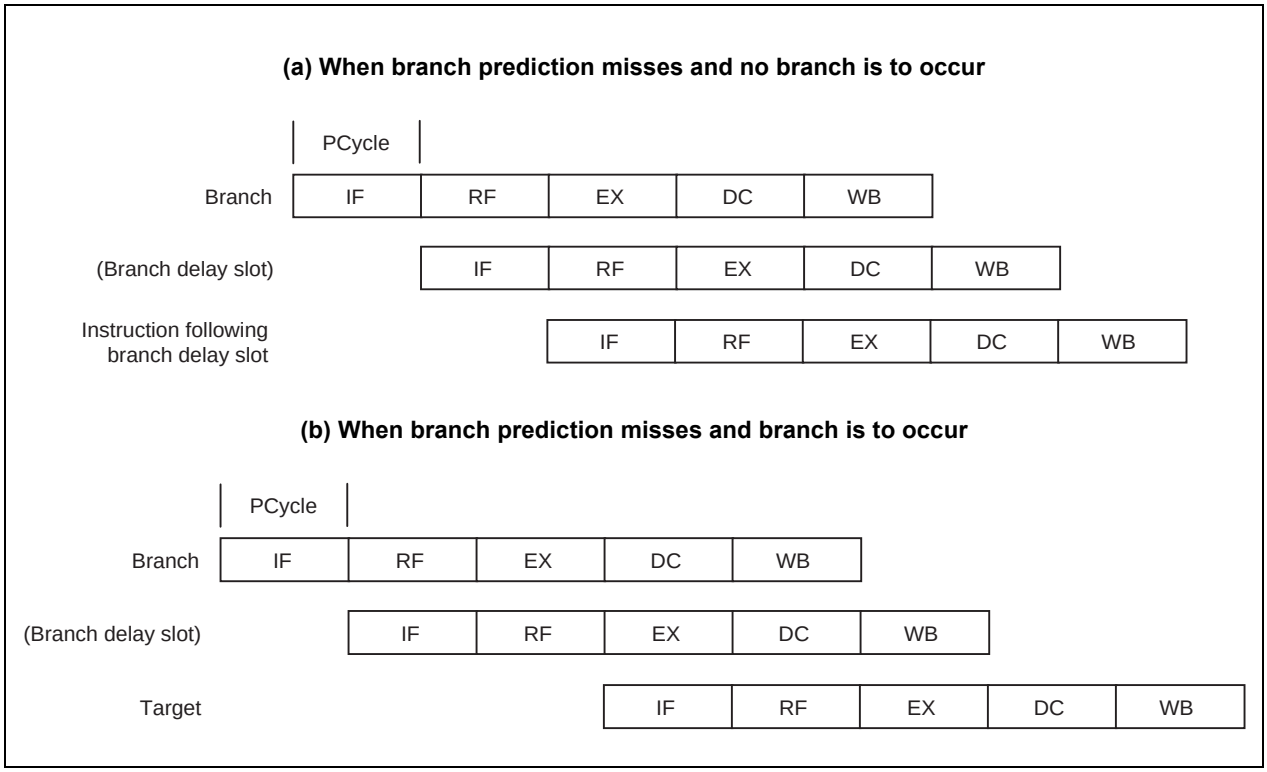
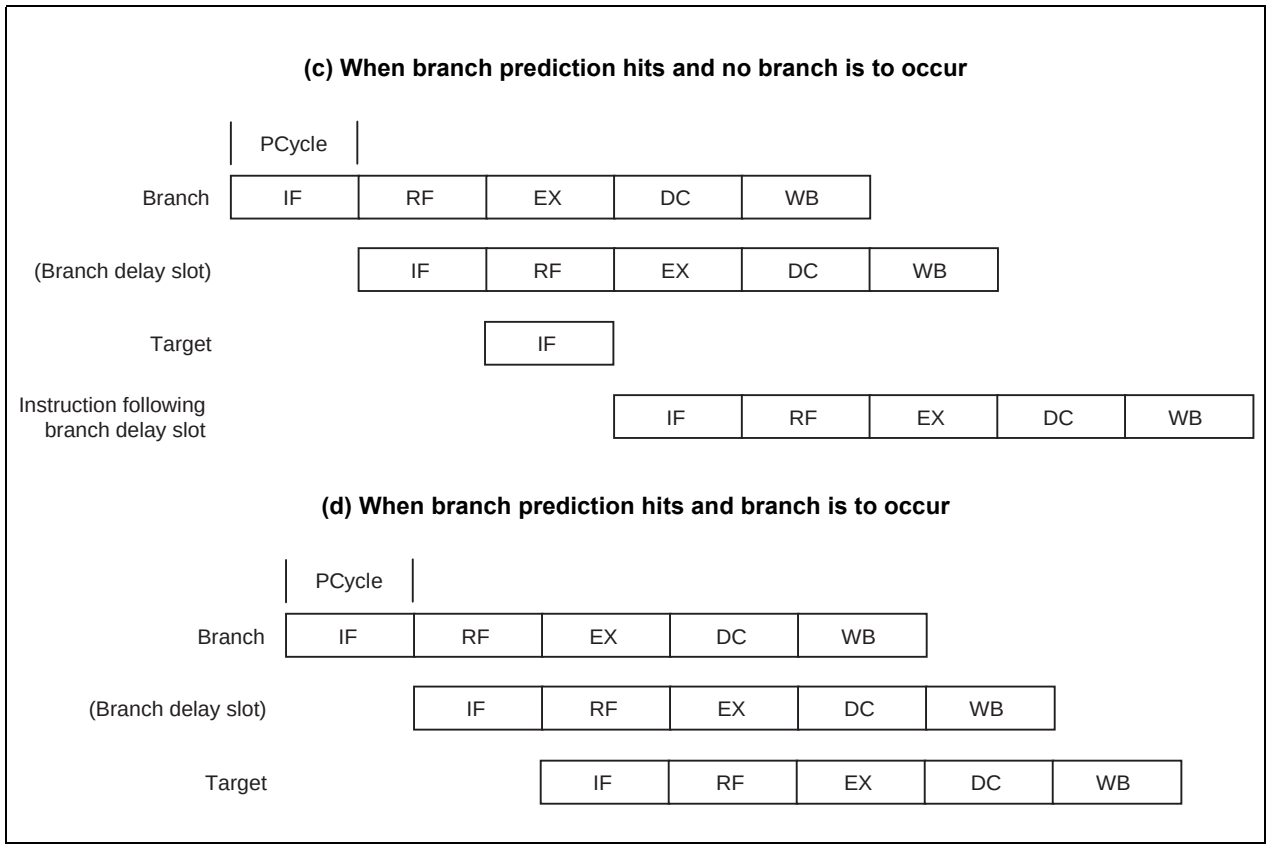


Figure 4-11. Pipeline on Branch Prediction (VR4122, VR4181A) (2/2)



4.3.2 VR4131

The VR4131 references the branch prediction table in the IF stage of a Branch instruction. If a hit occurs, the instruction at the branch destination address output from the branch prediction table is fetched.

When the branch condition is checked in the EX stage and it has been ascertained that a branch is to occur, the pipeline processing of the instruction at the branch destination continues. If it has been found that a branch is not to occur, the processing of the instruction at the branch destination is stopped, and the next instruction in the branch delay slot is fetched in the DC stage.

If it is found that the condition of a Branch instruction missed in the branch prediction table is satisfied and that a branch is to occur, the branch prediction table is updated in the DC stage.

The figure below illustrates the pipeline operation when branch prediction is performed.

Figure 4-12. Pipeline on Branch Prediction (VR4131, When the Branch Is in the Lower Address) (1/2)

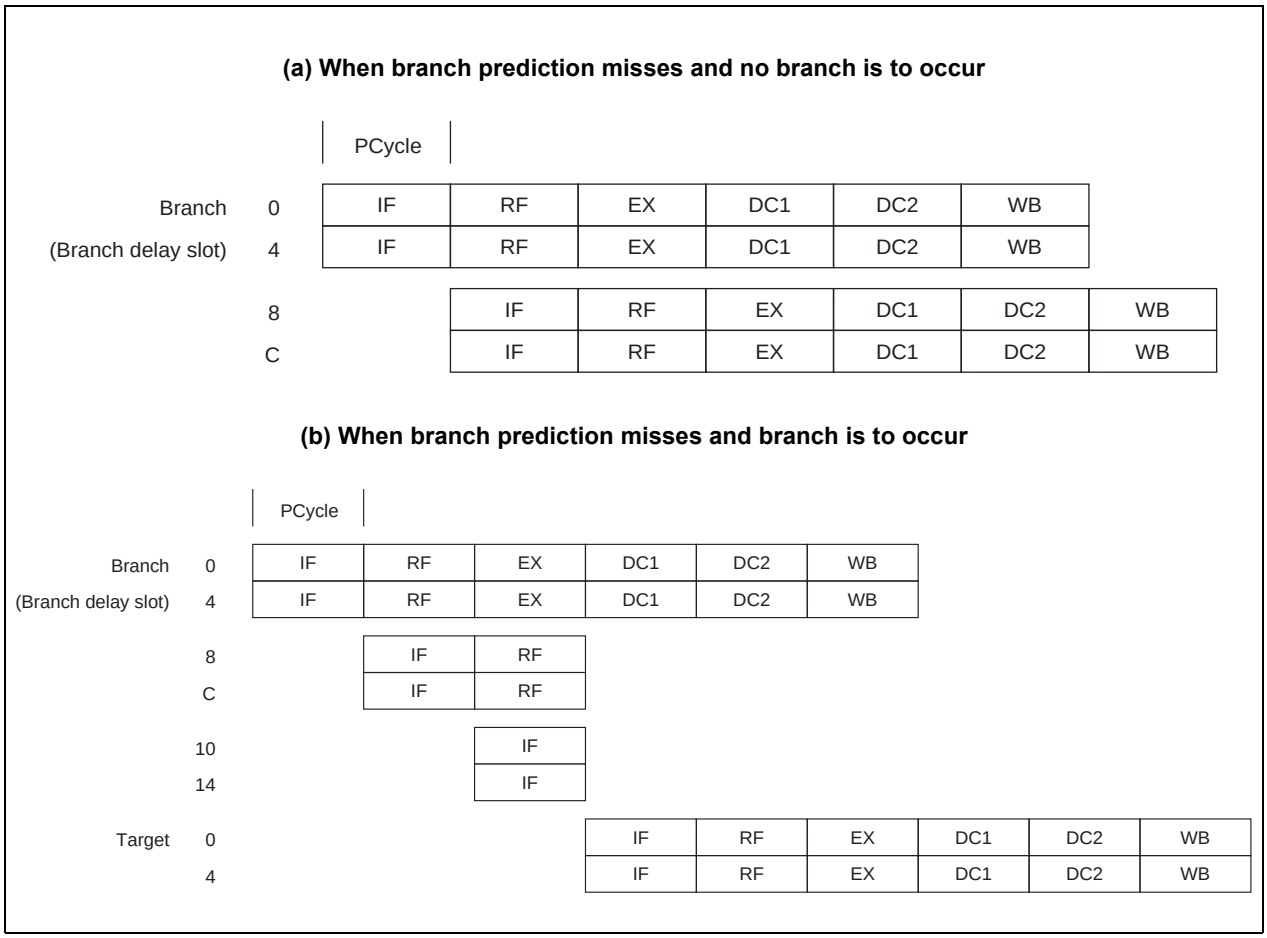


Figure 4-12. Pipeline on Branch Prediction (VR4131, When the Branch Is in the Lower Address) (2/2)

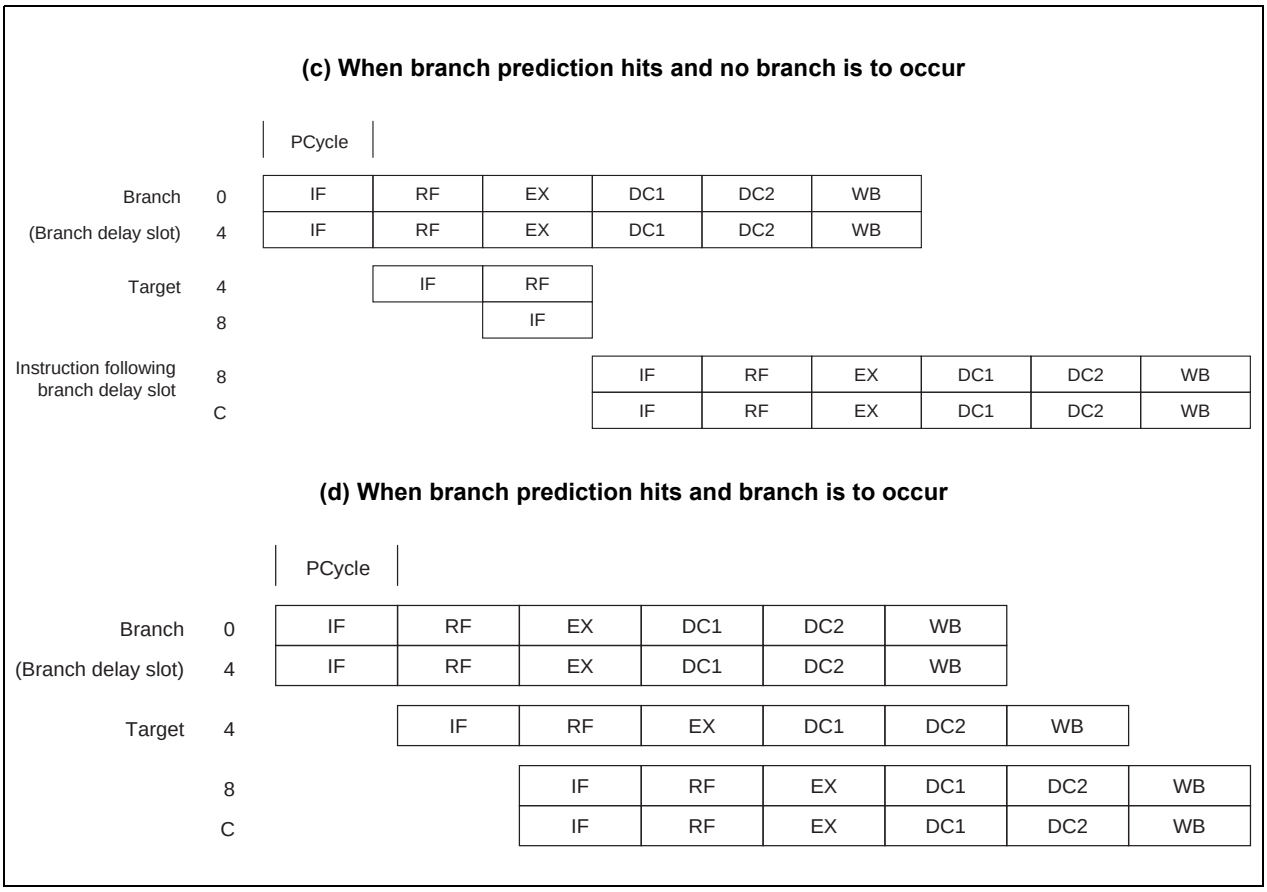


Figure 4-13. Pipeline on Branch Prediction (VR4131, When the Branch Is in the Higher Address) (1/2)

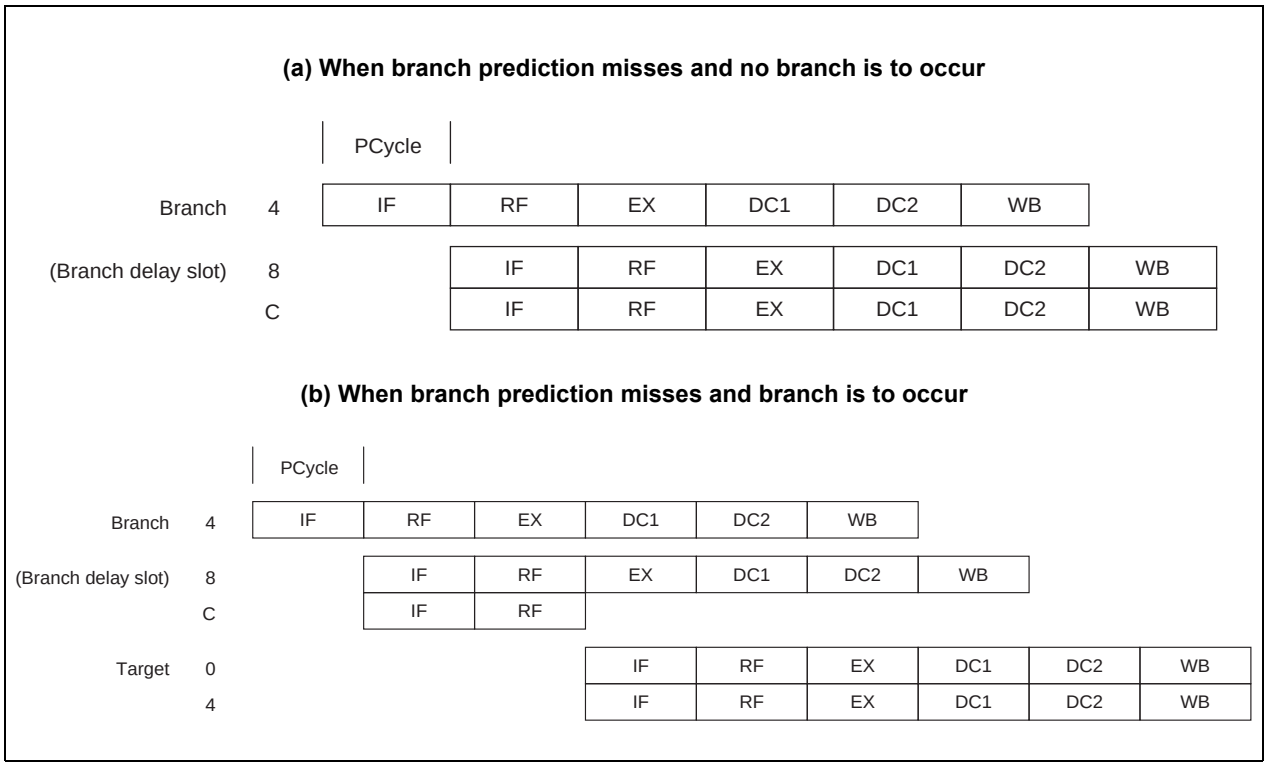
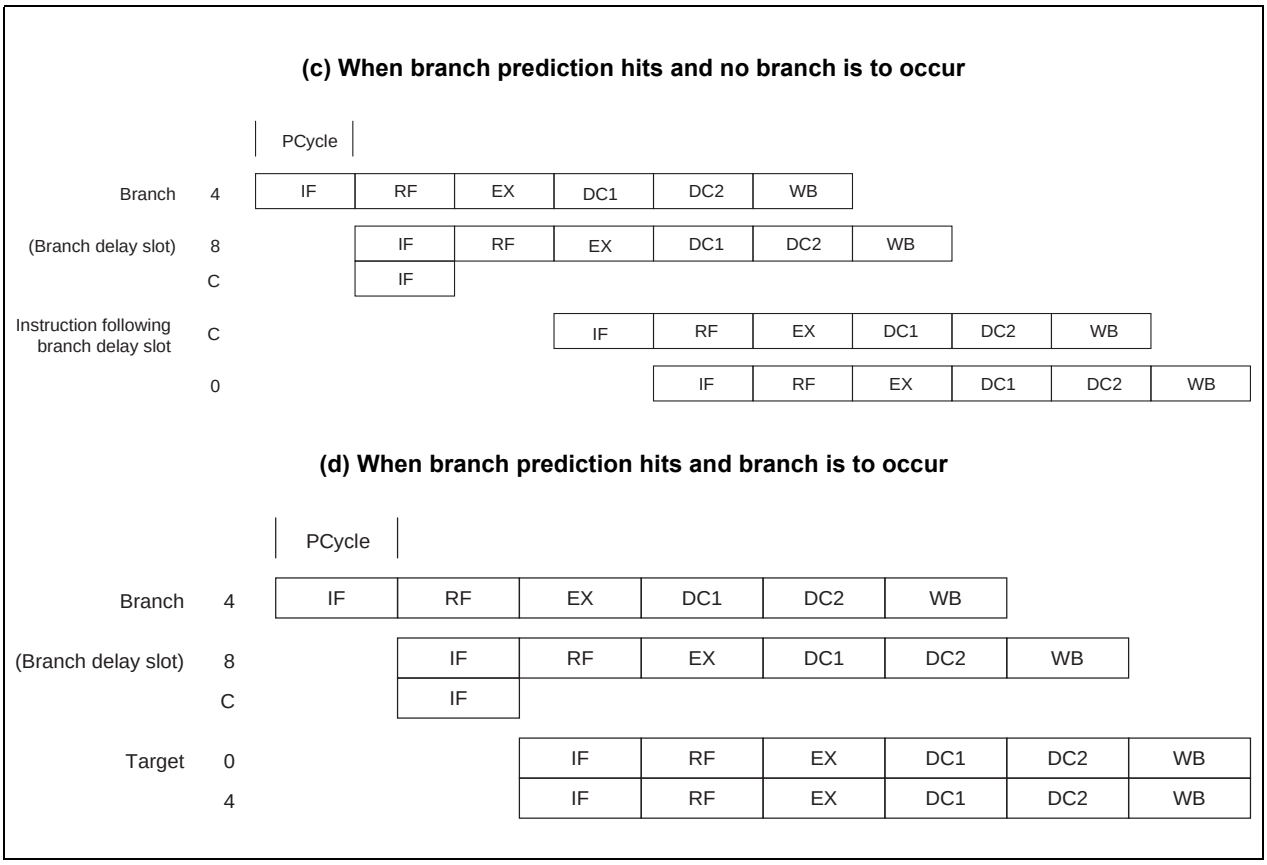


Figure 4-13. Pipeline on Branch Prediction (VR4131, When the Branch Is in the Higher Address) (2/2)



4.4 Load Delay

The instruction location immediately following a load instruction is referred to as the load delay slot.

The instruction in a load delay slot can use the contents of the loaded register, however in such cases hardware interlocks insert additional delay cycles. Consequently, scheduling load delay slots can be desirable, both for performance and VR-Series processor compatibility.

In the VR4121, VR4122, and VR4181A, two cycles of DC stage are necessary during a load instruction execution for data read from the data cache and data alignment, and therefore hardware automatically causes interlock.

4.5 Instruction Streaming

If a miss occurs in the instruction cache, a cycle to refill instructions from the main memory to the instruction cache is started. At this time, the VR4122, VR4131, and VR4181A continue pipeline processing while writing data (instructions) to the instruction cache and bypassing the data (instructions) to the instruction decoder of the CPU. Therefore, processing can be resumed earlier from a stall that takes place if a miss occurs in the instruction cache. This instruction data bypassing function is called streaming.

The instruction streaming function is enabled or disabled by the IS bit of the Config register of CP0. Instruction streaming is executed when the IS bit is cleared to 0; it is not executed when the bit is set to 1. The IS bit is cleared to 0 by default.

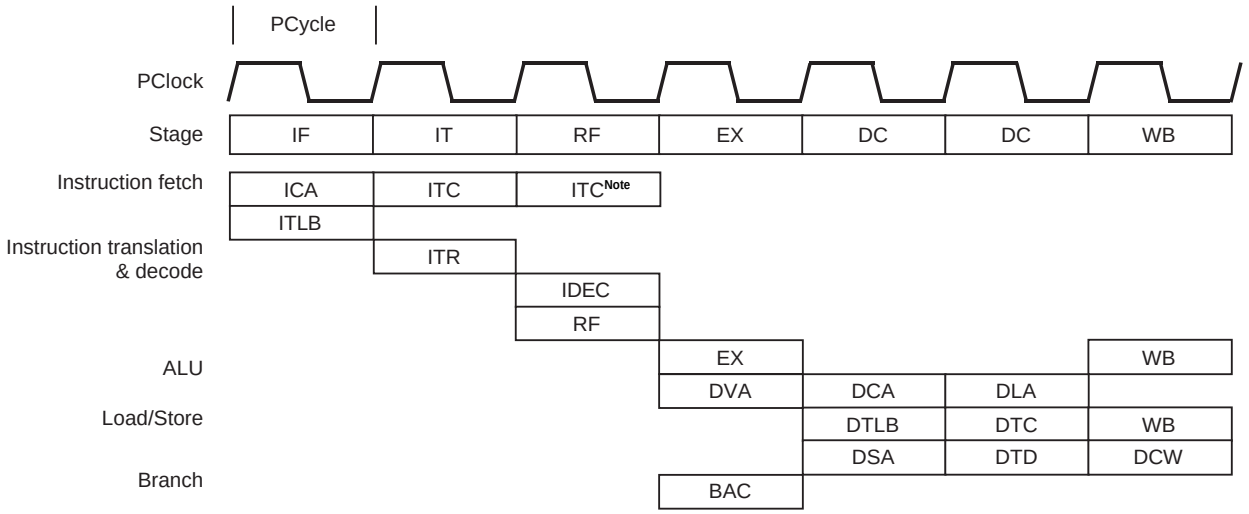
If instruction streaming is not executed, the pipeline is stalled until refilling the instruction cache has been completed.

4.6 Pipeline Activities

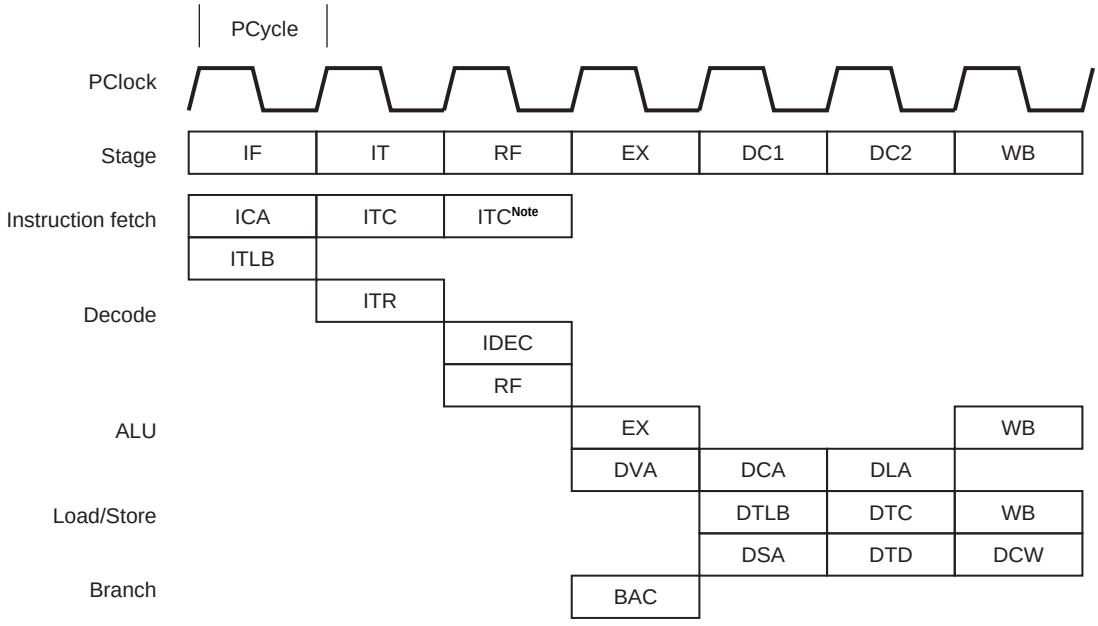
Figure 4-14 shows the activities that can occur during each pipeline stage; Table 4-1 describes these pipeline activities.

Figure 4-14. Pipeline Activities (1/2)

(a) VR4121, VR4122, and VR4181A



(b) VR4131



Note When MIPS III instruction mode

Figure 4-14. Pipeline Activities (2/2)

(c) VR4181

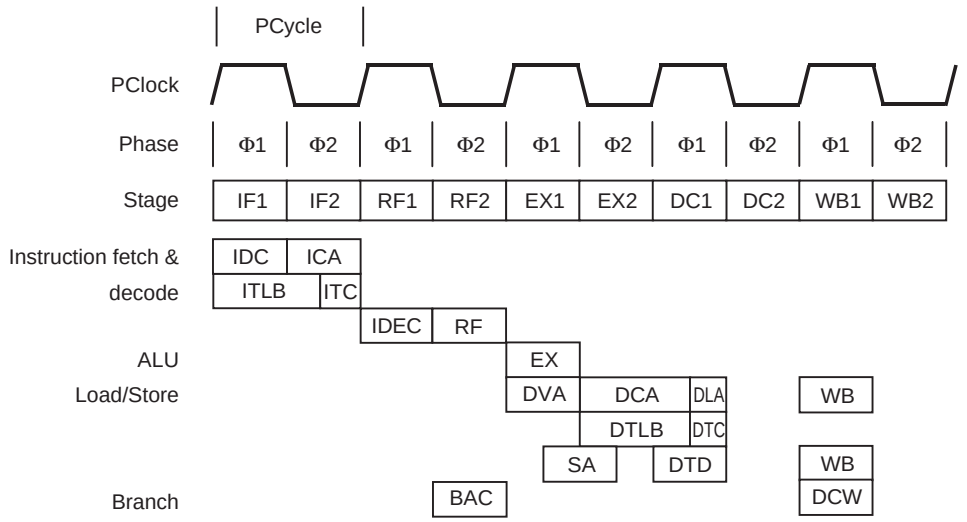


Table 4-1. Description of Pipeline Activities during Each Stage

Mnemonic	Description
IDC	Instruction cache address decode
ITLB	Instruction address translation
ICA	Instruction cache array access
ITR	Instruction translation
ITC	Instruction tag check
IDEC	Instruction decode
RF	Register operand fetch
BAC	Branch address calculation
EX	Execution stage
DVA	Data virtual address calculation
SA/DSA	Store align
DCA	Data cache address decode/array access
DTLB	Data address translation
DLA	Data cache load align
DTC	Data tag check
DTD	Data transfer to data cache
DCW	Data cache write
WB	Write back to register file

The operation of the pipeline is illustrated by the following examples that describe how typical instructions are executed. Each instruction is taken through the pipeline and the operations that occur in each relevant stage are described.

(1) Add instruction (ADD rd, rs, rt)

IF stage The eleven least-significant bits of the virtual address are used to access the instruction cache. Then the cache index is compared with the page frame number and the cache data is read out. The virtual PC is incremented by 4 so that the next instruction can be fetched.

IT stage A MIPS16 instruction is translated into a 32-bit length instruction (VR4121, VR4122, VR4131, and VR4181A only).

RF stage The 2-port register file is addressed with the rs and rt fields and the register data is valid at the register file output. At the same time, bypass multiplexers select inputs from either the EX- or DC-stage output in addition to the register file output, depending on the need for an operand bypass.

EX stage The operands flow into the ALU inputs, and the ALU operation is started. The result of the ALU operation is latched into the ALU output latch.

DC stage This stage is a NOP for this instruction. The data from the output of the EX stage (the ALU) is moved into the output latch of the DC.

WB stage The WB latch feeds the data to the inputs of the register file, which is accessed by the rd field. The data is written into the file.

Figure 4-15. ADD Instruction Pipeline Activities (VR4121, VR4122, VR4181A)

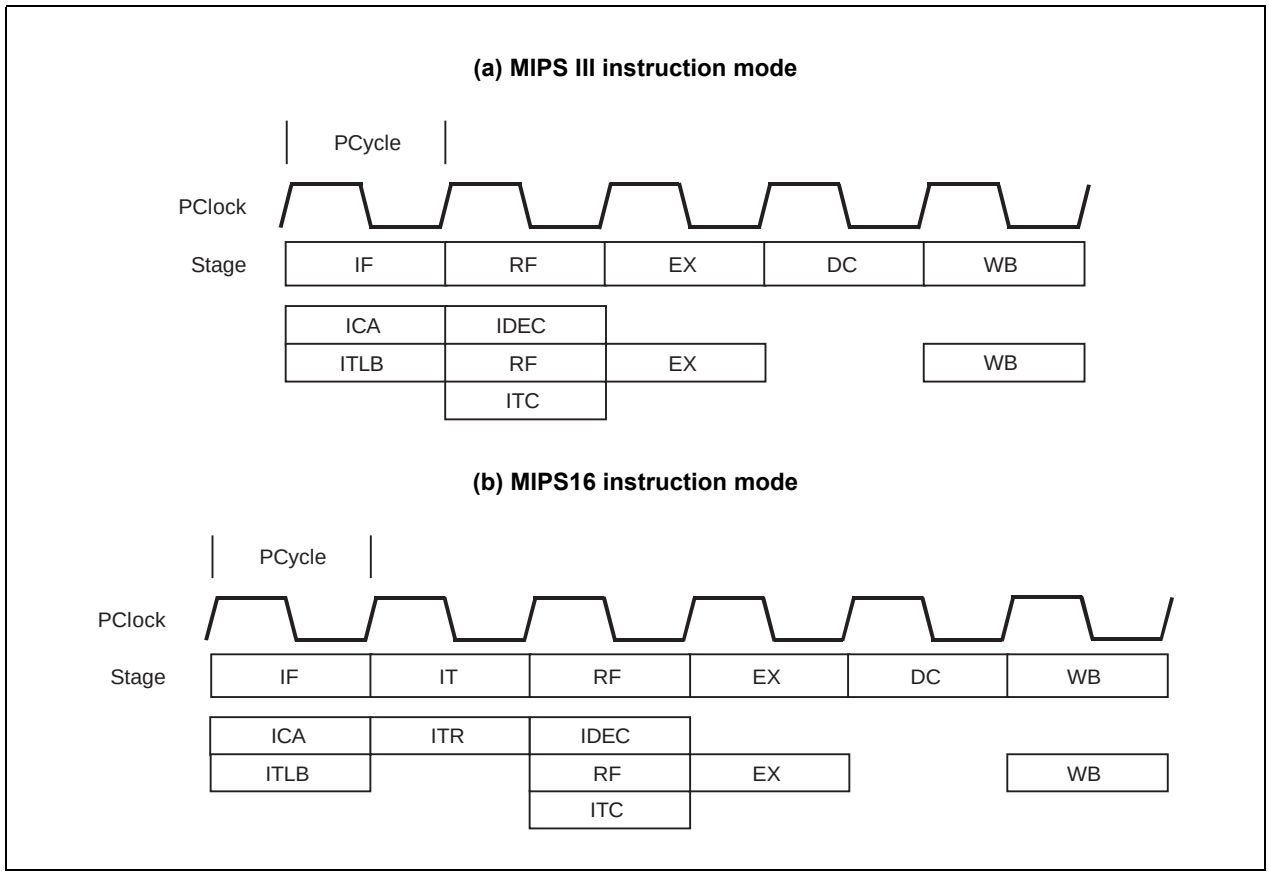


Figure 4-16. ADD Instruction Pipeline Activities (VR4131)

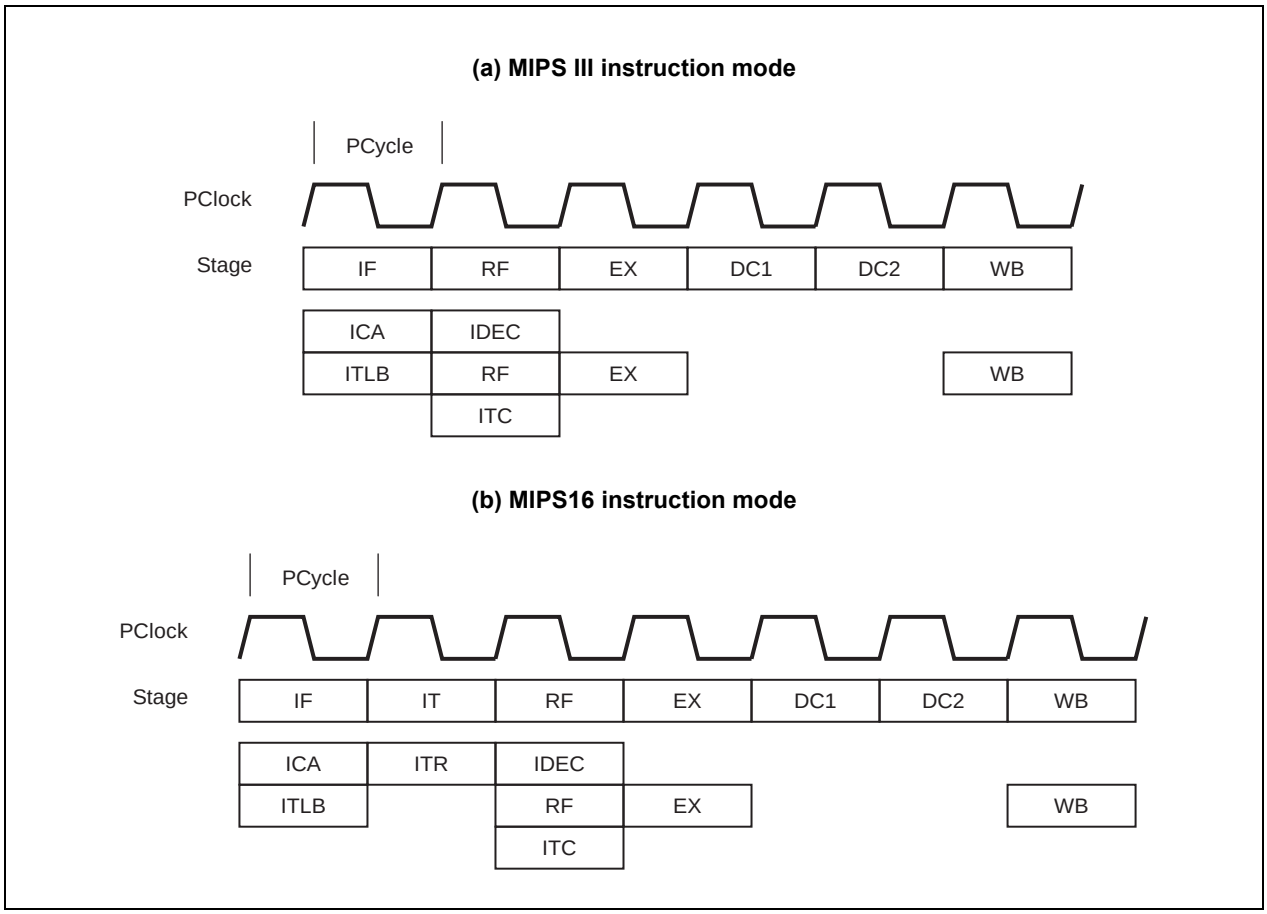
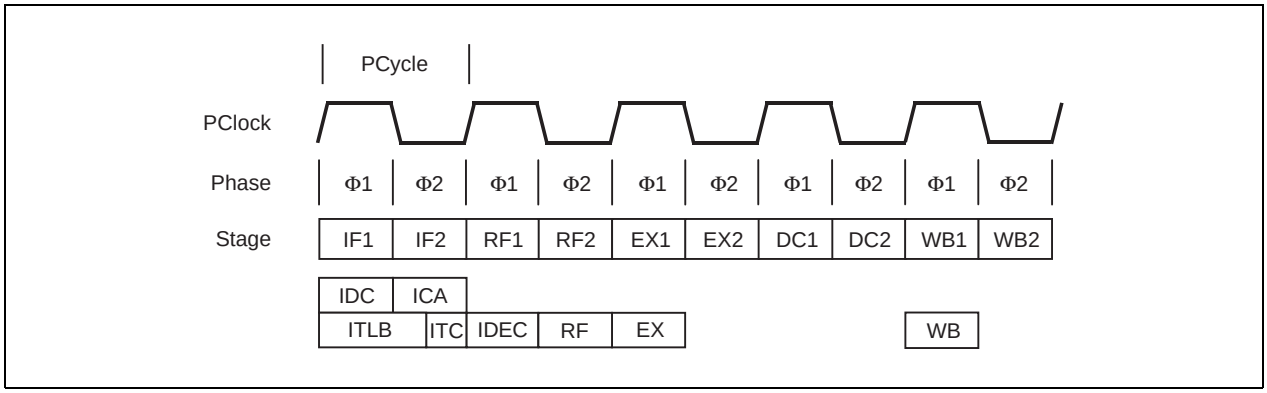


Figure 4-17. ADD Instruction Pipeline Activities (VR4181)



(2) Jump and Link Register instruction (JALR rd, rs)

IF stage Same as the IF stage for the ADD instruction.

IT stage Same as the IT stage for the ADD instruction (VR4121, VR4122, VR4131, and VR4181A only).

RF stage A register specified in the rs field is read from the file, and the value read from the rs register is input to the virtual PC latch synchronously. This value is used to fetch an instruction at the jump destination. The value of the virtual PC incremented during the IF stage is incremented again to produce the link address PC + 8 (PC + 4 in MIPS16 instruction mode) where PC is the address of the JALR instruction. The resulting value is the PC to which the program will eventually return. This value is placed in the Link output latch of the Instruction Address unit.

EX stage The PC + 8 (PC + 4 in MIPS16 instruction mode) value is moved from the Link output latch to the output latch of the EX stage.

DC stage The PC + 8 (PC + 4 in MIPS16 instruction mode) value is moved from the output latch of the EX stage to the output latch of the DC stage.

WB stage Refer to the ADD instruction. Note that if no value is explicitly provided for rd then register 31 is used as the default. If rd is explicitly specified, it cannot be the same register addressed by rs; if it is, the result of executing such an instruction is undefined.

Figure 4-18. JALR Instruction Pipeline Activities (VR4121, VR4122, VR4181A)

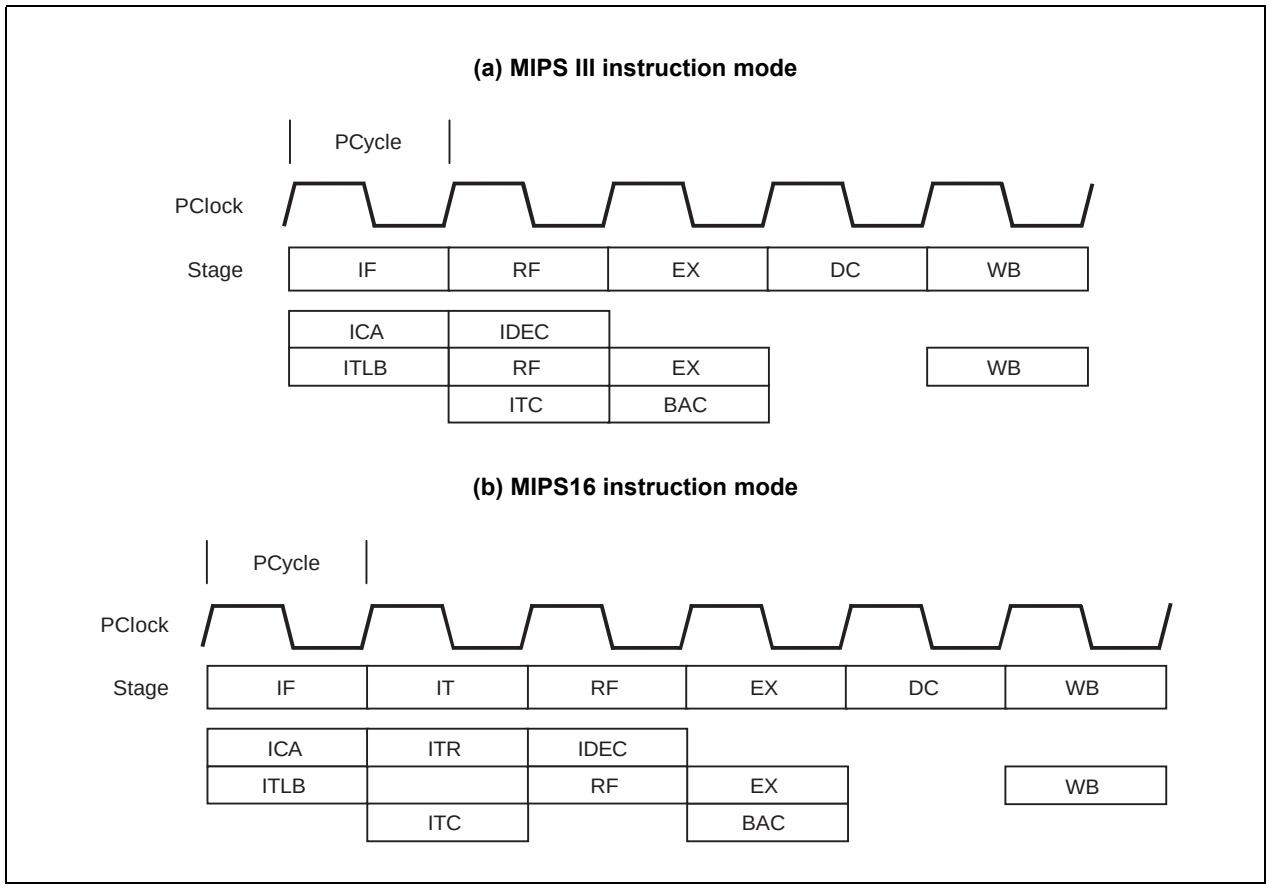


Figure 4-19. JALR Instruction Pipeline Activities (VR4131)

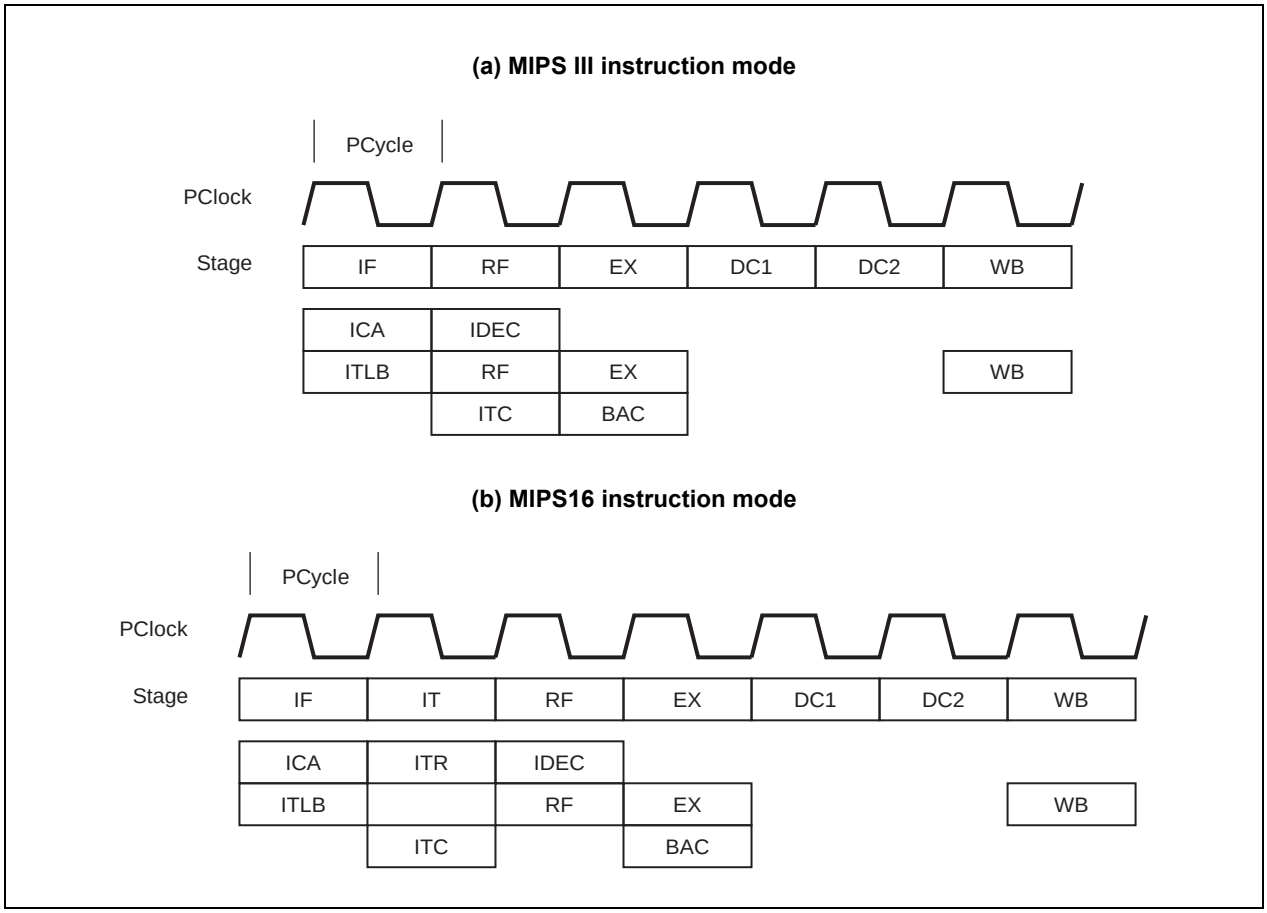
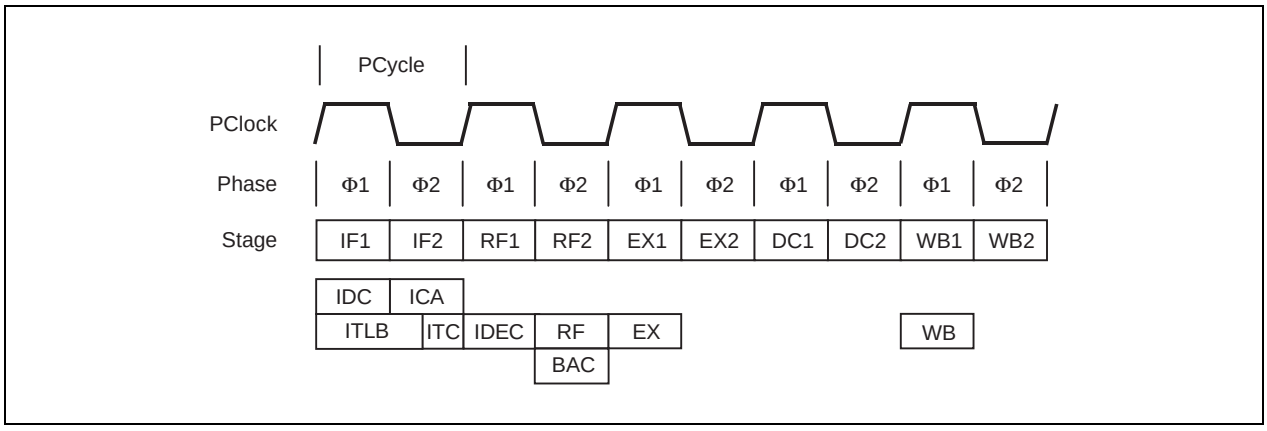


Figure 4-20. JALR Instruction Pipeline Activities (VR4181)



(3) Branch on Equal instruction (BEQ rs, rt, offset)

IF stage Same as the IF stage for the ADD instruction.

IT stage Same as the IT stage for the ADD instruction (VR4121, VR4122, VR4131, and VR4181A only).

RF stage The register file is addressed with the rs and rt fields. A check is performed to determine if each corresponding bit position of these two operands has equal values. If they are equal, the PC is set to PC + target, where target is the sign-extended offset field. If they are not equal, the PC is set to PC + 4.

EX stage The next PC resulting from the branch comparison is valid at the beginning of instruction fetch.

DC stage This stage is a NOP for this instruction.

WB stage This stage is a NOP for this instruction.

Figure 4-21. BEQ Instruction Pipeline Activities (VR4121, VR4122, VR4181A)

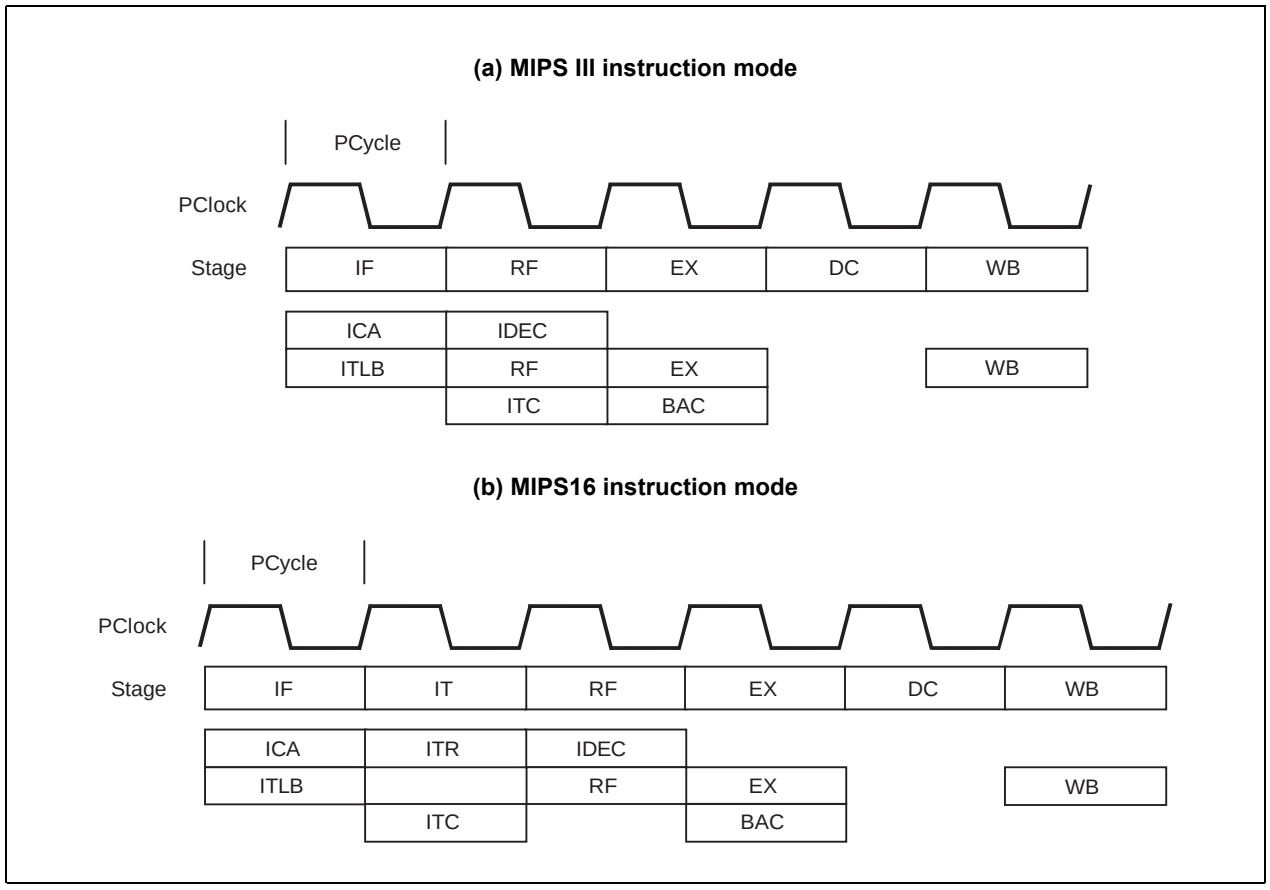


Figure 4-22. BEQ Instruction Pipeline Activities (VR4131)

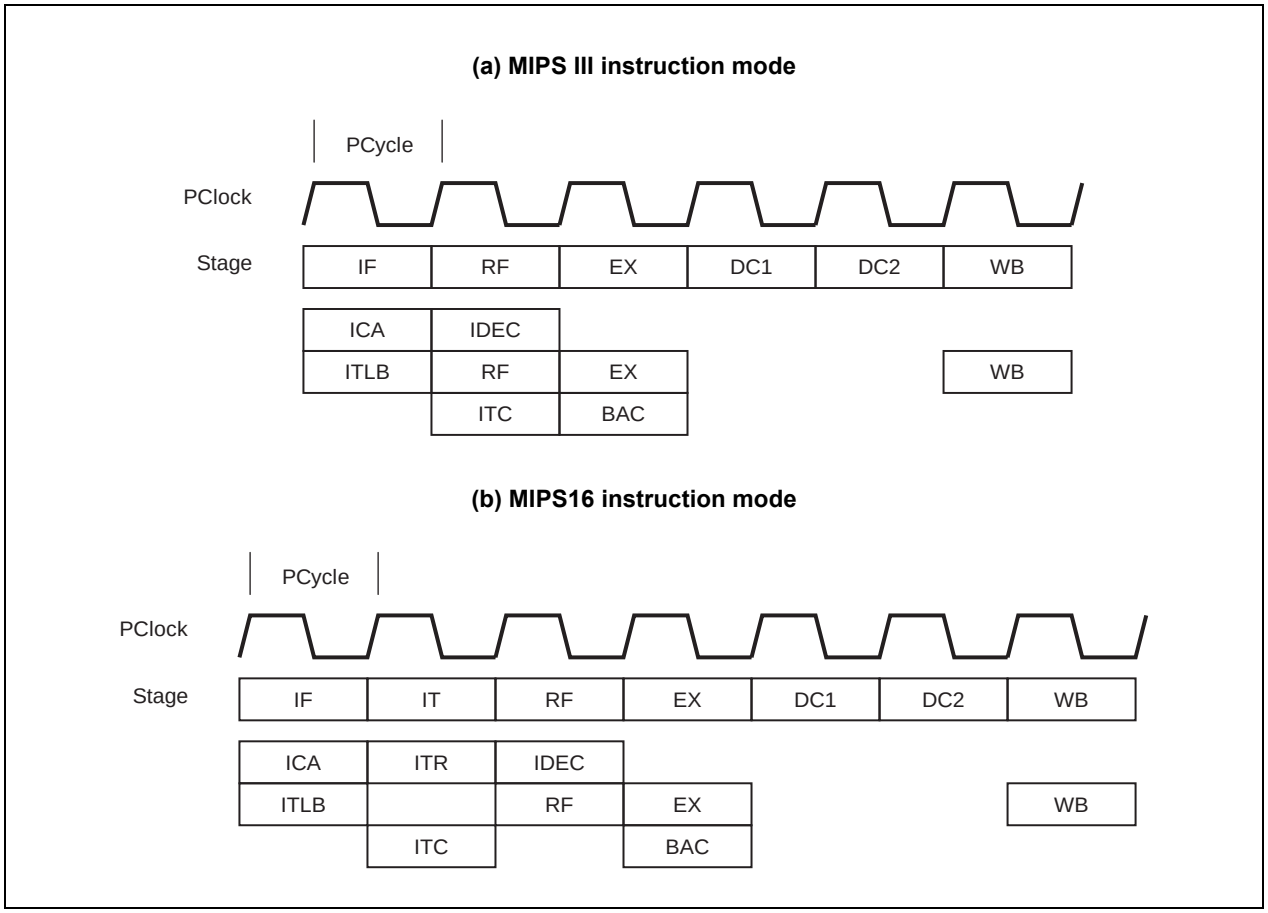
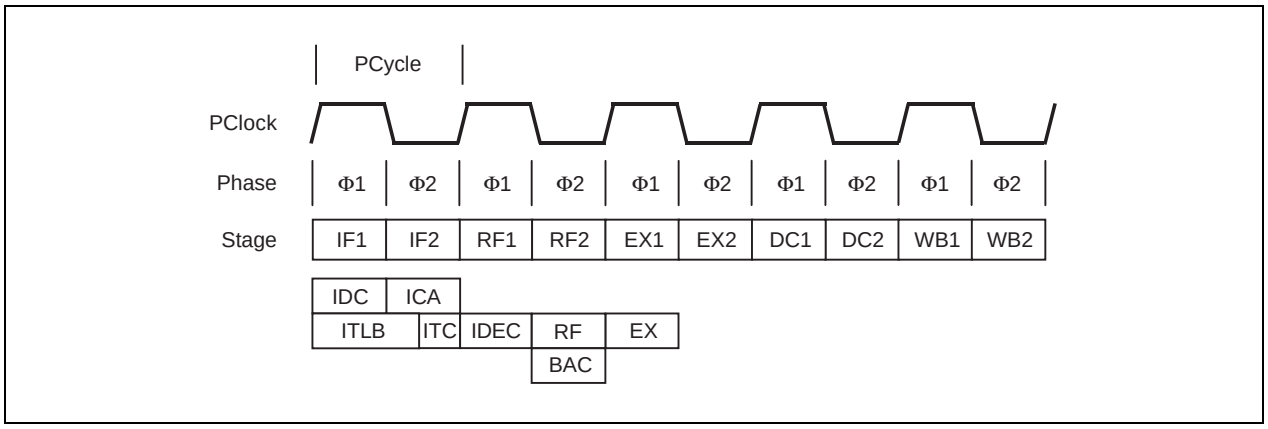


Figure 4-23. BEQ Instruction Pipeline Activities (VR4181)



(4) Trap if Less Than instruction (TLT rs, rt)

Remark TLT instruction is not included in the MIPS16 instruction set.

IF stage Same as the IF stage for the ADD instruction.

RF stage Same as the RF stage for the ADD instruction.

EX stage ALU controls are set to do an A – B operation. The operands flow into the ALU inputs, and the ALU operation is started. The result of the ALU operation is latched into the ALU output latch. The sign bits of operands and of the ALU output latch are checked to determine if a less than condition is true. If this condition is true, a Trap exception occurs. The value in the PC register is used as an exception vector value, and from now on any instruction will be invalid.

DC stage This stage is a NOP for this instruction.

WB stage The EPC register is loaded with the value of the PC if the less than condition was met in the EX stage. The Cause register ExCode field and BD bit are updated appropriately, as is the EXL bit of the Status register. If the less than condition was not met in the EX stage, no activity occurs in the WB stage.

Figure 4-24. TLT Instruction Pipeline Activities (VR4121, VR4122, VR4181A)

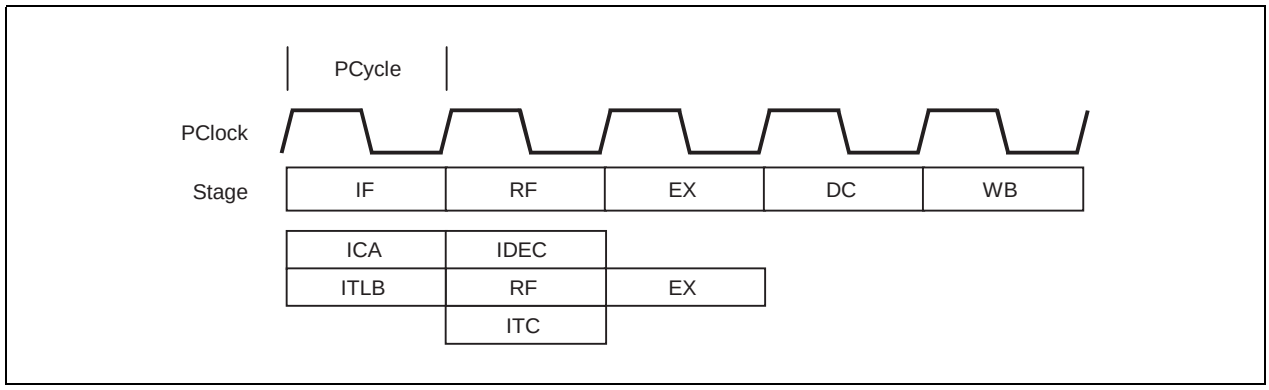


Figure 4-25. TLT Instruction Pipeline Activities (VR4131)

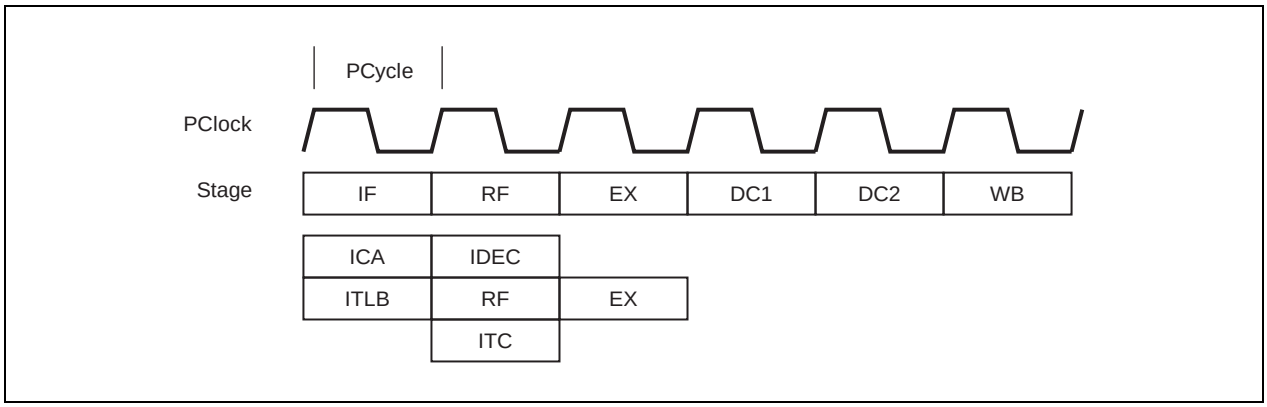
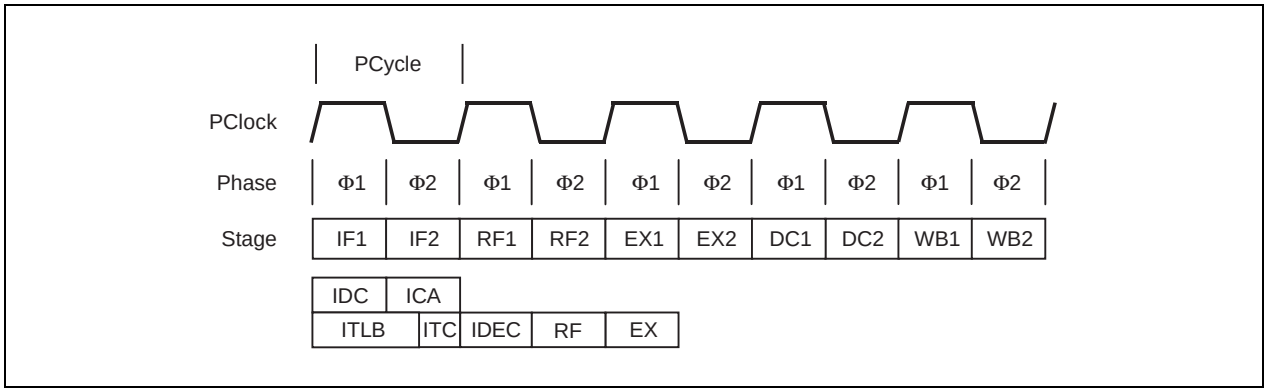


Figure 4-26. TLT Instruction Pipeline Activities (VR4181)



(5) Load Word instruction (LW rt, offset (base))

IF stage Same as the IF stage for the ADD instruction.

IT stage Same as the IT stage for the ADD instruction (VR4121, VR4122, VR4131, and VR4181A only).

RF stage Same as the RF stage for the ADD instruction. Note that the base field is in the same position as the rs field.

EX stage Refer to the EX stage for the ADD instruction. For LW, the inputs to the ALU come from GPR[base] through the bypass multiplexer and from the sign-extended offset field. The result of the ALU operation that is latched into the ALU output latch represents the effective virtual address of the operand (DVA).

DC stage The cache tag field is compared with the Page Frame Number (PFN) field of the TLB entry. After passing through the load aligner, aligned data is placed in the DC output latch.

DC2 stage After passing through the load aligner, aligned data is placed in the DC2 output latch (VR4121, VR4122, VR4131, and VR4181A only).

WB stage The cache read data is written into the register file addressed by the rt field.

Figure 4-27. LW Instruction Pipeline Activities (VR4121, VR4122, VR4181A)

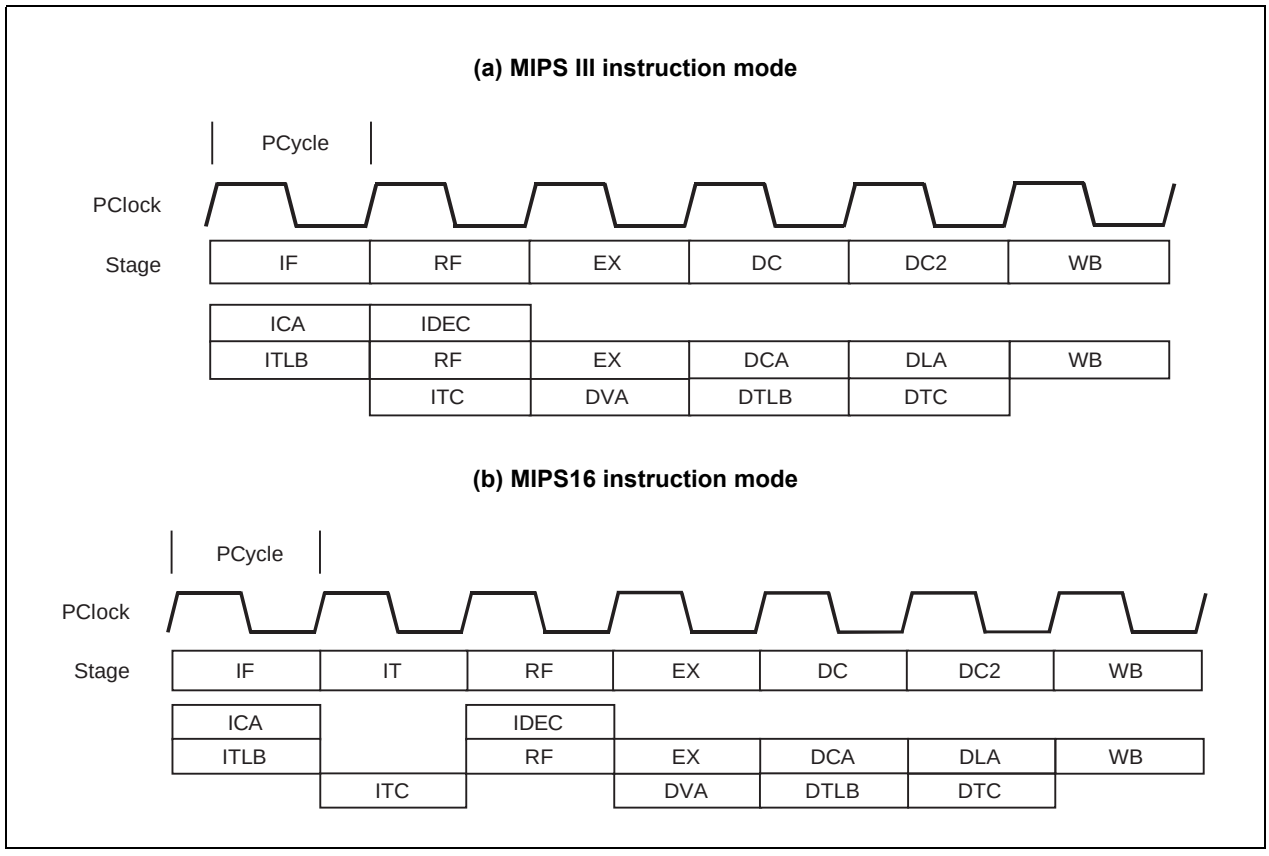


Figure 4-28. LW Instruction Pipeline Activities (VR4131)

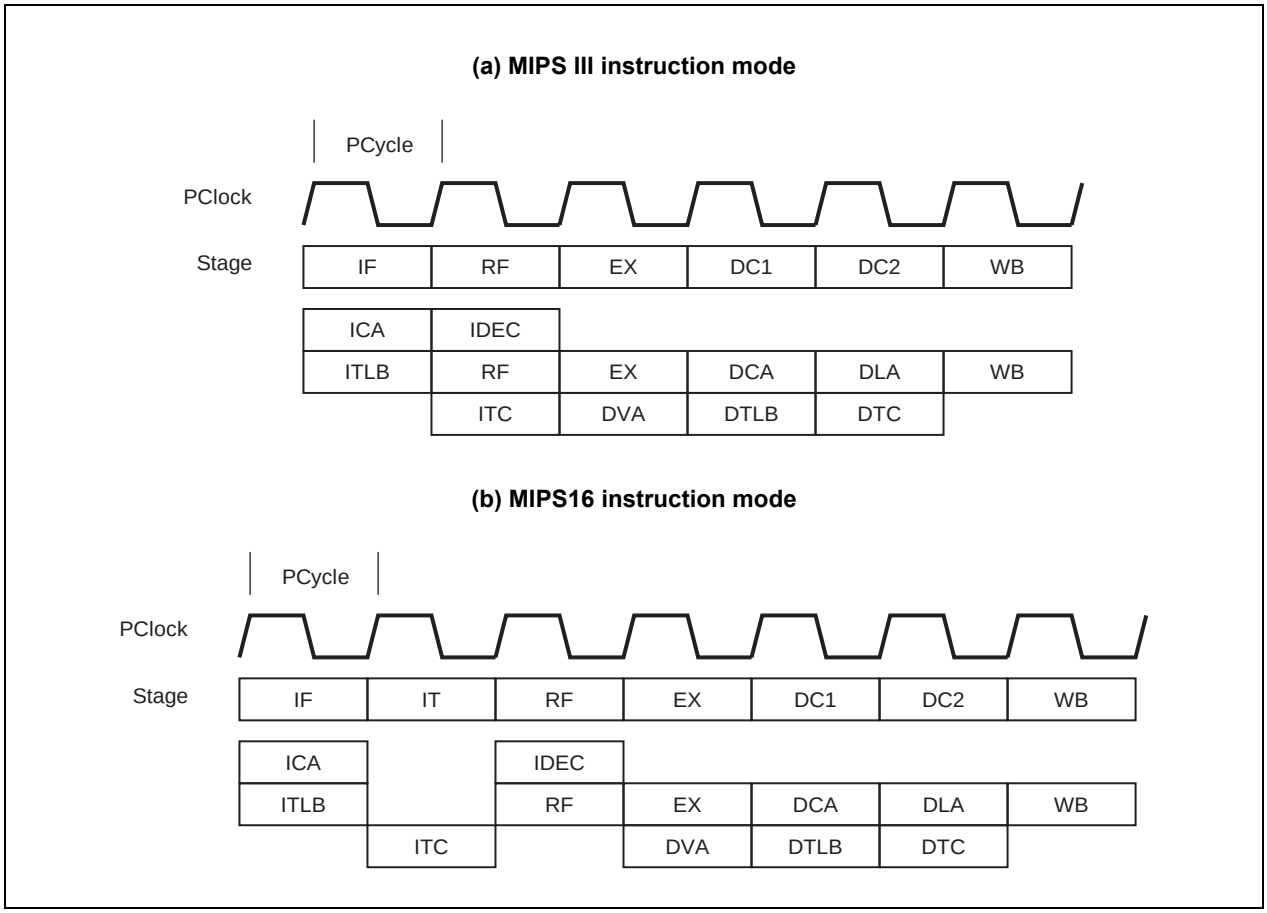
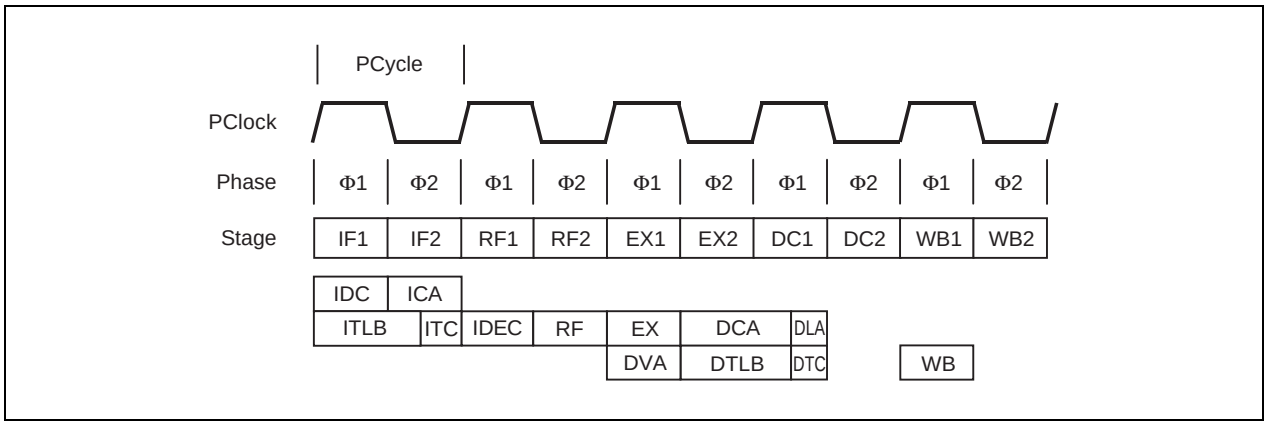


Figure 4-29. LW Instruction Pipeline Activities (VR4181)



(6) Store Word instruction (SW rt, offset (base))

IF stage — Same as the IF stage for the ADD instruction.

IT stage — Same as the IT stage for the ADD instruction (VR4121, VR4122, VR4131, and VR4181A only).

RF stage — Same as the RF stage for the LW instruction.

EX stage — Refer to the LW instruction for a calculation of the effective address. From the RF output latch, the GPR[rt] is sent through the bypass multiplexer and into the main shifter. The results of the ALU are latched in the output latches.

DC stage — Refer to the LW instruction for a description of the cache access. The store data is aligned.

DC2 stage — Refer to the LW instruction for a description of the cache access (VR4121, VR4122, VR4131, and VR4181A only).

WB stage — If there was a cache hit, the content of the store data output latch is written into the data cache at the appropriate word location.
Note that all store instructions use the data cache for two consecutive PCycles. If the following instruction requires use of the data cache, the pipeline is slipped for one PCycle to complete the writing of an aligned store data.

Figure 4-30. SW Instruction Pipeline Activities (VR4121, VR4122, VR4181A)

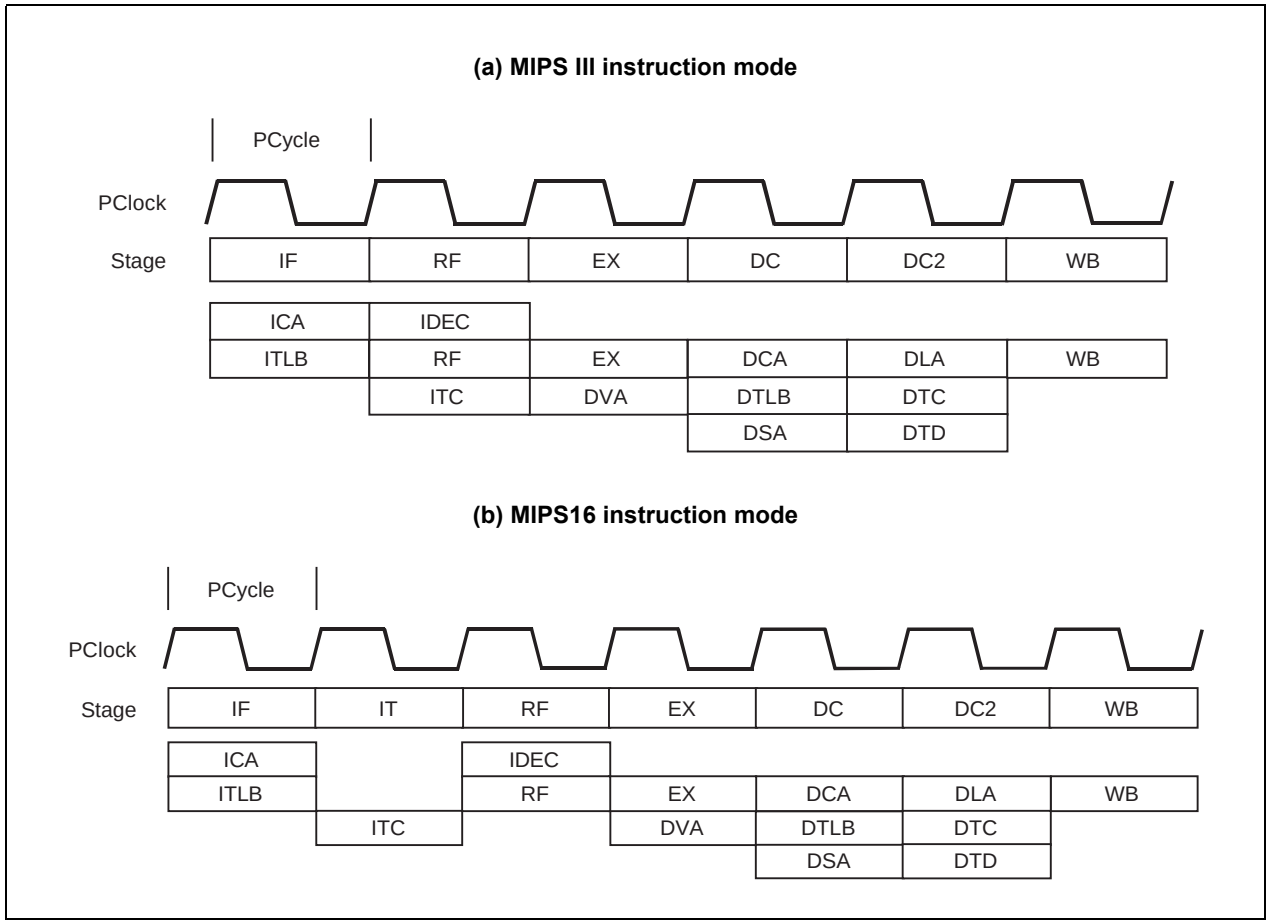


Figure 4-31. SW Instruction Pipeline Activities (VR4131)

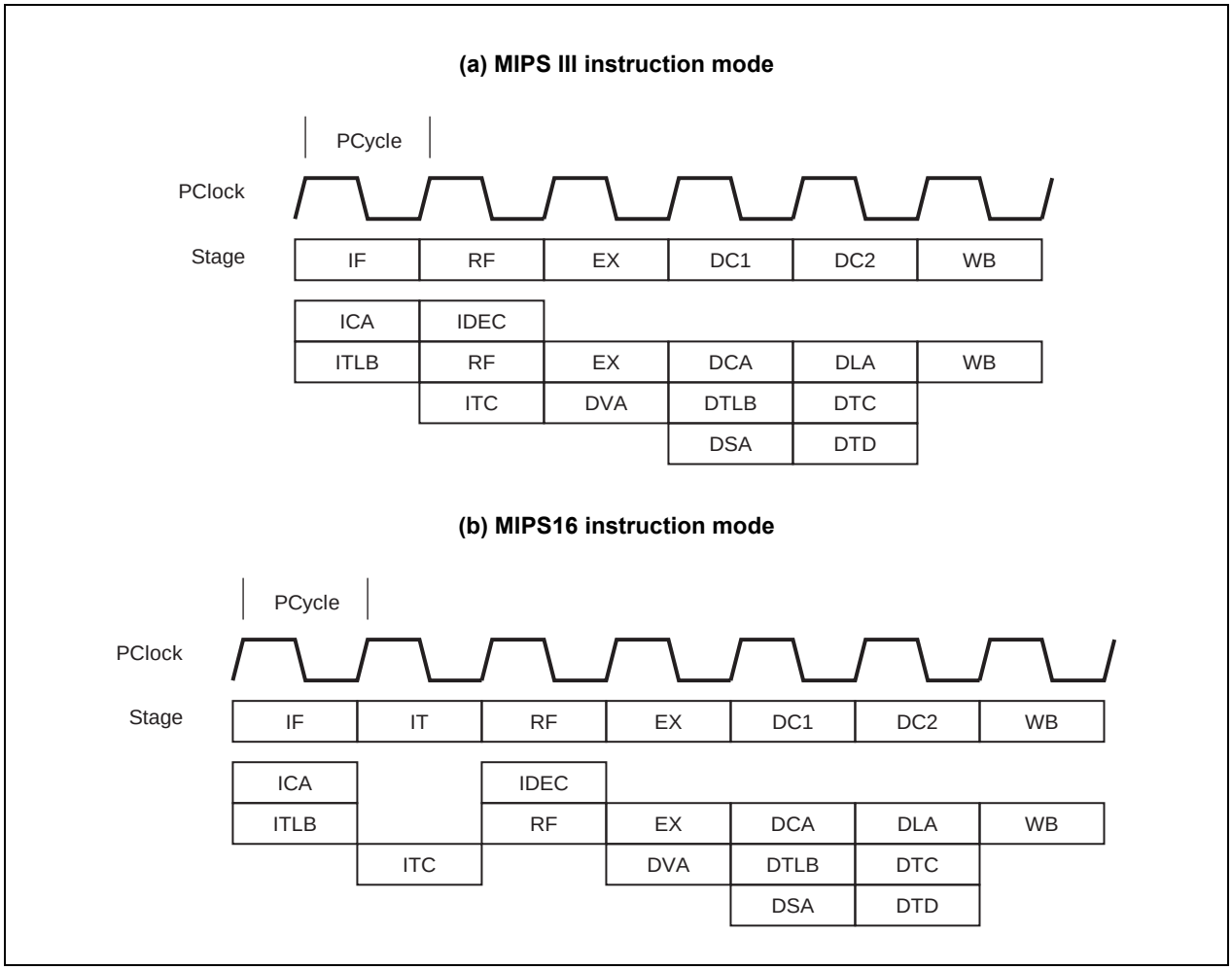
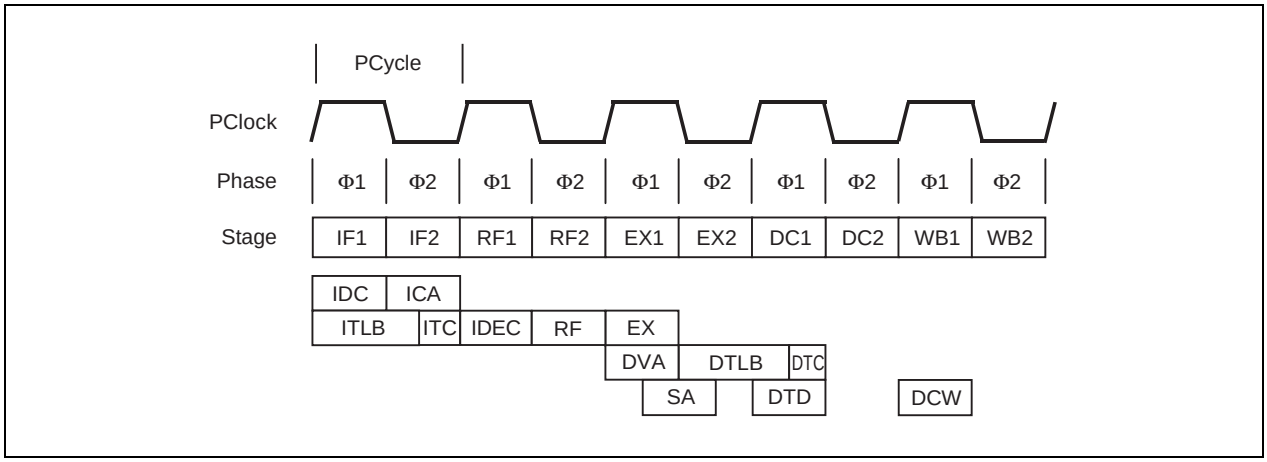


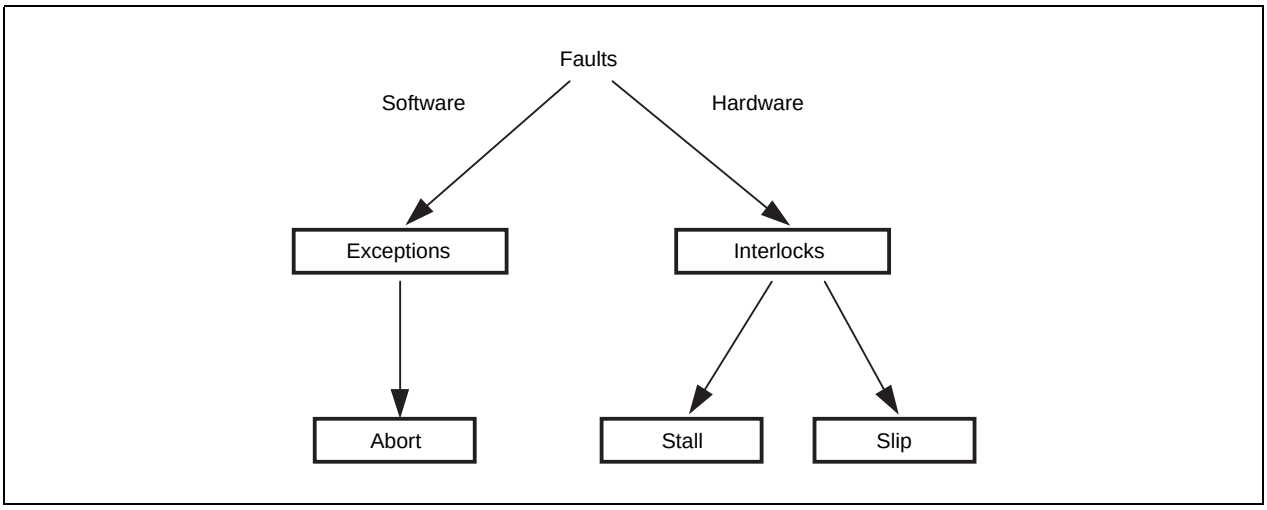
Figure 4-32. SW Instruction Pipeline Activities (VR4181)



4.7 Interlock and Exception

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. As shown in Figure 4-33, all interlock and exception conditions are collectively referred to as faults.

Figure 4-33. Interlocks, Exceptions, and Faults



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, as shown in Table 4-2. For instance, an LDI Interlock is raised in the Register Fetch (RF) stage.

Tables 4-3 and 4-4 describe the pipeline interlocks and exceptions listed in Table 4-2.

Table 4-2. Correspondence of Pipeline Stage to Interlock and Exception Conditions

Status \ Stage		IF	RF (IT)	EX	DC	WB
Interlock	Stall	–	ITM ICM	–	DTM DCM DCB	–
	Slip	–	LDI MDI SLI CP0	–	–	–
Exception		IAErr	NMI ITLB INTr IBE SYSC BP CUn RSVD	Trap OVF DAErr	Reset DTLB DTMod WAT DBE NMI (V_R4131) INTr (V_R4131)	–

Remark In the above table, exception conditions are listed up in higher priority order.

Table 4-3. Pipeline Interlock

Interlock	Description
ITM	Interrupt TLB Miss
ICM	Interrupt Cache Miss
LDI	Load Data Interlock
MDI	MD Busy Interlock
SLI	Store-Load Interlock
CP0	Coprocessor 0 Interlock
DTM	Data TLB Miss
DCM	Data Cache Miss
DCB	Data Cache Busy

Table 4-4. Description of Pipeline Exception

Exception	Description
IAErr	Instruction Address Error exception
NMI	Non-maskable Interrupt exception
ITLB	ITLB exception
INTr	Interrupt exception
IBE	Instruction Bus Error exception
SYSC	System Call exception
BP	Breakpoint exception
CUn	Coprocessor Unusable exception
RSVD	Reserved Instruction exception
Trap	Trap exception
OVF	Overflow exception
DAErr	Data Address Error exception
Reset	Reset exception
DTLB	DTLB exception
DTMod	DTLB Modified exception
WAT	Watch exception
DBE	Data Bus Error exception

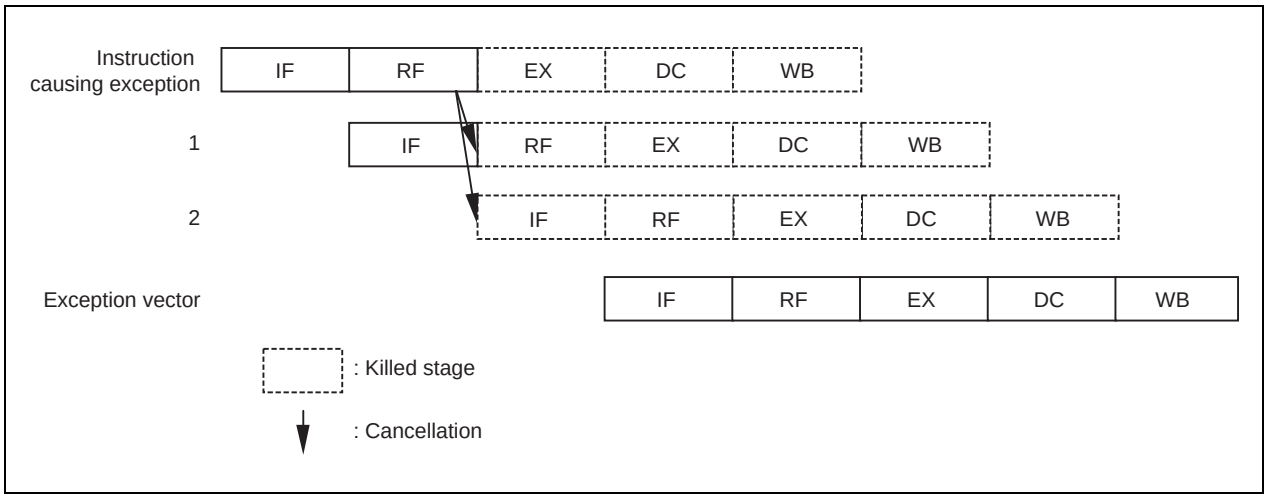
4.7.1 Exception conditions

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 an exceptional condition is detected for an instruction, the VR4100 Series will kill it and all following instructions. When this instruction reaches the WB stage, the exception flag and various information items are written to CP0 registers. The current PC is changed to the appropriate exception vector address and the exception bits of earlier pipeline stages are cleared.

This implementation allows all preceding instructions to complete execution and prevents all subsequent instructions from completing. Thus the value in the EPC is sufficient to restart execution. It also ensures that exceptions are taken in the order of execution; an instruction taking an exception may itself be killed by an instruction further down the pipeline that takes an exception in a later cycle.

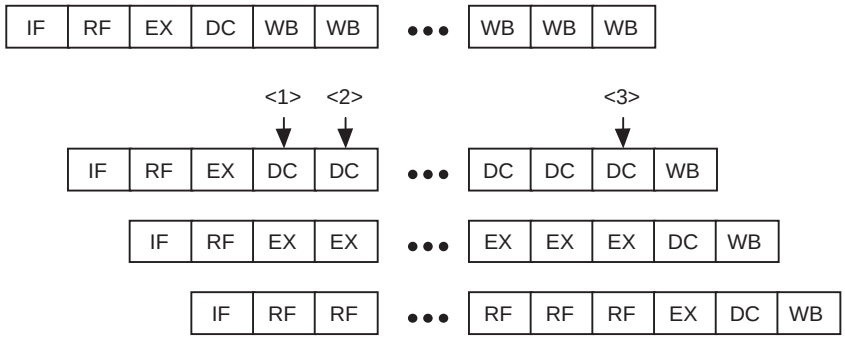
Figure 4-34. Exception Detection



4.7.2 Stall conditions

Stalls are used to stop the pipeline for conditions detected after the RF stage. When a stall occurs, the processor will resolve the condition and then the pipeline will continue. Figure 4-35 shows a data cache miss stall, and Figure 4-36 shows a CACHE instruction stall.

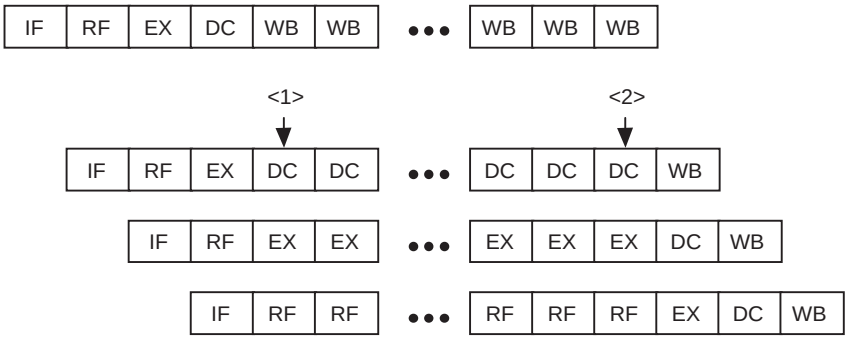
Figure 4-35. Data Cache Miss Stall



<1> Data cache miss
<2> Start moving data cache line to write buffer
<3> Get last word into cache and restart pipeline

If the cache line to be replaced is dirty — the W bit is set — the data is moved to the internal write buffer in the next cycle. The write-back data is returned to memory. The last word in the data is returned to the cache at <3>, and pipelining restarts.

Figure 4-36. CACHE Instruction Stall



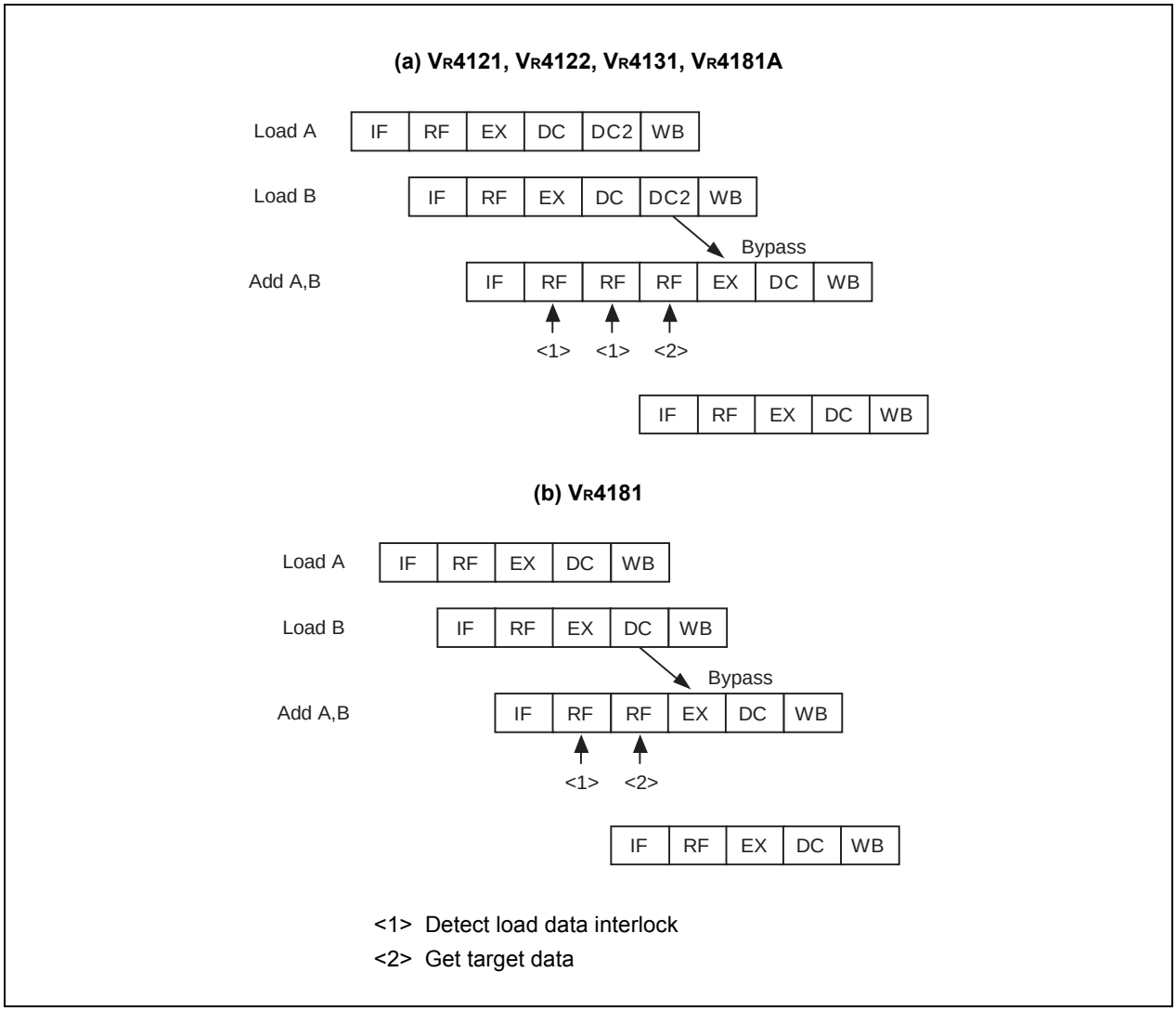
<1> CACHE instruction start
<2> CACHE instruction complete

When the CACHE instruction enters the DC pipe-stage, the pipeline stalls while the CACHE instruction is executed. The pipeline begins running again when the CACHE instruction is completed, allowing the instruction fetch to proceed.

4.7.3 Slip conditions

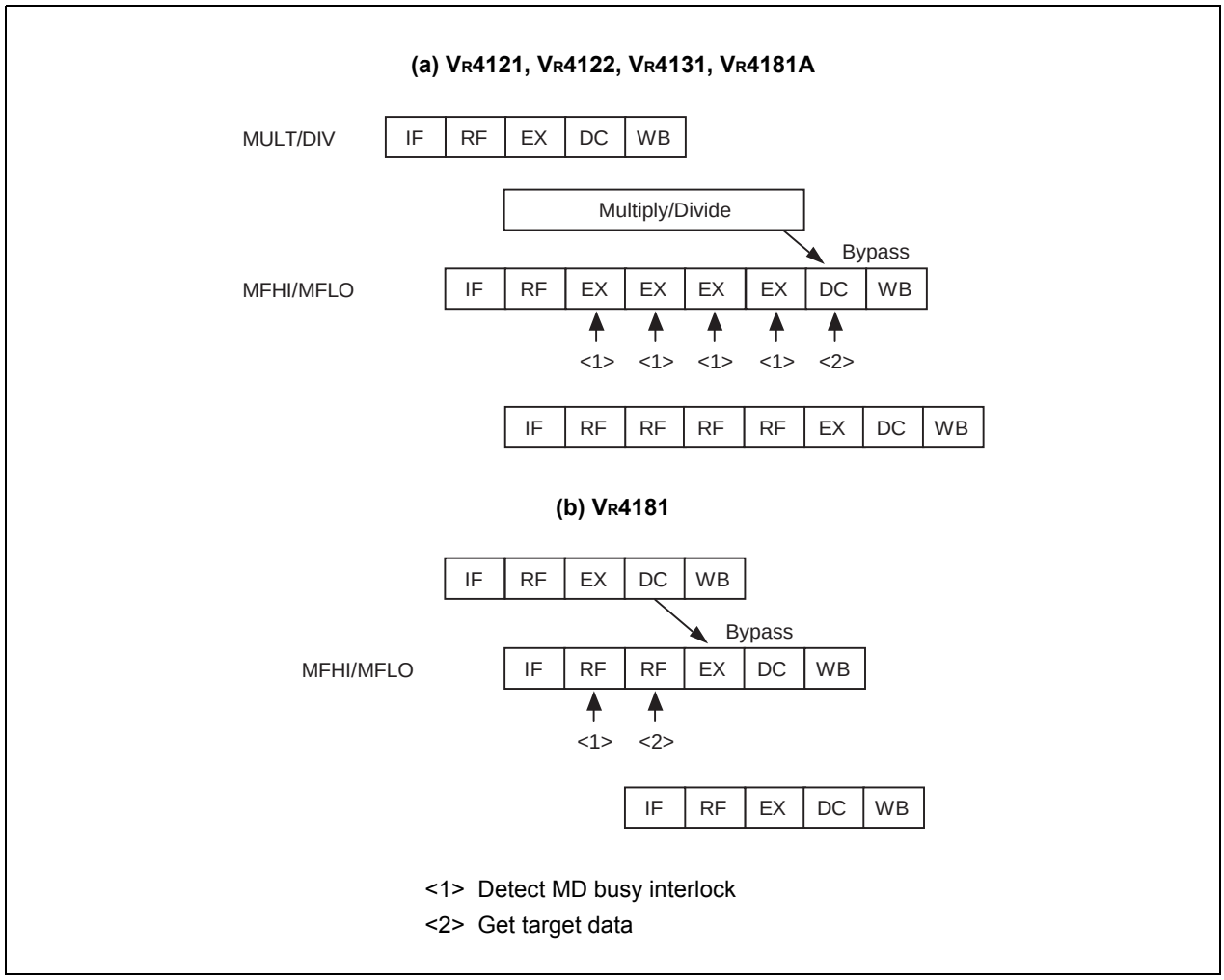
During the RF stage and the EX stage, internal logic will determine whether it is possible to start the current instruction in this cycle. If all of the source operands are available (either from the register file or via the internal bypass logic) and all the hardware resources necessary to complete the instruction will be available whenever required, then the instruction “run”; otherwise, the instruction will “slip”. Slipped instructions are retired on subsequent cycles until they issue. The backend of the pipeline (stages DC and WB) will advance normally during slips in an attempt to resolve the conflict. NOPs will be inserted into the bubble in the pipeline. Instructions killed by branch likely instructions, ERET or exceptions will not cause slips.

Figure 4-37. Load Data Interlock



Load Data Interlock is detected in the RF stage and also the pipeline slips in the stage. Load Data Interlock occurs when data fetched by a load instruction and data moved from HI, LO or CP0 register is required by the next immediate instruction. The pipeline begins running again at the clock after the target of the load is read from the data cache, HI, LO and CP0 registers. The data returned at the end of the DC stage is input into the end of the RF stage, using the bypass multiplexers.

Figure 4-38. MD Busy Interlock



MD Busy Interlock occurs when HI/LO register is required by MFHI/MFLO instruction before finishing Multiply/Divide execution. The pipeline begins running again at the clock after finishing Multiply/Divide execution.

In the VR4121, VR4122, VR4131, and VR4181A, MD Busy Interlock is detected in the EX stage and also the pipeline slips in the stage. The data returned from the HI/LO register at the end of the DC stage is input into the end of the EX stage, using the bypass multiplexer.

In the VR4181, MD Busy Interlock is detected in the RF stage and also the pipeline slips in the stage. The data returned from the HI/LO register at the end of the DC stage is input into the end of the RF stage, using the bypass multiplexer.

Store-Load Interlock is detected in the EX stage and the pipeline slips in the RF stage. Store-Load Interlock occurs when store instruction followed by load instruction is detected. The pipeline begins running again one clock later.

Coprocessor 0 Interlock is detected in the EX stage and the pipeline slips in the RF stage. Coprocessor Interlock occurs when an MTC0 instruction for the Config or Status register is detected. The pipeline begins running again one clock later.

4.7.4 Bypassing

In some cases, data and conditions produced in the EX, DC and WB stages of the pipeline are made available to the EX stage (only) through the bypass data path.

Operand bypass allows an instruction in the EX stage to continue without having to wait for data or conditions to be written to the register file at the end of the WB stage. Instead, the Bypass Control Unit is responsible for ensuring data and conditions from later pipeline stages are available at the appropriate time for instructions earlier in the pipeline.

The Bypass Control Unit is also responsible for controlling the source and destination register addresses supplied to the register file.

CHAPTER 5 MEMORY MANAGEMENT SYSTEM

The VR4100 Series provides a memory management unit (MMU) which uses a translation lookaside buffer (TLB) to translate virtual addresses into physical addresses. This chapter describes the virtual and physical address spaces, the virtual-to-physical address translation, the operation of the TLB in making these translations, and the CP0 registers that provide the software interface to the TLB.

5.1 Processor Modes

5.1.1 Operating mode

The processor has three operating modes, and accessible address spaces are determined by these modes.

- User mode
- Supervisor mode
- Kernel mode

User and Kernel modes are common to all VR-Series processors. Generally, Kernel mode is used to executing the operating system, while User mode is used to run application programs. The VR4000 Series ™ and later processors have a third mode, which is called Supervisor mode and categorized in between User and Kernel modes. This mode is used to configure a high-security system.

When an exception occurs, the CPU enters Kernel mode, and remains in this mode until an exception return instruction (ERET) is executed. The ERET instruction brings back the processor to the mode in which it was just before the exception occurs.

Access to the kernel address space is allowed when the processor is in Kernel mode.

Access to the supervisor address space is allowed when the processor is in Kernel or Supervisor mode.

Access to the user address space is allowed in any of the three operating modes.

5.1.2 Addressing mode

In the VR4100 Series, 32- or 64-bit mode is independently selectable for User, Supervisor, and Kernel operating modes. A processor in 64-bit mode translates 64-bit addresses and processes data in 64-bit unit.

5.2 Translation Lookaside Buffer (TLB)

Virtual addresses are translated into physical addresses using an on-chip TLB. The on-chip TLB is a fully-associative memory that holds 32 entries, which provide mapping to 32 odd/even page pairs for one entry. The pages can have five different sizes, 1 K, 4 K, 16 K, 64 K, and 256 K, and can be specified in each entry. If it is supplied with a virtual address, each of the 32 TLB entries is checked simultaneously to see whether they match the virtual addresses that are provided with the ASID field and saved in the EntryHi register.

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.

If no match occurs (TLB “miss”), an exception is taken and software refills the TLB from the page table resident in memory. The software writes to an entry selected using the Index register or a random entry indicated in the Random register.

If more than one entry in the TLB matches the virtual address being translated, TLB operations are not performed correctly. In the VR4181, the TLB-Shutdown (TS) bit of the Status register is set to 1, and the TLB becomes unusable (an attempt to access the TLB results in a TLB Refill exception regardless of whether there is an entry that hits). The TS bit can be cleared only by a reset. The VR4121, VR4122, VR4131, and VR4181A have no TS bit, and their operation is undefined if more than one entry in the TLB matches.

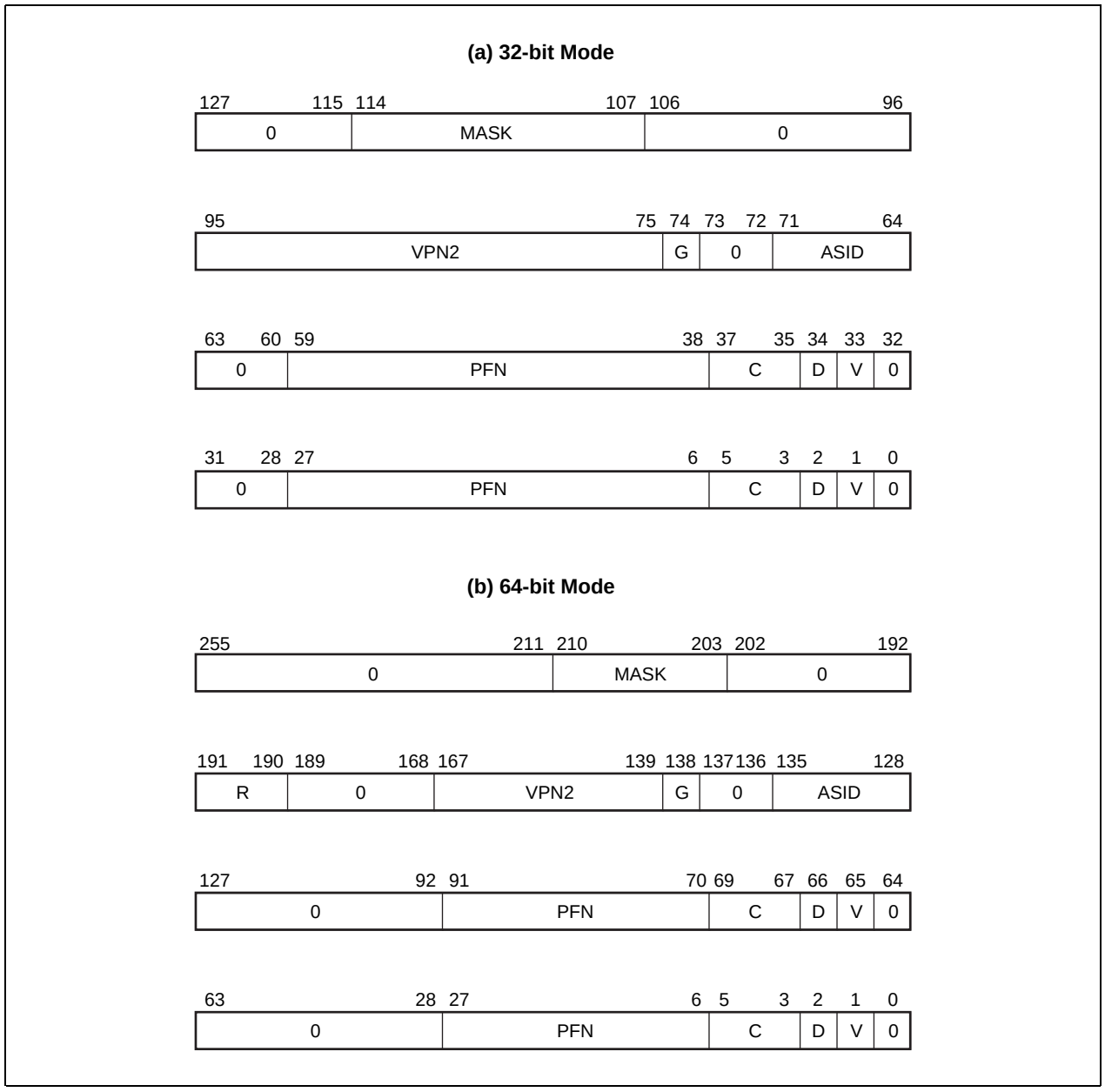
Note that virtual addresses may be converted to physical addresses without using a TLB, depending on the address space that is being subjected to address translation. For example, address translation for the kseg0 or kseg1 address space does not use mapping. The physical addresses of these address spaces are determined by subtracting the base address of the address space from the virtual addresses.

5.2.1 Format of a TLB entry

Each TLB entry has fields corresponding to the EntryHi, EntryLo0, EbtryLo1, and PageMask registers. The format of the EntryHi, EntryLo0, EbtryLo1, and PageMask registers are nearly the same as the TLB entry. However, the bit in the EntryHi register that corresponds to the TLB G bit is a reserved bit (0), and the bit in the TLB entry that corresponds to the G bit of the EntryLo register is reserved to 0. For details about other bits, refer to the descriptions of each register.

Figure 5-1 shows the TLB entry formats for both 32- and 64-bit modes.

Figure 5-1. Format of a TLB Entry

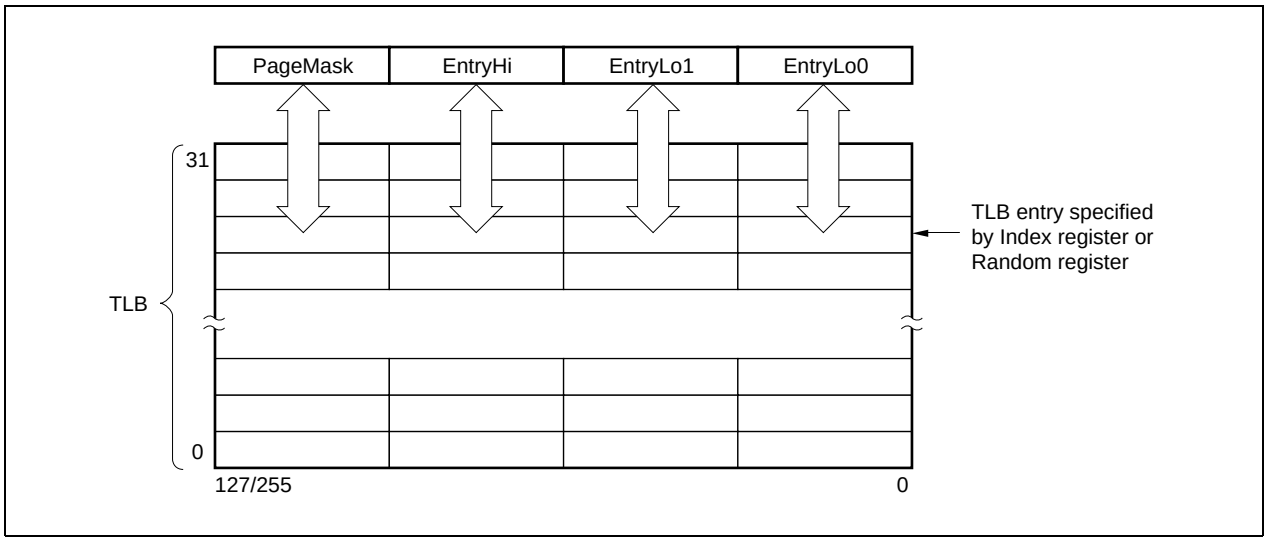


5.2.2 Manipulation of TLB

The contents of each TLB entry can be read or written through the EntryHi, EntryLo0, EbtryLo1, and PageMask registers with TLB manipulation instructions, as shown in Figure 5-2. An entry specified through the Index register or indicated in the Random register is used as a target.

The TLB must also be initialized and set after reset. Refer to **VR Series Programming Guide Application Note** for details about procedures and program examples of initialization.

Figure 5-2. TLB Manipulation Overview



5.2.3 TLB instructions

The instructions used for TLB control are described below. Refer to Chapter 9 for details about each instruction.

(1) Translation lookaside buffer probe (TLBP)

The translation lookaside buffer probe (TLBP) instruction loads the Index register with a TLB number that matches the content of the EntryHi register. If there is no TLB number that matches the TLB entry, the highest-order bit of the Index register is set.

(2) Translation lookaside buffer read (TLBR)

The translation lookaside buffer read (TLBR) instruction loads the EntryHi, EntryLo0, EntryLo1, and PageMask registers with the content of the TLB entry indicated by the content of the Index register.

(3) Translation lookaside buffer write index (TLBWI)

The translation lookaside buffer write index (TLBWI) instruction writes the contents of the EntryHi, EntryLo0, EntryLo1, and PageMask registers to the TLB entry indicated by the content of the Index register.

(4) Translation lookaside buffer write random (TLBWR)

The translation lookaside buffer write random (TLBWR) instruction writes the contents of the EntryHi, EntryLo0, EntryLo1, and PageMask registers to the TLB entry indicated by the content of the Random register.

5.2.4 TLB exceptions

If there is no TLB entry that matches the virtual address, a TLB Refill exception occurs. If the access control bits (D and V) indicate that the access is not valid, a TLB Modified or TLB Invalid exception occurs. If the C bit is 010, the retrieved physical address directly accesses main memory, bypassing the cache.

See Chapter 6 for details of the TLB Miss exception.

5.3 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 of all entries in the TLB. Either of the following comparisons is performed for the virtual page number (VPN):

- In 32-bit mode, the high-order bits[Note] of the 32-bit virtual address are compared to the contents of the VPN2 (virtual page number divided by two) of each TLB entry.
- In 64-bit mode, the high-order bits[Note] of the 64-bit virtual address are compared to the contents of the VPN2 (virtual page number divided by two) and R of each TLB entry.

Note The number of bits differs from page sizes. The table below shows the examples of high-order bits of the virtual address in page size of 256 KB and 1 KB.

Page size / Mode	256 KB	1 KB
32-bit mode	bits 31 to 19	bits 31 to 11
64-bit mode	bits 63, 62, 39 to 19	bits 63, 62, 39 to 11

It is a match when there is an entry whose VPN field is the same as that of virtual address, and either:

- the Global (G) bit of the TLB entry is set to 1, 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 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 to 1 for a valid address translation to take place, it is not involved in the determination of a matching TLB entry. The offset is concatenated to the retrieved physical address. An offset, which indicates an address within the page frame space, is the low-order bits of the virtual address and is output without passing through the TLB.

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

Figure 5-3 illustrates an outline of the address translation, and Figure 5-4 illustrates the TLB address translation flow.

Figure 5-3. Virtual-to-Physical Address Translation

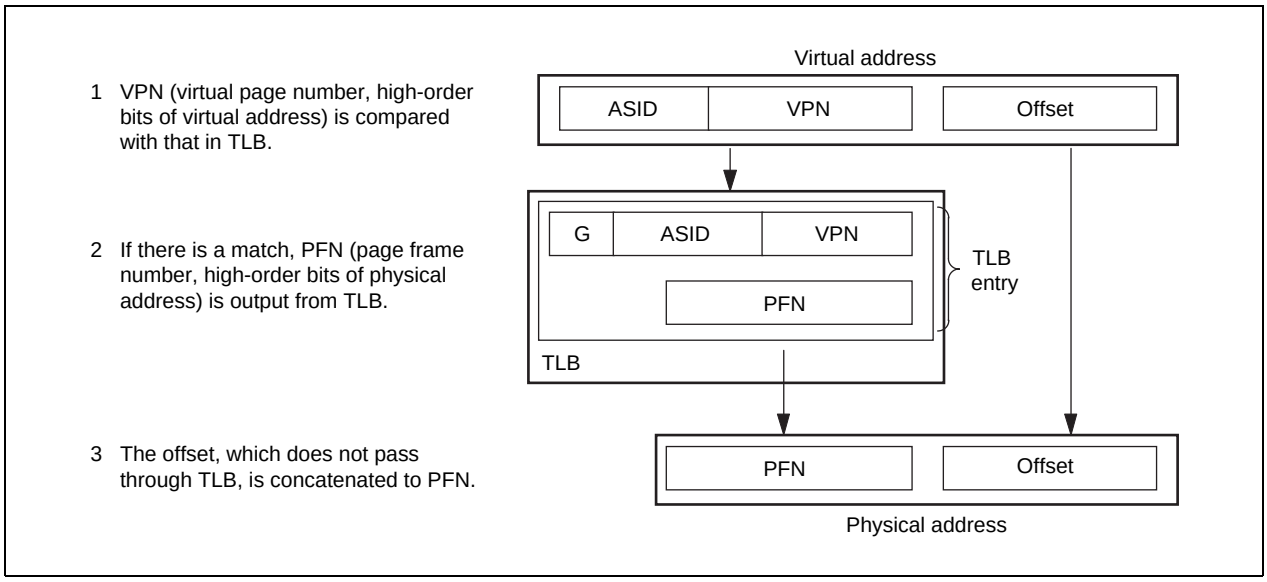
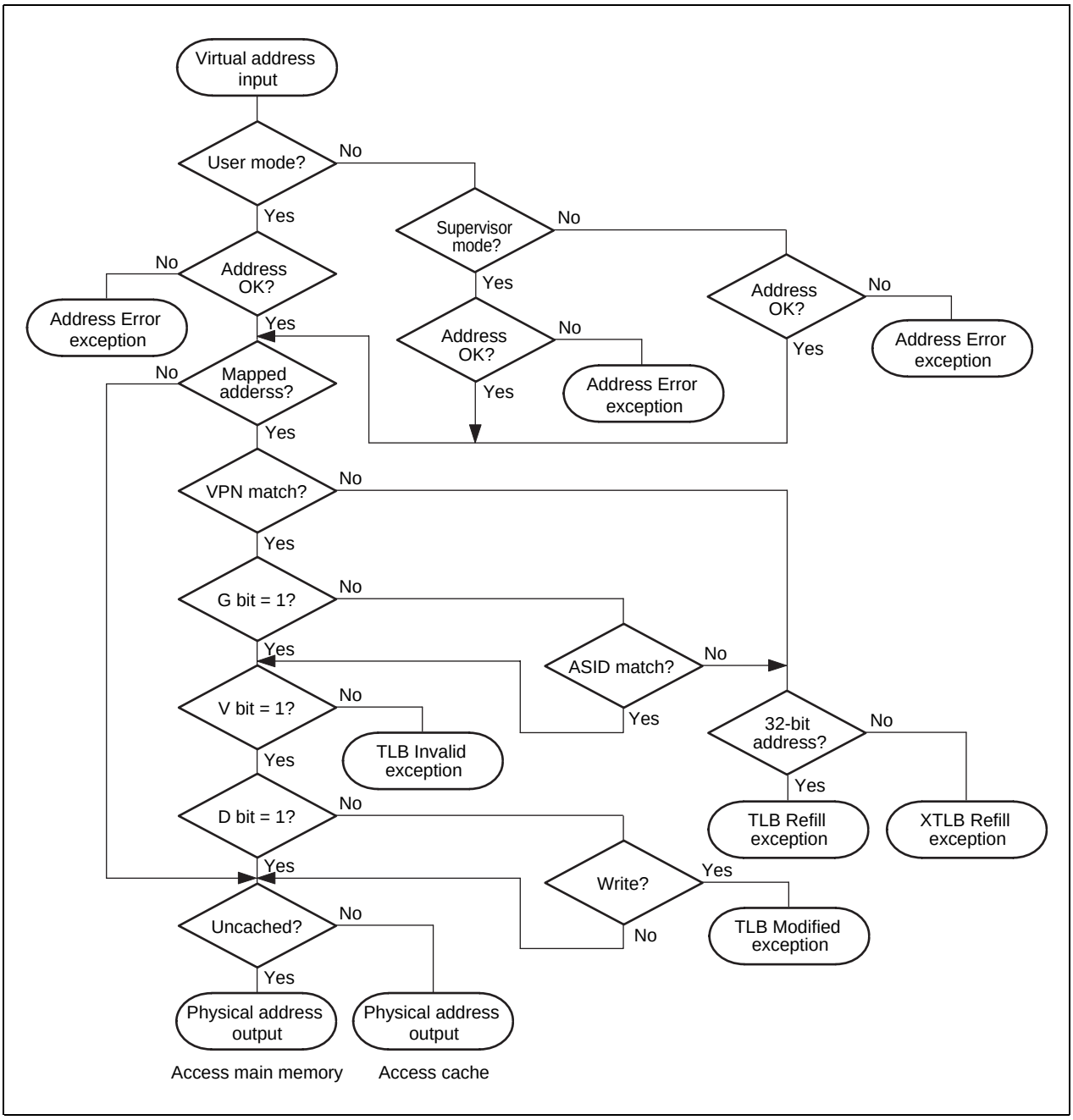


Figure 5-4. Address Translation in TLB

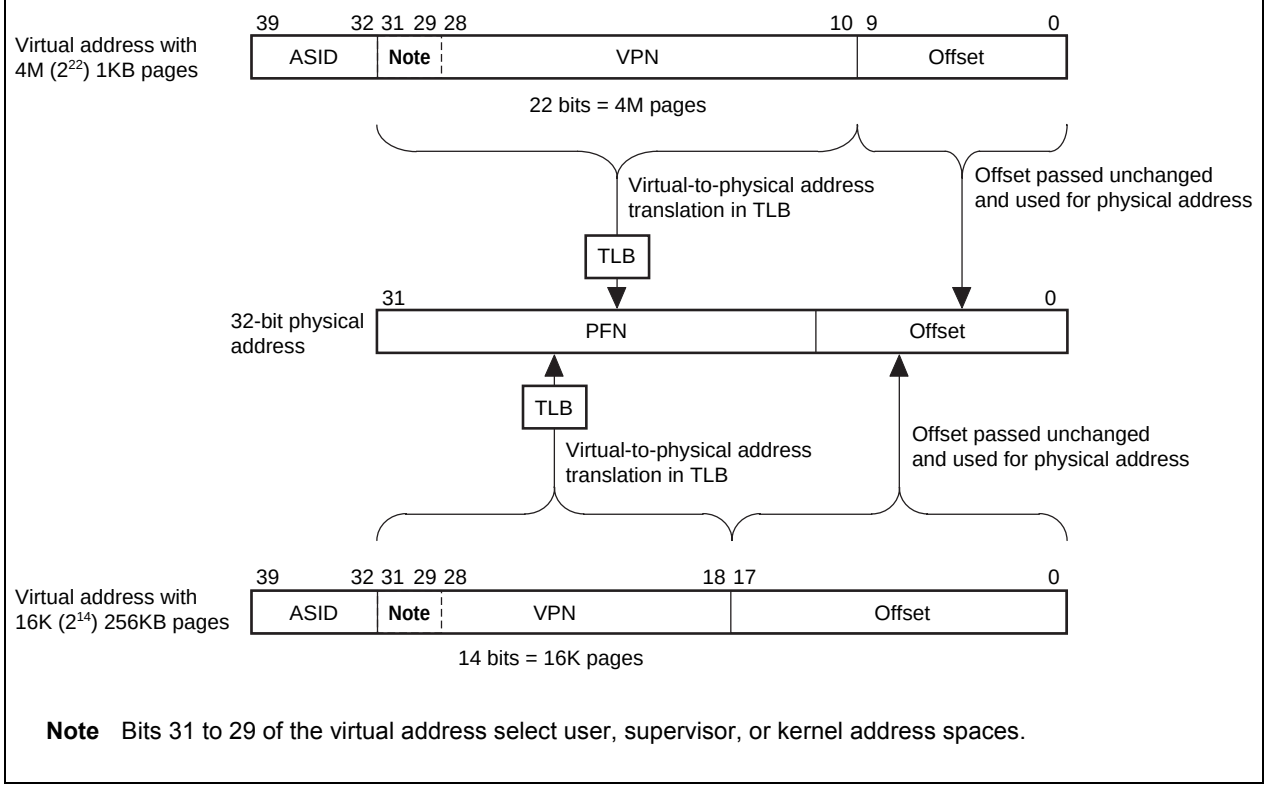


5.3.1 32-bit mode address translation

Figure 5-5 shows the virtual-to-physical-address translation of a 32-bit mode address. The pages can have five different sizes between 1 KB (10 bits) and 256 KB (18 bits), each being 4 times as large as the preceding one in ascending order, that is 1 K, 4 K, 16 K, 64 K, and 256 K. This figure illustrates the two possible page sizes: a 1 KB page (10 bits) and a 256 KB page (18 bits).

- Shown at the top of Figure 5-5 is the virtual address space in which the page size is 1 KB and the offset is 10 bits. The 22 bits excluding the ASID field represents the virtual page number (VPN), enabling selecting a page table of 4 M entries.
- Shown at the bottom of Figure 5-5 is the virtual address space in which the page size is 256 KB and the offset is 18 bits. The 14 bits excluding the ASID field represents the VPN, enabling selecting a page table of 16 K entries.

Figure 5-5. 32-bit Mode Virtual Address Translation



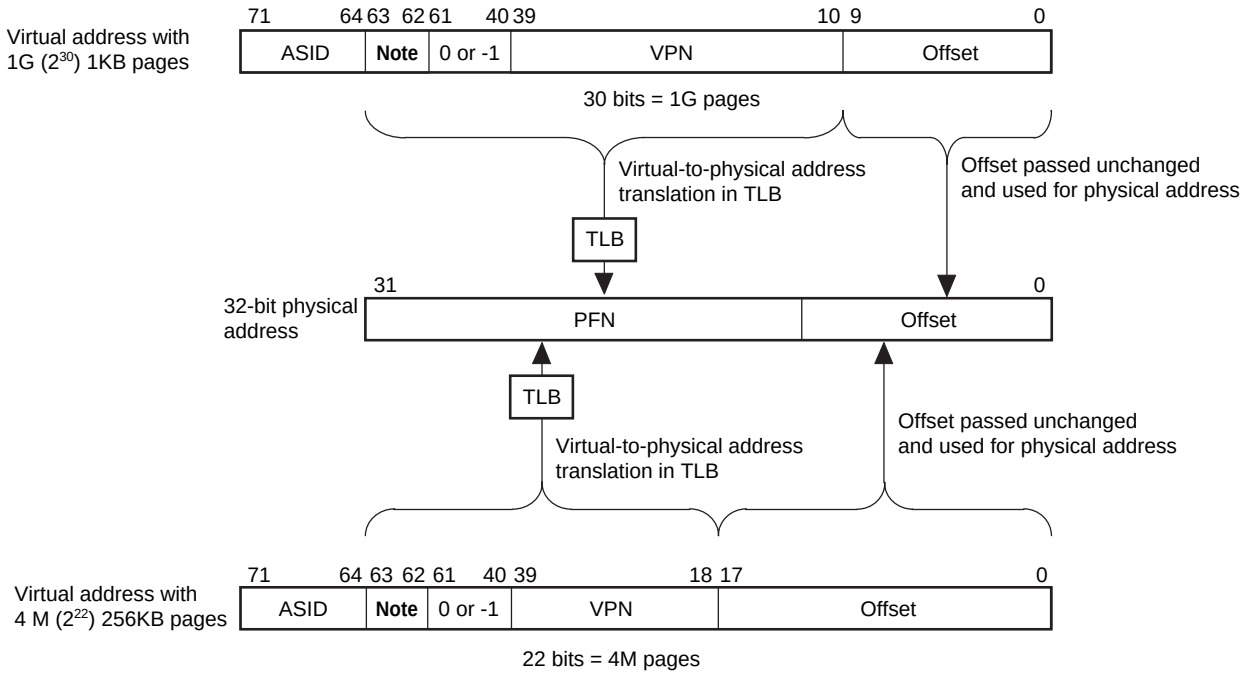
Note Bits 31 to 29 of the virtual address select user, supervisor, or kernel address spaces.

5.3.2 64-bit mode address translation

Figure 5-6 shows the virtual-to-physical-address translation of a 64-bit mode address. The pages can have five different sizes between 1 KB (10 bits) and 256 KB (18 bits), each being 4 times as large as the preceding one in ascending order, that is 1 K, 4 K, 16 K, 64 K, and 256 K. This figure illustrates the two possible page sizes: a 1 KB page (10 bits) and a 256 KB page (18 bits).

- Shown at the top of Figure 5-6 is the virtual address space in which the page size is 1 KB and the offset is 10 bits. The 30 bits excluding the ASID field represents the virtual page number (VPN), enabling selecting a page table of 1 G entry.
- Shown at the bottom of Figure 5-6 is the virtual address space in which the page size is 256 KB and the offset is 18 bits. The 22 bits excluding the ASID field represents the VPN, enabling selecting a page table of 4 M entries.

Figure 5-6. 64-bit Mode Virtual Address Translation



Note Bits 63 and 62 of the virtual address select user, supervisor, or kernel address spaces.

5.4 Address Spaces

The address space of the CPU is extended in memory management system, by converting (translating) huge virtual memory addresses into physical addresses.

The physical address space of the VR4100 Series is 4 GB and 32-bit width addresses are used.

For the virtual address space, up to 2 GB (2^{31} bytes) are provided as a user's area and 32-bit width addresses are used in the 32-bit mode. In the 64-bit mode, up to 1 TB (2^{40} bytes) is provided as a user's area and 64-bit width addresses are used. For the format of the TLB entry in each mode, refer to **5.2.1**.

As shown in Figures 5-5 and 5-6, the virtual address is extended with an address space identifier (ASID), which reduces the frequency of TLB flushing when switching contexts. This 8-bit ASID is in the CP0 EntryHi register, and the Global (G) bit is in the EntryLo0 and EntryLo1 registers, described later in this chapter.

5.4.1 User mode virtual address space

During User mode, a 2 GB (2^{31} bytes) virtual address space (useg) can be used in the 32-bit mode. In the 64-bit mode, a 1 TB (2^{40} bytes) virtual address space (xuseg) can be used.

As shown in Tables 5-5 and 5-6, each virtual address is extended independently as another virtual address by setting an 8-bit address space ID area (ASID), to support user processes of up to 256. The contents of TLB can be retained after context switching by allocating each process by ASID. useg and xuseg can be referenced via TLB. Whether a cache is used or not is determined for each page by the TLB entry (depending on the C bit setting in the TLB entry).

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:

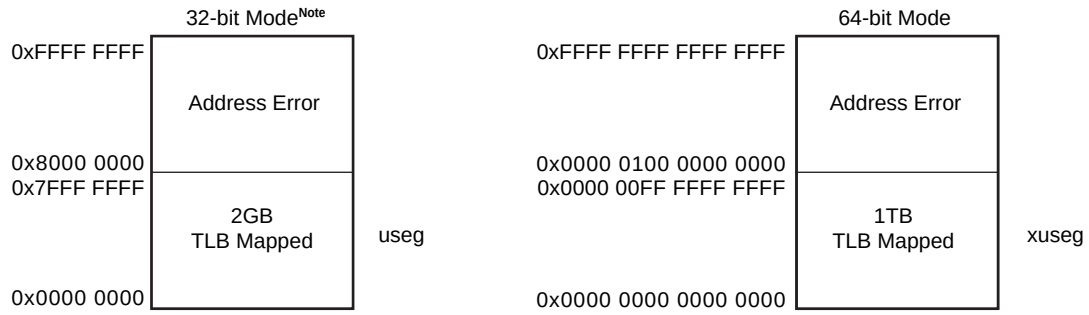
- KSU = 10
- EXL = 0
- ERL = 0

In conjunction with these bits, the UX bit in the Status register selects 32- or 64-bit User mode addressing as follows:

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

Figure 5-7 shows the address mapping for the User mode, and Table 5-1 lists the characteristics of each user segment (useg and xuseg).

Figure 5-7. User Mode Address Space



Note The VR4100 Series uses 64-bit addresses within it. When the processor is running in Kernel mode, it saves the contents of each register or restores their previous contents to initialize them before switching the context. For 32-bit mode addressing, bit 31 is sign-extended to bits 32 to 63, and the resulting 32 bits are used for addressing. Usually, it is impossible for 32-bit mode programs to generate invalid addresses. If context switching occurs and the processor enters Kernel mode, however, an attempt may be made to save an address other than the sign-extended 32-bit address mentioned above to a 64-bit register. In this case, user-mode programs are likely to generate an invalid address.

Table 5-1. User Mode Segments

Mode	Address bit value	Status register bit value				Segment name	Address range	Size
		KSU	EXL	ERL	UX			
32-bit	A31 = 0	10	0	0	0	useg	0x0000 0000 to 0x7FFF FFFF	2 GB (2^{31} bytes)
64-bit	A(63:40) = 0	10	0	0	1	xuseg	0x0000 0000 0000 0000 to 0x0000 00FF FFFF FFFF	1 TB (2^{40} bytes)

(1) useg (32-bit mode)

In User mode, when UX = 0 in the Status register and the most significant bit of the virtual address is 0, this virtual address space is labeled useg.

Any attempt to reference an address with the most-significant bit set while in User mode causes an Address Error exception (see **CHAPTER 6 EXCEPTION PROCESSING**).

The TLB Refill exception vector is used for TLB misses.

(2) xuseg (64-bit mode)

In User mode, when UX = 1 in the Status register and bits 63 to 40 of the virtual address are all 0, this virtual address space is labeled xuseg.

Any attempt to reference an address with bits 63:40 equal to 1 causes an Address Error exception (see **CHAPTER 6 EXCEPTION PROCESSING**).

The XTLB Refill exception vector is used for TLB misses.

5.4.2 Supervisor mode virtual address space

Supervisor mode 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.

All of the suseg, sseg, xsuseg, xsseg, and csseg spaces are referenced via TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

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

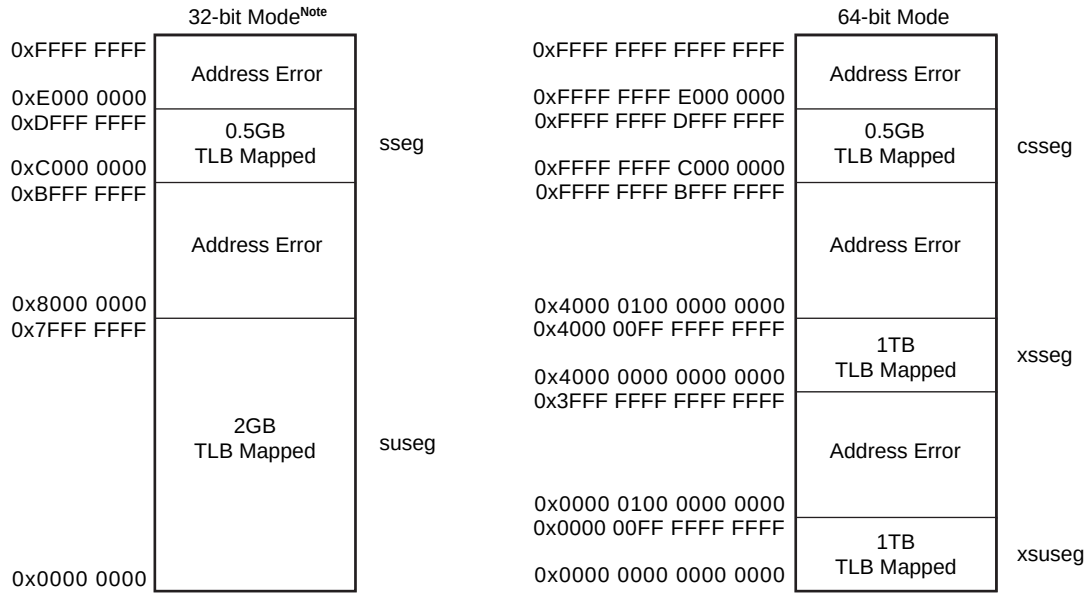
- KSU = 01
- EXL = 0
- ERL = 0

In conjunction with these bits, the SX bit in the Status register selects 32- or 64-bit Supervisor mode addressing as follows:

- When SX = 0, 32-bit supervisor space is selected.
- When SX = 1, 64-bit supervisor space is selected.

Figure 5-8 shows the supervisor mode address space, and Table 5-2 lists the characteristics of the Supervisor mode segments.

Figure 5-8. Supervisor Mode Address Space



Note The VR4100 Series uses 64-bit addresses within it. For 32-bit mode addressing, bit 31 is sign-extended to bits 32 to 63, and the resulting 32 bits are used for addressing. Usually, it is impossible for 32-bit mode programs to generate invalid addresses. In an operation of base register + offset for addressing, however, a two's complement overflow may occur, causing an invalid address. Note that the result becomes undefined. Two factors that can cause a two's complement follow:

- When offset bit 15 is 0, base register bit 31 is 0, and bit 31 of the operation "base register + offset" is 1
- When offset bit 15 is 1, base register bit 31 is 1, and bit 31 of the operation "base register + offset" is 0

Table 5-2. 32-bit and 64-bit Supervisor Mode Segments

Mode	Address bit value	Status register bit value				Segment name	Address range	Size
		KSU	EXL	ERL	SX			
32-bit	A31 = 0	01	0	0	0	suseg	0x0000 0000 to 0x7FFF FFFF	2 GB (2^{31} bytes)
32-bit	A(31:29) = 110	01	0	0	0	sseg	0xC000 0000 to 0xDFFF FFFF	512 MB (2^{29} bytes)
64-bit	A(63:62) = 00	01	0	0	1	xsuseg	0x0000 0000 0000 0000 to 0x0000 00FF FFFF FFFF	1 TB (2^{40} bytes)
64-bit	A(63:62) = 01	01	0	0	1	xsseg	0x4000 0000 0000 0000 to 0x4000 00FF FFFF FFFF	1 TB (2^{40} bytes)
64-bit	A(63:62) = 11	01	0	0	1	csseg	0xFFFF FFFF C000 0000 to 0xFFFF FFFF DFFF FFFF	512 MB (2^{29} bytes)

(1) suseg (32-bit Supervisor mode, user space)

When SX = 0 in the Status register and the most-significant bit of the virtual address space is set to 0, the suseg virtual address space is selected; it covers 2 GB (2^{31} bytes) 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) sseg (32-bit Supervisor mode, supervisor space)

When SX = 0 in the Status register and the three most-significant bits of the virtual address space are 110, the sseg virtual address space is selected; it covers 512 MB (2^{29} bytes) of the current supervisor virtual 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) xsuseg (64-bit Supervisor mode, user space)

When SX = 1 in the Status register and bits 63 and 62 of the virtual address space are set to 00, the xsuseg virtual address space is selected; it covers 1 TB (2^{40} bytes) 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) xsseg (64-bit Supervisor mode, current supervisor space)

When SX = 1 in the Status register and bits 63 and 62 of the virtual address space are set to 01, the xsseg virtual address space is selected; it covers 1 TB (2^{40} bytes) of the current supervisor virtual 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 0x4000 0000 0000 0000 and runs through 0x4000 00FF FFFF FFFF.

(5) csseg (64-bit Supervisor mode, separate supervisor space)

When SX = 1 in the Status register and bits 63 and 62 of the virtual address space are set to 11, the csseg virtual address space is selected; it covers 512 MB (2^{29} bytes) of the separate supervisor virtual 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 0xFFFF FFFF C000 0000 and runs through 0xFFFF FFFF DFFF FFFF.

5.4.3 Kernel mode virtual address space

If the Status register satisfies any of the following conditions, the processor runs in Kernel mode.

- KSU = 00
- EXL = 1
- ERL = 1

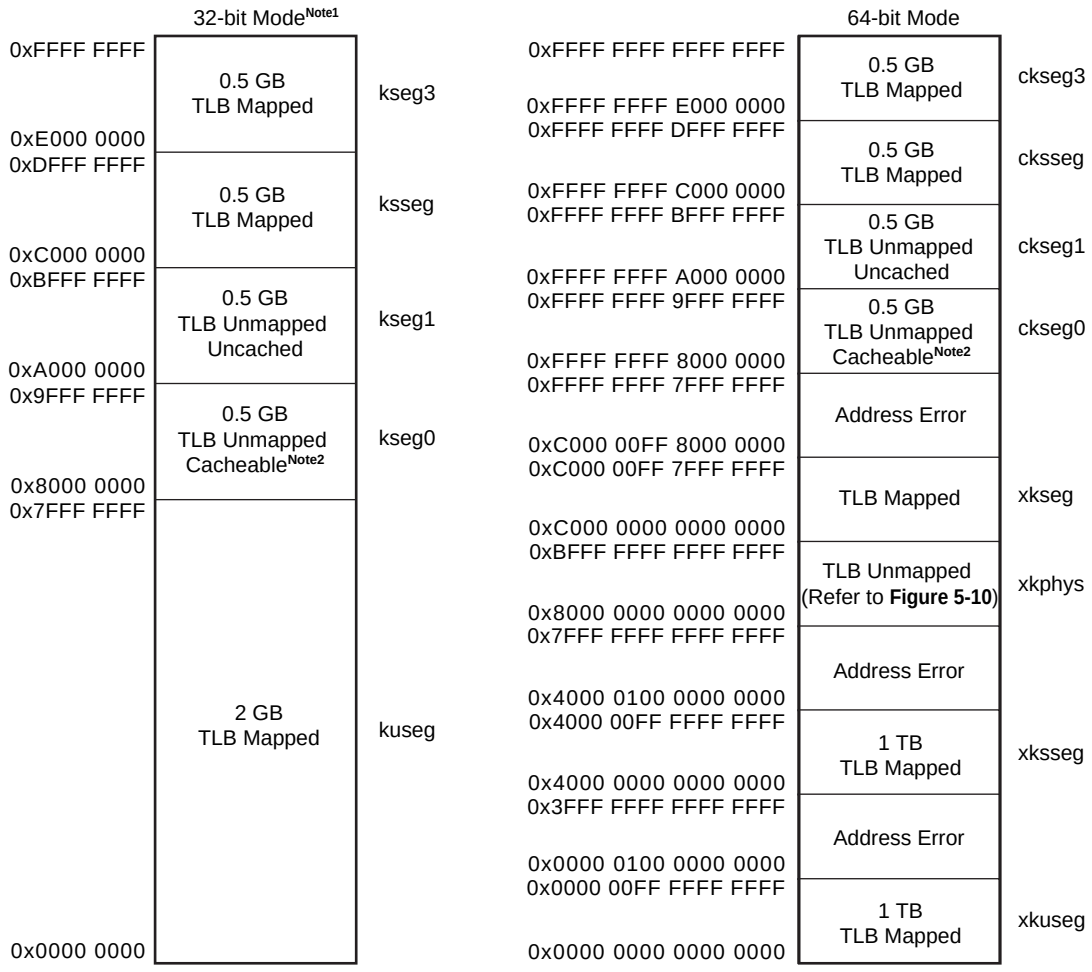
The addressing width in Kernel mode varies according to the state of the KX bit of the Status register, as follows:

- 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 in Kernel mode until an exception return (ERET) instruction is executed and results in ERL and/or EXL = 0. The ERET instruction restores the processor to the mode existing prior to the exception.

Kernel mode virtual address space is divided into regions differentiated by the high-order bits of the virtual address, as shown in Figure 5-9. Table 5-3 lists the characteristics of the 32-bit Kernel mode segments, and Table 5-4 lists the characteristics of the 64-bit Kernel mode segments.

Figure 5-9. Kernel Mode Address Space



Notes 1. The VR4100 Series uses 64-bit addresses within it. For 32-bit mode addressing, bit 31 is sign-extended to bits 32 to 63, and the resulting 32 bits are used for addressing. Usually, a 64-bit instruction is used for the program in 32-bit mode. In an operation of base register + offset for addressing, however, a two's complement overflow may occur, causing an invalid address. Note that the result becomes undefined. Two factors that can cause a two's complement follow:

- When offset bit 15 is 0, base register bit 31 is 0, and bit 31 of the operation "base register + offset" is 1
- When offset bit 15 is 1, base register bit 31 is 1, and bit 31 of the operation "base register + offset" is 0

2. The K0 field of the Config register controls cacheability of kseg0 and ckseg0.

Figure 5-10. xkphys Area Address Space

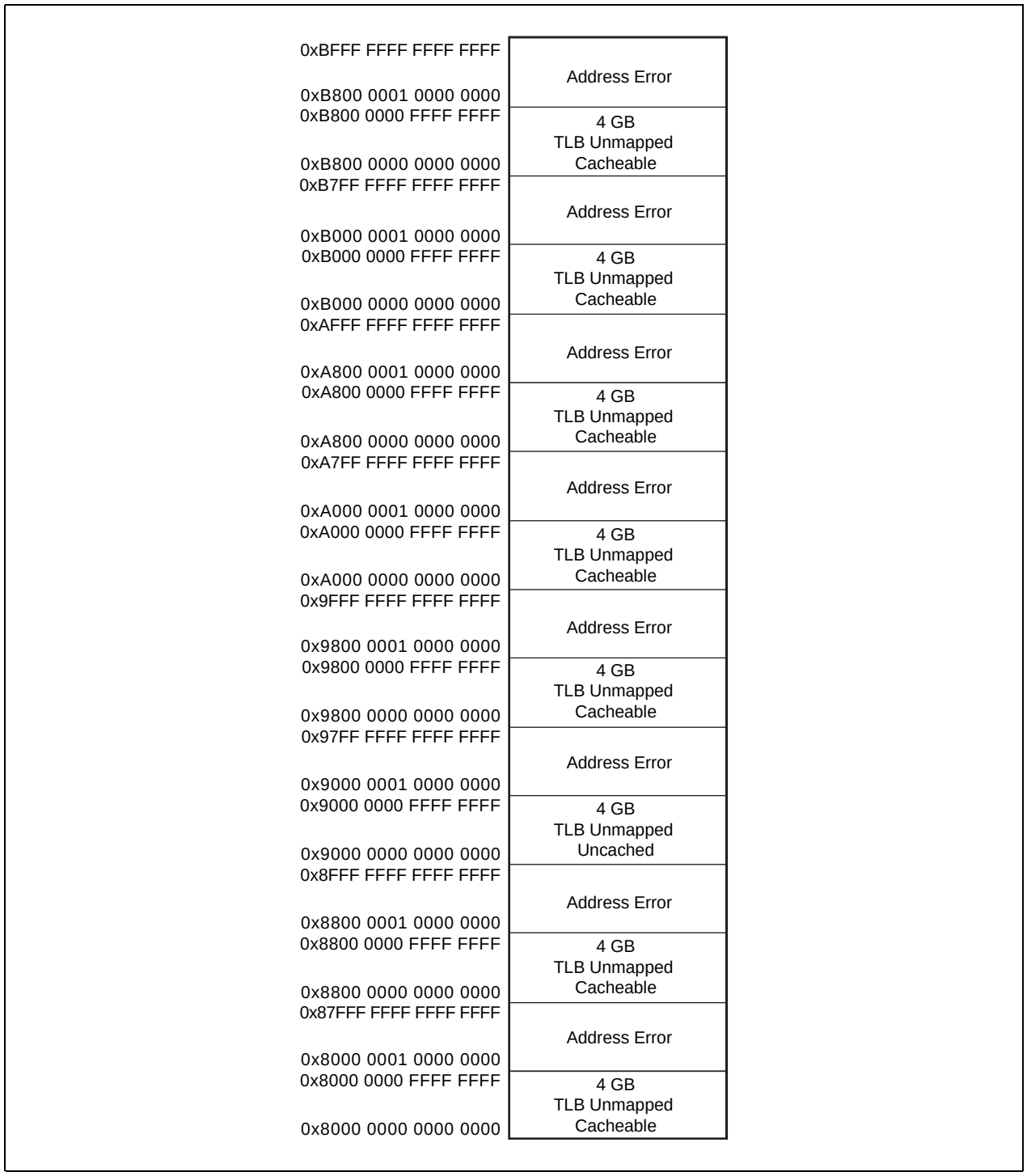


Table 5-3. 32-bit Kernel Mode Segments

Address bit value	Status register bit value				Segment name	Virtual address	Physical Address	Size
	KSU	EXL	ERL	KX				
A31 = 0	KSU = 00 or EXL = 1 or ERL = 1			0	kuseg	0x0000 0000 to 0x7FFF FFFF	TLB map	2 GB (2^{31} bytes)
A(31:29) = 100				0	kseg0	0x8000 0000 to 0x9FFF FFFF	0x0000 0000 to 0x1FFF FFFF	512 MB (2^{29} bytes)
A(31:29) = 101				0	kseg1	0xA000 0000 to 0xBFFF FFFF	0x0000 0000 to 0x1FFF FFFF	512 MB (2^{29} bytes)
A(31:29) = 110				0	ksseg	0xC000 0000 to 0xDFFF FFFF	TLB map	512 MB (2^{29} bytes)
A(31:29) = 111				0	kseg3	0xE000 0000 to 0xFFFF FFFF	TLB map	512 MB (2^{29} bytes)

(1) kuseg (32-bit Kernel mode, user space)

When KX = 0 in the Status register, and the most-significant bit of the virtual address space is 0, the kuseg virtual address space is selected; it is the current 2 GB (2^{31}-byte) user address space.

The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

References to kuseg are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

If the ERL bit of the Status register is 1, the user address space is assigned 2 GB (2^{31} bytes) without TLB mapping and becomes unmapped (with virtual addresses being used as physical addresses) and uncached so that the cache error handler can use it. This allows the Cache Error exception code to operate uncached using r0 as a base register.

(2) kseg0 (32-bit Kernel mode, kernel space 0)

When KX = 0 in the Status register and the most-significant three bits of the virtual address space are 100, the kseg0 virtual address space is selected; it is the current 512 MB (2^{29}-byte) physical space.

References to kseg0 are not mapped through TLB; the physical address selected is defined by subtracting 0x8000 0000 from the virtual address.

The K0 field of the Config register controls cacheability (refer to **5.5.8**).

(3) kseg1 (32-bit Kernel mode, kernel space 1)

When KX = 0 in the Status register and the most-significant three bits of the virtual address space are 101, the kseg1 virtual address space is selected; it is the current 512 MB (2^{29}-byte) physical space.

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

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

(4) ksseg (32-bit Kernel mode, supervisor space)

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

References to ksseg are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

(5) kseg3 (32-bit Kernel mode, kernel space 3)

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

References to kseg3 are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

Table 5-4. 64-bit Kernel Mode Segments

Address bit value	Status register bit value				Segment name	Virtual address	Physical address	Size
	KSU	EXL	ERL	KX				
A(63:62) = 00	KSU = 00 or EXL = 1 or ERL = 1			1	xkuseg	0x0000 0000 0000 0000 to 0x0000 00FF FFFF FFFF	TLB map	1 TB (2^{40} bytes)
A(63:62) = 01				1	xksseg	0x4000 0000 0000 0000 to 0x4000 00FF FFFF FFFF	TLB map	1 TB (2^{40} bytes)
A(63:62) = 10				1	xkphys	0x8000 0000 0000 0000 to 0xBFFF FFFF FFFF FFFF	0x0000 0000 to 0xFFFF FFFF	4 GB (2^{32} bytes)
A(63:62) = 11				1	xkseg	0xC000 0000 0000 0000 to 0xC000 00FF 7FFF FFFF	TLB map	2^{40} - 2^{31} bytes
A(63:62) = 11 A(63:31) = -1				1	ckseg0	0xFFFF FFFF 8000 0000 to 0xFFFF FFFF 9FFF FFFF	0x0000 0000 to 0x1FFF FFFF	512 MB (2^{29} bytes)
A(63:62) = 11 A(63:31) = -1				1	ckseg1	0xFFFF FFFF A000 0000 to 0xFFFF FFFF BFFF FFFF	0x0000 0000 to 0x1FFF FFFF	512 MB (2^{29} bytes)
A(63:62) = 11 A(63:31) = -1				1	cksseg	0xFFFF FFFF C000 0000 to 0xFFFF FFFF DFFF FFFF	TLB map	512 MB (2^{29} bytes)
A(63:62) = 11 A(63:31) = -1				1	ckseg3	0xFFFF FFFF E000 0000 to 0xFFFF FFFF FFFF FFFF	TLB map	512 MB (2^{29} bytes)

(6) xkuseg (64-bit Kernel mode, user space)

When KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 00, the xkuseg virtual address space is selected; it is the 1 TB (2^{40}-byte) current user address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

References to xkuseg are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

If the ERL bit of the Status register is 1, the user address space is assigned 2 GB (2^{31} bytes) without TLB mapping and becomes unmapped (with virtual addresses being used as physical addresses) and uncached so that the cache error handler can use it. This allows the Cache Error exception code to operate uncached using r0 as a base register.

(7) xksseg (64-bit Kernel mode, current supervisor space)

When KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 01, the xksseg address space is selected; it is the 1 TB (2^{40}-byte) current supervisor address space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

References to xksseg are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

(8) xkphys (64-bit Kernel mode, physical spaces)

When the KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 10, the virtual address space is called xkphys and selected as either cached or uncached. If any of bits 58 to 32 of the address is 1, an attempt to access that address results in an address error.

Whether cache can be used or not is determined by bits 59 to 61 of the virtual address. Table 5-5 shows cacheability corresponding to 8 address spaces.

Table 5-5. Cacheability and the xkphys Address Space

Bits 61 to 59	Cacheability	Address range
0	Cached	0x8000 0000 0000 0000 to 0x8000 0000 FFFF FFFF
1	Cached	0x8800 0000 0000 0000 to 0x8800 0000 FFFF FFFF
2	Uncached	0x9000 0000 0000 0000 to 0x9000 0000 FFFF FFFF
3	Cached	0x9800 0000 0000 0000 to 0x9800 0000 FFFF FFFF
4	Cached	0xA000 0000 0000 0000 to 0xA000 0000 FFFF FFFF
5	Cached	0xA800 0000 0000 0000 to 0xA800 0000 FFFF FFFF
6	Cached	0xB000 0000 0000 0000 to 0xB000 0000 FFFF FFFF
7	Cached	0xB800 0000 0000 0000 to 0xB800 0000 FFFF FFFF

(9) xkseg (64-bit Kernel mode, kernel spaces)

When the KX = 1 in the Status register and bits 63 and 62 of the virtual address space are 11, the virtual address space is called xkseg and selected as either 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
  References to xkseg are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.
- one of the four 32-bit kernel compatibility spaces, as described in the next section.

(10) 64-bit Kernel mode compatible spaces (ckseg0, ckseg1, cksseg, and ckseg3)

If the conditions listed below are satisfied in Kernel mode, ckseg0, ckseg1, cksseg, or ckseg3 (each having 512 Mbytes) is selected as a compatible space according to the state of the bits 30 and 29 (two low-order bits) of the address.

- The KX bit of the Status register is 1.
- Bits 63 and 62 of the 64-bit virtual address are 11.
- Bits 61 to 31 of the virtual address are all 1.

(a) ckseg0

This space is an unmapped region, compatible with the 32-bit mode kseg0 space. The K0 field of the Config register controls cacheability and coherency (refer to **5.5.8**).

(b) ckseg1

This space is an unmapped and uncached region, compatible with the 32-bit mode kseg1 space.

(c) cksseg

This space is the current supervisor virtual space, compatible with the 32-bit mode ksseg space.
References to cksseg are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

(d) ckseg3

This space is the current supervisor virtual space, compatible with the 32-bit mode kseg3 space.
References to ckseg3 are mapped through TLB. Whether cache can be used or not is determined by bit C of each page's TLB entry.

5.5 Memory Management Registers

This section describes the CP0 registers that are accessed by the memory management system and software. Table 5-6 lists the CP0 registers. About the exception processing registers of the CP0 registers, refer to **CHAPTER 6 EXCEPTION PROCESSING**.

Table 5-6 CP0 Registers

(a) Memory Management Registers

Register name	Register number
Index register	0
Random register	1
EntryLo0 register	2
EntryLo1 register	3
PageMask register	5
Wired register	6
EntryHi register	10
PRId register	15
Config register	16
LLAddr register Note1	17
TagLo register	28
TagHi register	29
–	–

(b) Exception Processing Registers

Register name	Register number
Context register	4
BadVAddr register	8
Count register	9
Compare register	11
Status register	12
Cause register	13
EPC register	14
WatchLo register	18
WatchHi register	19
XContext register	20
Parity Error register Note2	26
Cache Error register Note2	27
ErrorEPC register	30

Notes 1. This register is defined to maintain compatibility with the VR4000 and VR4400. The content of this register is meaningless in the normal operation.

2. This register is defined to maintain compatibility with the VR4100. This register is not used in the normal operation.

Details about each register are explained below. The parenthesized number in section titles is the register number (refer to **1.2.3**).

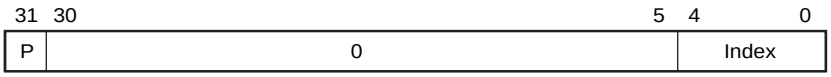
5.5.1 Index register (0)

The Index register is a 32-bit, read/write register containing five low-order bits to index an entry in the TLB. The most-significant 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.

The contents of the Index register after reset are undefined so that it must be initialized by software.

Figure 5-11. Index Register



P : Indicates whether probing is successful or not. It is set to 1 if the latest TLBP instruction fails. It is cleared to 0 when the TLBP instruction is successful.

Index : Specifies an index to a TLB entry that is a target of the TLBR or TLBWI instruction.

0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

5.5.2 Random register (1)

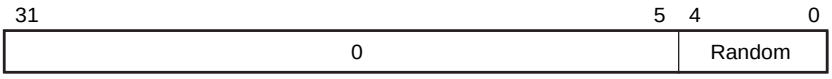
The Random register is a read-only register. The low-order 5 bits are used in referencing a TLB entry. This register is decremented each time an instruction is executed. The values that can be set in the register are as follows:

- The lower bound is the content of the Wired register.
- The upper bound is 31.

The Random register specifies the entry in the TLB that is affected by the TLBWR instruction. The register is readable to verify proper operation of the processor.

The Random register is set to the value of the upper bound upon Cold Reset. This register is also set to the upper bound when the Wired register is written. Figure 5-12 shows the format of the Random register.

Figure 5-12. Random Register



Random : TLB random index

0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

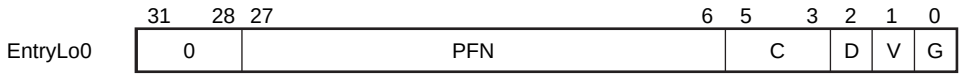
5.5.3 EntryLo0 (2) and EntryLo1 (3) registers

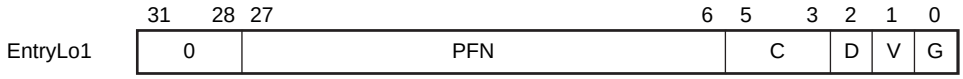
The EntryLo register consists of two registers that have identical formats: EntryLo0, used for even virtual pages and EntryLo1, used for odd virtual pages. The EntryLo0 and EntryLo1 registers are both read-/write-accessible. They are used to access the built-in TLB. When a TLB read/write operation is carried out, the EntryLo0 and EntryLo1 registers hold the contents of the low-order 32 bits of TLB entries at even and odd addresses, respectively.

The contents of these registers after reset are undefined so that they must be initialized by software.

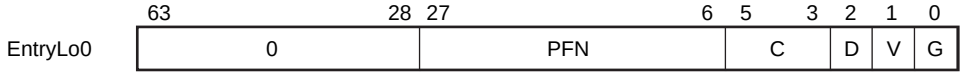
Figure 5-13. EntryLo0 and EntryLo1 Registers

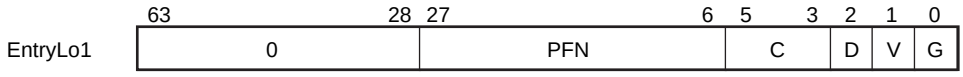
(a) 32-bit Mode





(b) 64-bit Mode





PFN : Page frame number; high-order bits of the physical address.

C : Specifies the TLB page attribute (see **Table 5-7**).

D : Dirty. If this bit is set to 1, 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 to 1, it indicates that the TLB entry is valid; otherwise, a TLB Invalid exception (TLBL or TLBS) occurs.

G : Global. If this bit is set in both EntryLo0 and EntryLo1, then the processor ignores the ASID during TLB lookup.

0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

The coherency attribute (C) bits are used to specify whether to use the cache in referencing a page. When the cache is used, whether the page attribute is “cached” or “uncached” is selected by algorithm.

Table 5-7 lists the page attributes selected according to the value in the C bits.

Table 5-7. Cache Algorithm

C bit value	Cache algorithm
0	Cached
1	Cached
2	Uncached
3	Cached
4	Cached
5	Cached
6	Cached
7	Cached

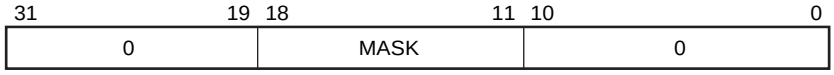
5.5.4 PageMask register (5)

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 page size for each TLB entry, as shown in Table 5-8. Page sizes must be from 1 KB to 256 KB.

TLB read and write instructions use this register as either a source or a destination; Bits 18 to 11 that are targets of comparison are masked during address translation.

The contents of the PageMask register after reset are undefined so that it must be initialized by software.

Figure 5-14. PageMask Register



MASK : Page comparison mask, which determines the virtual page size for the corresponding entry.
0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

Table 5-8 lists the mask pattern for each page size. If the mask pattern is one not listed below, the TLB behaves unexpectedly.

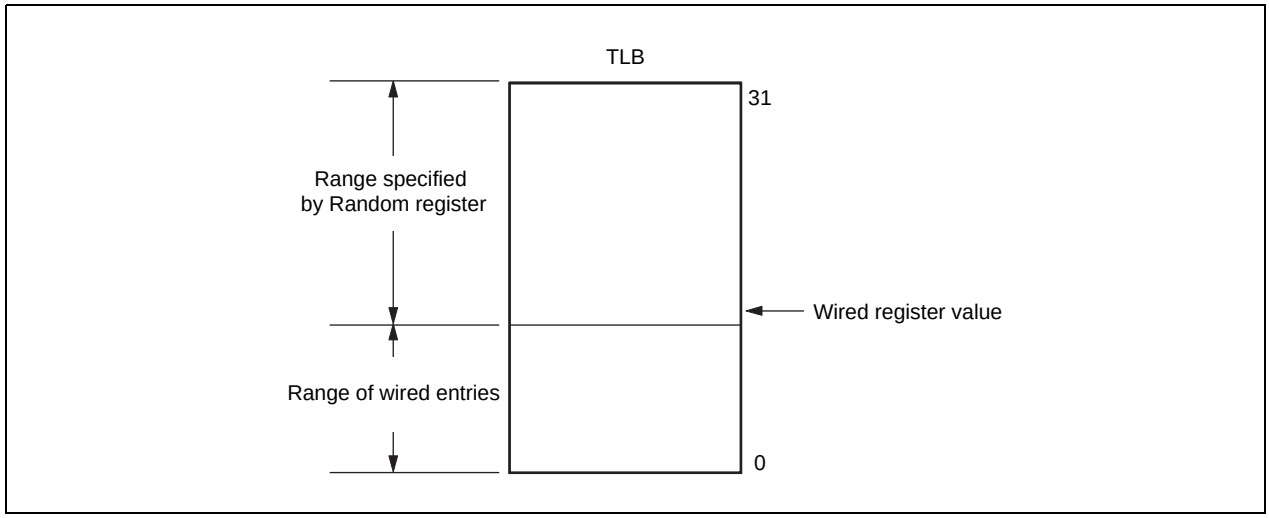
Table 5-8. Mask Values and Page Sizes

Page size	Bit							
	18	17	16	15	14	13	12	11
1 KB	0	0	0	0	0	0	0	0
4 KB	0	0	0	0	0	0	1	1
16 KB	0	0	0	0	1	1	1	1
64 KB	0	0	1	1	1	1	1	1
256 KB	1	1	1	1	1	1	1	1

5.5.5 Wired register (6)

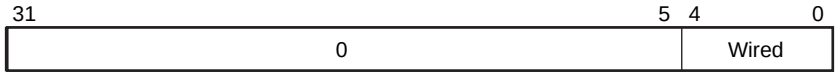
The Wired register is a read/write register that specifies the lower boundary of the random entry of the TLB as shown in Figure 5-15. Wired entries cannot be overwritten by a TLBWR instruction. They can, however, be overwritten by a TLBWI instruction. Random entries can be overwritten by both instructions.

Figure 5-15. Positions Indicated by the Wired Register



The Wired register is set to 0 upon Cold Reset. Writing this register also sets the Random register to the value of its upper bound (see **5.5.2 Random register (1)**). Figure 5-16 shows the format of the Wired register.

Figure 5-16. Wired Register



Wired : TLB wired boundary

0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

5.5.6 EntryHi register (10)

The EntryHi register is write-accessible. It is used to access the built-in TLB. The EntryHi register holds the high-order bits of a TLB entry for TLB read and write operations. If a TLB Refill, TLB Invalid, or TLB Modified exception occurs, the EntryHi register holds the high-order bit of the TLB entry. The EntryHi register is also set with the virtual page number (VPN2) for a virtual address where an exception occurred and the ASID. See Chapter 6 for details of the TLB exception.

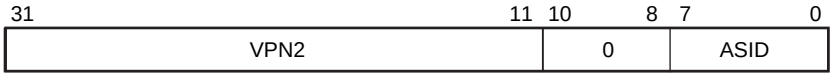
The ASID is used to read from or write to the ASID field of the TLB entry. It is also checked with the ASID of the TLB entry as the ASID of the virtual address during address translation.

The EntryHi register is accessed by the TLBP, TLBWR, TLBWI, and TLBR instructions.

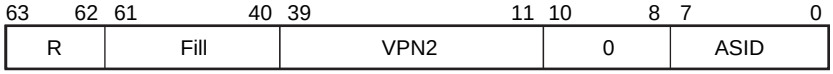
The contents of the EntryHi register after reset are undefined so that it must be initialized by software.

Figure 5-17. EntryHi Register

(a) 32-bit Mode



(b) 64-bit Mode



VPN2 : Virtual page number divided by two (mapping to two pages)

ASID : Address space ID. An 8-bit ASID field that lets multiple processes share the TLB; each process has a distinct mapping of otherwise identical virtual page numbers.

R : Space type (00 → user, 01 → supervisor, 11 → kernel). Matches bits 63 and 62 of the virtual address.

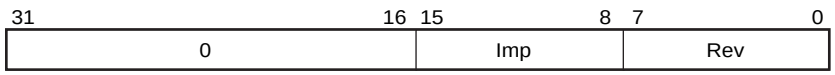
Fill : Reserved. Ignored on write. When read, returns zero.

0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

5.5.7 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 5-18 shows the format of the PRId register.

Figure 5-18. PRId Register



Imp : CPU core processor ID number (0x0C for the VR4100 Series)
Rev : CPU core processor revision number
0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

The processor revision number is stored as a value in the form y.x, where y is a major revision number in bits 7 to 4 and x is a minor revision number in bits 3 to 0.

The processor revision number identifies the revision of a CPU core. The major revision number (bits 7 to 4) identifies the VR4100 Series processors as follows:

Processor	Rev field
VR4121	0110xxxx
VR4122	0111xxxx (xxxx may be 0010 or less)
VR4131	1000xxxx
VR4181	0101xxxx
VR4181A	0111xxxx (xxxx may be 0011 or greater)

The minor revision number (bits 3 to 0) may be different even though the same processor names.

There is no guarantee that changes to the CPU core will necessarily be reflected in the PRId register, or changes to the revision number necessarily reflect real CPU core changes. Therefore, create a program that does not depend on the processor revision number field.

5.5.8 Config register (16)

The Config register specifies various configuration options selected on VR4100 Series processors.

Some configuration options, as defined by the EC, M16, and BE fields, are set by the hardware during Cold Reset and are included in the Config register as read-only status bits for the software to access. Other configuration options are read/write (AD, EP, and K0 fields) and controlled by software; on Cold Reset these fields are undefined. Since only a subset of the VR4000 Series options are available in the VR4100 Series, some bits are set to constants (e.g., bits 14 to 13) that were variable in the VR4000 Series. The Config register should be initialized by software before caches are used. Figure 5-19 shows the format of the Config register.

The contents of writable fields except for IS and BP bits in the Config register after reset are undefined so that they must be initialized by software.

Figure 5-19. Config Register (1/2)

(a) VR4121, VR4181

31	30–28	27–24	23	22–21	20	19–18	17	16	15	14–13	12	11–9	8–6	5–3	2–0
0	EC	EP	AD	0	M16	0	1	0	BE	10	CS	IC	DC	0	K0

(b) VR4122

31	30–28	27–24	23	22–21	20	19–18	17	16	15	14–13	12	11–9	8–6	5	4–3	2–0
IS	EC	EP	AD	0	M16	0	1	BP	BE	10	CS	IC	DC	IB	0	K0

(c) VR4131, VR4181A

31	30–28	27–24	23	22–21	20	19–18	17	16	15	14–13	12	11–9	8–6	5	4	3	2–0
IS	EC	EP	AD	0	M16	0	1	BP	BE	10	CS	IC	DC	IB	DB	0	K0

IS : Instruction streaming function (VR4122, VR4131, VR4181A only)
- 0 → ON (default value)
- 1 → OFF

EC : System clock ratio (see **Table 5-9**)

EP : Transfer data pattern (cache write-back pattern) setting
- 0 → DD: 1 word/1 cycle
- Others → Reserved

AD : Accelerate data mode
- 0 → VR4000 Series compatible mode
- 1 → Reserved

M16 : MIPS16 ISA mode enable/disable indication (read only)
- 0 → MIPS16 instruction cannot be executed
- 1 → MIPS16 instruction can be executed

BE : Endian mode of memory and a kernel.
- 0 → Little endian
- 1 → Big endian (VR4131 only)

Figure 5-19. Config Register (2/2)

CS : Cache size mode indication (n = IC, DC). Fixed to 1 in the VR4100 Series.
0 → Reserved
1 → $2^{(n+10)}$ bytes

IC : Instruction cache size indication. $2^{(IC+10)}$ bytes in the VR4100 Series (see **Table 5-10**).

DC : Data cache size indication. $2^{(DC+10)}$ bytes in the VR4100 Series (see **Table 5-11**).

IB : Instruction cache refill size setting (VR4122, VR4131, and VR4181A only, and fixed to 1 in the VR4181A).
0 → 4 words (16 bytes)
1 → 8 words (32 bytes)

DB : Data cache refill size setting (VR4131 and VR4181A only, and fixed to 1 in the VR4181A).
0 → 4 words (16 bytes)
1 → 8 words (32 bytes)

K0 : kseg0 cache coherency algorithm
010 → Uncached
Others → Cached

1 : 1 is returned when read.

0 : 0 is returned when read.

Caution Be sure to set the EP field and the AD bit to 0. If they are set with any other values, the processor may behave unexpectedly.

(1) Instruction streaming function (VR4122, VR4131, and VR4181A only)

Instruction streaming can shorten the period during which the pipeline is stalled. Usually, the pipeline is stalled until the cache line is refilled if an instruction cache miss occurs. With the VR4122, VR4131, and VR4181A, however, the stalled pipeline is resumed, even if refilling is not completed, as soon as the instruction to be fetched has been read from the external memory.

(2) Indication of clock frequency ratio

The EC area indicates the ratio of the internal peripheral function operating clock frequency to the pipeline clock (PClock) frequency. The frequency ratio to be indicated differs depending on the processor, as follows.

Table 5-9 System Interface Clock Ratio (to PClock)

EC field	VR4121	VR4122	VR4131	VR4181	VR4181A
0	1/1.5	Reserved		1/2	Reserved
1	1/2			1/3	1/2
2	1/2.5	Reserved		1/4	Reserved
3	1/3			Reserved	1/3
4	1/4			Reserved	1/4
5	1/5			Reserved	1/5
6	1/6			Reserved	1/6
7	1/1	Reserved			1/1

(3) Branch prediction function (VR4122, VR4131, and VR4181A only)

Usually, a branch delay of at least 1 clock occurs in order to check the branch condition and calculate the branch destination address when a branch instruction is fetched. The VR4122, VR4131, and VR4181A can reduce the occurrence of this delay using branch prediction.

The VR4122, VR4131, and VR4181A have a branch prediction table to which branch instructions whose branch conditions have been satisfied and their branch destination addresses are registered. When the next branch instruction is fetched, this branch prediction table is referenced. If the same branch instruction is in the table (hit), an instruction is fetched from the branch destination address in the table. This branch prediction is performed and branch instructions can be executed without delay if the BP bit is cleared to 0.

(4) Indication of cache size

The IC and DC fields indicate the respective capacities of the instruction cache and data cache. Because the capacities of the caches differ depending on the processor, these fields are fixed to the value corresponding to the processor.

Table 5-10 Instruction Cache Sizes

Processor	Size	IC field
VR4121	16 KB	4
VR4122	32 KB	5
VR4131	16 KB	4
VR4181	4 KB	2
VR4181A	8 KB	3

Table 5-11 Data Cache Sizes

Processor	Size	DC field
VR4121	8 KB	3
VR4122	16 KB	4
VR4131	16 KB	4
VR4181	4 KB	2
VR4181A	8 KB	3

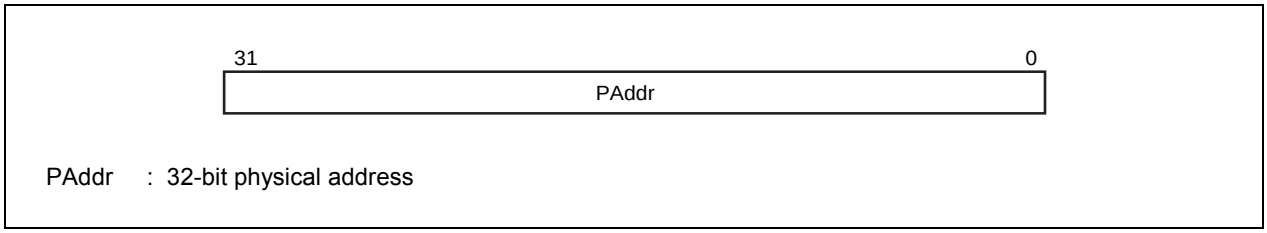
5.5.9 Load Linked Address (LLAddr) register (17)

The read/write Load Linked Address (LLAddr) register is not used with the VR4100 Series processor except for diagnostic purpose, and serves no function during normal operation.

LLAddr register is implemented just for compatibility between the VR4100 Series and VR4000/VR4400.

The contents of the LLAddr register after reset are undefined.

Figure 5-20. LLAddr Register



5.5.10 TagLo (28) and TagHi (29) registers

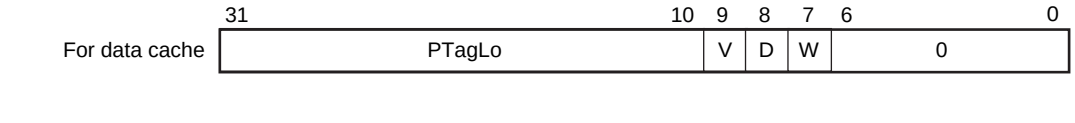
The TagLo and TagHi registers are 32-bit read/write registers that hold the primary cache tag during cache initialization, cache diagnostics, or cache error processing. The TagLo and TagHi registers are written by the CACHE and MTC0 instructions.

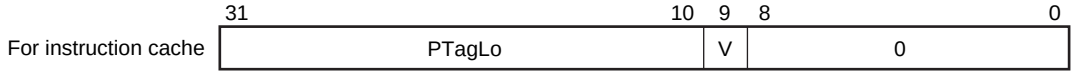
Figures 5-21 and 5-22 show the format of these registers.

The contents of these registers after reset are undefined.

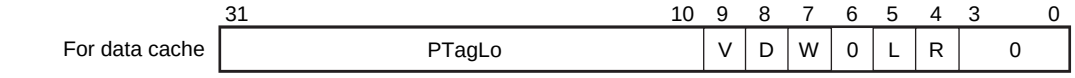
Figure 5-21. TagLo Register

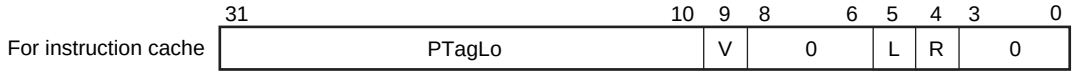
(a) VR4121, VR4122, VR4181, VR4181A





(b) VR4131





PTagLo : Specifies physical address bits 31 to 10.

V : Valid bit

D : Dirty bit. However, this bit is defined only for the compatibility with the VR4000 Series processors, and does not indicate the status of cache memory in spite of its readability and writability. This bit cannot change the status of cache memory. In the VR4131, a write to this bit is ignored and the same value as the V bit is read on read.

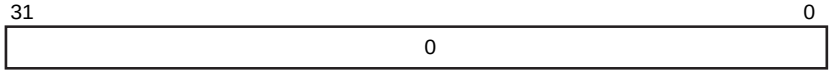
W : Write-back bit (set if cache line has been updated)

L : Lock bit. If this bit is set, the cache line is not refilled on cache misses.

R : LRU bit. Indicates the way to be refilled on cache misses.

0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

Figure 5-22. TagHi Register



0 : Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

CHAPTER 6 EXCEPTION PROCESSING

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

6.1 Exception Processing Overview

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 an exception, the normal sequence of instruction execution is suspended and the processor enters Kernel mode (see Chapter 5 for a description of system operating modes). If an exception occurs while executing a MIPS16 instruction, the processor stops the MIPS16 instruction execution, and shifts to the 32-bit instruction execution mode. The processor then disables interrupts and transfers control for execution to the exception handler (located at a specific address as an exception handling routine implemented by software). The handler saves the context of the processor, including the contents of the program counter, the current operating mode (User or Supervisor), statuses, and interrupt enabling. 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. Note that no branch delay slot generated by executing a branch instruction exists when the processor operates in the MIPS16 mode.

When MIPS16 instructions are enabled to be executed, bit 0 of the EPC register indicates the operating mode in which an exception occurred. It indicates 1 when in the MIPS16 instruction mode, and indicates 0 when in the MIPS III instruction mode.

The VR4100 Series processors have registers other than above that retain address, cause, or status information during exception processing. Details about these registers are described in **6.2 Exception Processing Registers**. For detailed descriptions about exception processing, refer to **6.4 Details of Exceptions**.

6.1.1 Precision of exceptions

VR4100 Series exceptions are logically precise; the instruction that causes an exception and all those that follow it are aborted and can be re-executed after servicing the exception. When succeeding instructions are killed, exceptions associated with those instructions are also killed. Exceptions are not taken in the order detected, but in instruction fetch order.

The exception handler can still determine exception and its origin. The cause of the program can be restarted by rewriting the destination register - not automatically, however, as in the case of all the other precise exceptions where no status change occurs.

6.2 Exception Processing Registers

This section describes the CP0 registers that are used in exception processing. Table 6-1 lists the CP0 registers. About the memory management registers of the CP0 registers, refer to **CHAPTER 5 MEMORY MANAGEMENT SYSTEM**.

Table 6-1. CP0 Registers

(a) Exception Processing Registers

Register name	Register number
Context register	4
BadVAddr register	8
Count register	9
Compare register	11
Status register	12
Cause register	13
EPC register	14
WatchLo register	18
WatchHi register	19
XContext register	20
Parity Error register Note1	26
Cache Error register Note1	27
ErrorEPC register	30

(b) Memory Management Registers

Register name	Register number
Index register	0
Random register	1
EntryLo0 register	2
EntryLo1 register	3
PageMask register	5
Wired register	6
EntryHi register	10
PRId register	15
Config register	16
LLAddr register Note2	17
TagLo register	28
TagHi register	29
–	–

Notes 1. This register is defined to maintain compatibility with the VR4100. This register is not used in the normal operation.

2. This register is defined to maintain compatibility with the VR4000 and VR4400. The content of this register is meaningless in the normal operation.

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.

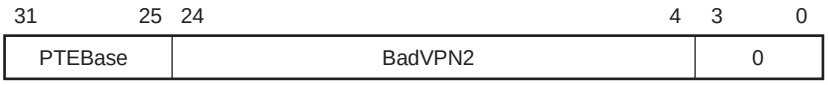
Details about each register are explained below. The parenthesized number in section titles is the register number (refer to **1.2.3**).

6.2.1 Context register (4)

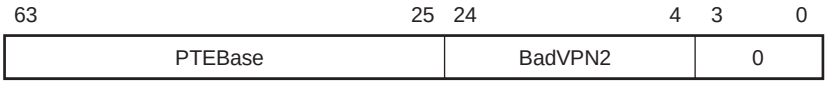
The Context register is a read/write register containing the pointer to an entry in the page table entry (PTE) array on the memory; this array is a table that stores virtual-to-physical address translations. When there is a TLB miss, the operating system loads the unsuccessfully translated entry from the PTE array to the TLB. The Context register is used by the TLB Refill exception handler for loading TLB entries. 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 6-1 shows the format of the Context register.

Figure 6-1. Context Register

(a) 32-bit Mode



(b) 64-bit Mode



PTEBase: The PTEBase field is a base address of the PTE entry table.

BadVPN2: The BadVPN2 field is written by hardware if a TLB miss occurs. This field holds the value (VPN2) obtained by halving the virtual page number of the most recent virtual address for which translation failed.

0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

The PTEBase field is used by software as the pointer to the base address of the PTE table in the current user address space.

The 21-bit BadVPN2 field contains bits 31 to 11 of the virtual address that caused the TLB miss; bit 10 is excluded because a single TLB entry maps to an even-odd page pair. For a 1 KB page size, this format can directly address the pair-table of 8-byte PTEs. When the page size is 4 KB or more, shifting or masking this value produces the correct PTE reference address.

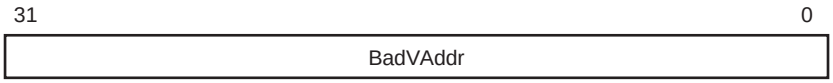
6.2.2 BadVAddr register (8)

The Bad Virtual Address (BadVAddr) register is a read-only register that saves the most recent virtual address that failed to have a valid translation, or that had an addressing error. Figure 6-2 shows the format of the BadVAddr register.

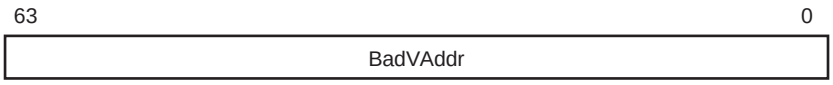
Caution This register saves no information after a bus error exception, because it is not an address error exception.

Figure 6-2. BadVAddr Register

(a) 32-bit Mode



(b) 64-bit Mode



BadVAddr: Most recent virtual address for which an addressing error occurred, or for which address translation failed.

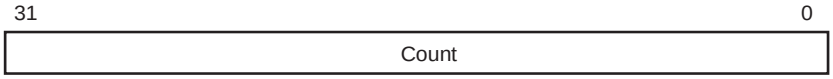
6.2.3 Count register (9)

The read/write Count register acts as a timer. It is incremented in synchronization with the MasterOut clock (internal clock), regardless of whether instructions are being executed, retired, or any forward progress is actually made through the pipeline.

This register is a free-running type. When the register reaches all ones, it rolls over to zero and continues counting. This register is used for self-diagnostic test, system initialization, or the establishment of inter-process synchronization.

Figure 6-3 shows the format of the Count register.

Figure 6-3. Count Register



Count: 32-bit up-date count value that is compared with the value of the Compare register.

6.2.4 Compare register (11)

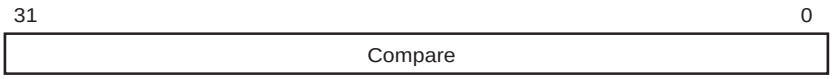
The Compare register causes a timer interrupt; it maintains a stable value that does not change on its own.

When the value of the Count register (see **6.2.3**) equals the value of the Compare register, the IP7 bit 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 request.

For diagnostic purposes, the Compare register is a read/write register. Normally, this register should be only used for a write. Figure 6-4 shows the format of the Compare register.

Figure 6-4. Compare Register



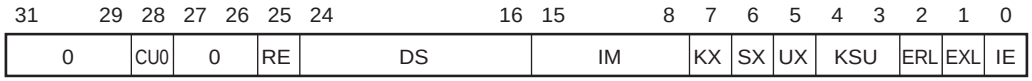
Compare: Value that is compared with the count value of the Count register.

6.2.5 Status register (12)

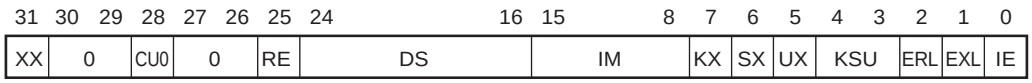
The Status register is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. Figure 6-5 shows the format of the Status register.

Figure 6-5. Status Register (1/2)

(a) VR4121, VR4122, VR4181, VR4181A



(b) VR4131



XX: Write 0 in a write operation. When this bit is read, 0 is read (VR4131 only).

CU0: Enables/disables the use of the coprocessor (1 → Enabled, 0 → Disabled).
CP0 can be used by the kernel at all times.

RE: Enables/disables reversing of the endian setting in User mode (0 → Disabled, 1 → Enabled). This bit must be set to 0 in the VR4100 Series.

DS: Diagnostic Status field (see **Figure 6-6**).

Figure 6-5. Status Register (2/2)

IM: Interrupt Mask field used to enable/disable interrupts (0 → Disabled, 1 → Enabled). This field consists of 8 bits that are used to control eight interrupts. The bits are assigned to interrupts as follows:

- IM7: Masks a timer interrupt.
- IM(6:2): Mask ordinary interrupts (Int(4:0)[Note]). However, Int3[Note] occurs in the VR4121 and VR4181A only, and Int4[Note] in the VR4181A only.
- IM(1:0): Mask software interrupts.

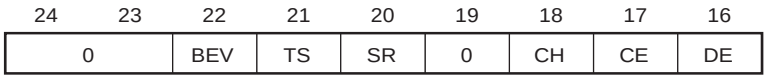
Note Int(4:0) are internal signals of the CPU core. For details about connection to the on-chip peripheral units, refer to **Hardware User's Manual** of each processor.

KX: Enables 64-bit addressing in Kernel mode (0 → 32-bit, 1 → 64-bit).
SX: Enables 64-bit addressing and operation in Supervisor mode (0 → 32-bit, 1 → 64-bit).
UX: Enables 64-bit addressing and operation in User mode (0 → 32-bit, 1 → 64-bit).
KSU: Sets and indicates the operating mode (00 → Kernel, 01 → Supervisor, 10 → User).
ERL: Sets and indicates the error level (0 → Normal, 1 → Error).
EXL: Sets and indicates the exception level (0 → Normal, 1 → Exception).
IE: Sets and indicates interrupt enabling/disabling (0 → Disabled, 1 → Enabled).
0: Reserved for future use. Write 0 in a write operation. When this bit is read, 0 is read.

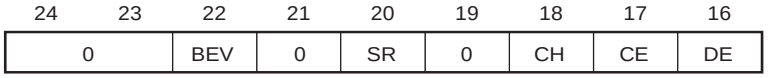
Figure 6-6 shows the details of the Diagnostic Status (DS) field. All DS field bits other than the TS bit are readable and writable.

Figure 6-6. Status Register Diagnostic Status Field

(a) VR4181



(b) VR4121, VR4122, VR4131, VR4181A



BEV: Specifies the base address of a TLB Refill exception vector and common exception vector (0 → Normal, 1 → Bootstrap).

TS: Occurs the TLB to be shut down (VR4181 only) (0 → Not shut down, 1 → Shut down). This bit is read only and used to avoid any problems that may occur when multiple TLB entries match the same virtual address. After the TLB has been shut down, reset the processor to enable restart. Note that the TLB is shut down even if a TLB entry matching a virtual address is marked as being invalid (with the V bit cleared).

SR: Occurs a Soft Reset or NMI exception (0 → Not occurred, 1 → Occurred).

CH: CP0 condition bit (0 → False, 1 → True). This bit can be read and written by software only; it cannot be accessed by hardware.

CE, DE: These are prepared to maintain compatibility with the VR4100, and are not used in the VR4100 Series hardware.

0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

The Status register has the following fields where the modes and access status are set.

(1) Interrupt enable

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

- IE bit is set to 1.
- EXL bit is cleared to 0.
- ERL bit is cleared to 0.
- The appropriate bit of the IM field is set to 1.

(2) Operating modes

The following Status register bit settings are required for User, Kernel, and Supervisor modes.

- The processor is in User mode when KSU field = 10, EXL bit = 0, and ERL bit = 0.
- The processor is in Supervisor mode when KSU field = 01, EXL bit = 0, and ERL bit = 0.
- The processor is in Kernel mode when KSU field = 00, EXL bit = 1, or ERL bit = 1.

Access to the kernel address space is allowed when the processor is in Kernel mode.
Access to the supervisor address space is allowed when the processor is in Supervisor or Kernel mode.
Access to the user address space is allowed in any of the three operating modes.

(3) Addressing modes

The following 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 bit = 1. If this bit is set, an XTLB Refill exception occurs if a TLB miss occurs in the Kernel mode address space. 64-bit operations are always valid in Kernel mode.
- 64-bit addressing and operations are enabled for Supervisor mode when SX bit = 1. If this bit is set, an XTLB Refill exception occurs if a TLB miss occurs in the Supervisor mode address space.
- 64-bit addressing and operations are enabled for User mode when UX bit = 1. If this bit is set, an XTLB Refill exception occurs if a TLB miss occurs in the User mode address space.

(4) Status after reset

The contents of the Status register are undefined after Cold resets, except for the following bits in the diagnostic status field.

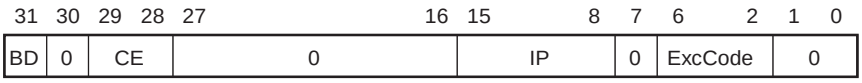
- TS bit is cleared to 0 (VR4181 only).
- SR bit is cleared to 0.
  SR bit is 0 after Cold reset, and is 1 after Soft reset or NMI exception.
- ERL and BEV bits are both set to 1.

Remark Cold reset and Soft reset are CPU core reset. For details, refer to **Hardware User's Manual** of each processor.

6.2.6 Cause register (13)

The 32-bit read/write Cause register holds the cause of the most recent exception. A 5-bit exception code indicates one of the causes (see **Table 6-2**). Other bits holds the detailed information of the specific exception. All bits in the Cause register, with the exception of the IP1 and IP0 bits, are read-only; IP1 and IP0 are used for software interrupts. Figure 6-7 shows the fields of this register; Table 6-2 describes the Cause register codes.

Figure 6-7. Cause Register



BD: Indicates whether the most recent exception occurred in the branch delay slot (1 → In delay slot, 0 → Normal).

CE: Indicates the coprocessor number in which a Coprocessor Unusable exception occurred.
This field will remain undefined for as long as no exception occurs.

IP: Indicates whether an interrupt is pending (1 → Interrupt pending, 0 → No interrupt pending).
The bits are assigned to interrupts as follows:

- IM7: A timer interrupt.
- IM(6:2): Ordinary interrupts (Int(4:0)[Note]). However, Int3[Note] occurs in the VR4121 and VR4181A only, and Int4[Note] in the VR4181A only.
- IM(1:0): Software interrupts. Only these bits cause an interrupt exception, when they are set to 1 by means of software.

Note Int(4:0) are internal signals of the CPU core. For details about connection to the on-chip peripheral units, refer to **Hardware User's Manual** of each processor.

ExcCode: Exception code field (see **Table 6-2**).

0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

Table 6-2. Cause Register Exception Code Field

Exception code	Mnemonic	Description
0	Int	Interrupt exception
1	Mod	TLB Modified exception
2	TLBL	TLB Refill exception (load or fetch)
3	TLBS	TLB Refill exception (store)
4	AdEL	Address Error exception (load or fetch)
5	AdES	Address Error exception (store)
6	IBE	Bus Error exception (instruction fetch)
7	DBE	Bus Error exception (data load or store)
8	Sys	System Call exception
9	Bp	Breakpoint exception
10	RI	Reserved Instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Integer Overflow exception
13	Tr	Trap exception
14 to 22	—	Reserved for future use
23	WATCH	Watch exception
24 to 31	—	Reserved for future use

The VR4100 Series has eight interrupt request sources, IP7 to IP0, that are used for the following purpose. For the detailed description of interrupts, refer to Chapter 8.

(1) IP7

This bit indicates whether there is a timer interrupt request.

It is set when the values of Count register and Compare register match.

(2) IP6 to IP2

IP6 to IP2 reflect the state of the interrupt request signal of the CPU core.

(3) IP1 and IP0

These bits are used to set/clear a software interrupt request.

6.2.7 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. The contents of this register change depending on whether execution of MIPS16 instructions is enabled or disabled. Setting the MIPS16EN pin after RTC reset specifies whether execution of the MIPS16 instructions is enabled or disabled.

When the MIPS16 instruction execution is disabled, the EPC register contains either:

- Virtual address of the instruction that caused the exception, or
- Virtual address of the immediately preceding branch or jump instruction (when the instruction associated with the exception is in a branch delay slot, and the BD bit in the Cause register is set to 1).

When the MIPS16 instruction execution is enabled, the EPC register contains either:

- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs, or
- Virtual address of the immediately preceding branch or jump instruction and ISA mode at which an exception occurs (when the instruction associated with the exception is in a branch delay slot of the jump instruction, and the BD bit in the Cause register is set to 1).

When the 16-bit instruction is executed, the EPC register contains either:

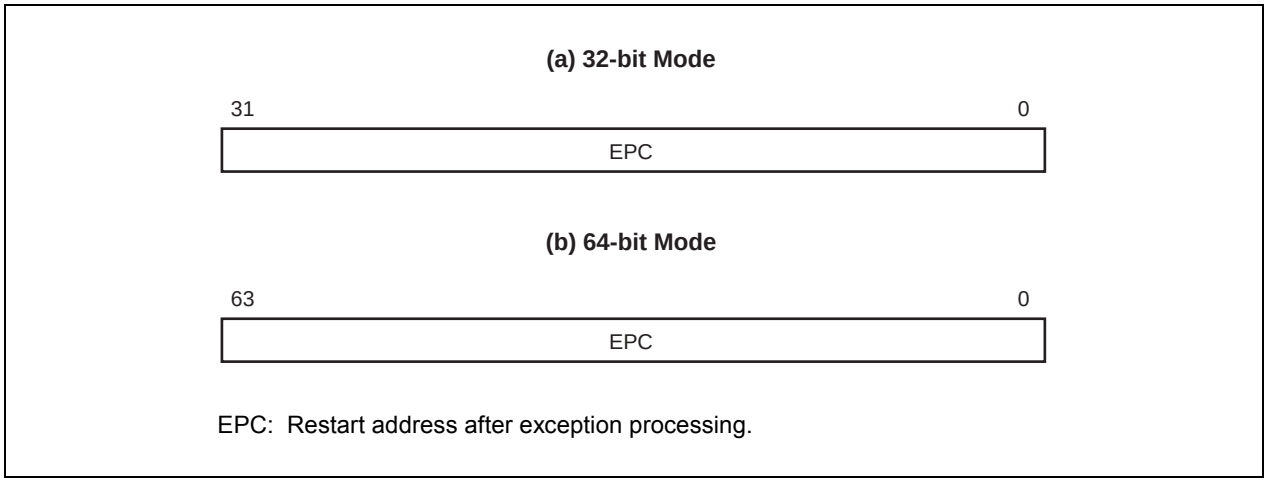
- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs, or
- Virtual address of the immediately preceding Extend or jump instruction and ISA mode at which an exception occurs (when the instruction associated with the exception is in a branch delay slot of the jump instruction or in the instruction following the Extend instruction, and the BD bit in the Cause register is set to 1).

The EXL bit in the Status register is set to 1 to keep the processor from overwriting the address of the exception-causing instruction contained in the EPC register in the event of another exception.

The EPC register never indicates the address of the instruction in branch delay slot.

Figure 6-8 shows the EPC register format when MIPS16 ISA is disabled, and Figure 6-9 shows the EPC register format when MIPS16 ISA is enabled.

Figure 6-8. EPC Register (When MIPS16 ISA Is Disabled)



EPC: Restart address after exception processing.

Figure 6-9. EPC Register (When MIPS16 ISA Is Enabled)

EPC: Bits 31 to 1 of restart address after exception processing.

EIM: ISA mode at which an exception occurs.
(1 → when MIPS16 SIA instruction is executed, 0 → when MIPS III ISA instruction is executed.)

EPC: Bits 63 to 1 of restart address after exception processing.

EIM: ISA mode at which an exception occurs.
(1 → when MIPS16 SIA instruction is executed, 0 → when MIPS III ISA instruction is executed.)

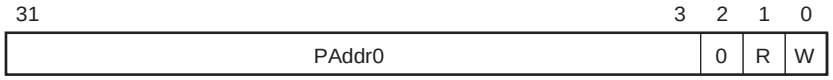
6.2.8 WatchLo (18) and WatchHi (19) registers

The VR4100 Series processor provides a debugging feature to detect references to a selected physical address; load and store instructions to the location specified by the WatchLo and WatchHi registers cause a Watch exception.

Figures 6-10 and 6-11 show the format of the WatchLo and WatchHi registers.

The contents of these registers after reset are undefined so that they must be initialized by software.

Figure 6-10. WatchLo Register



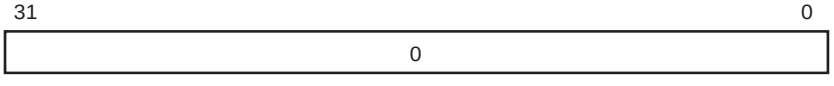
PAddr0: Specifies physical address bits 31 to 3.

R: Specifies detection of watch address references when load instructions are executed (1 → Detect, 0 → Not detect).

W: Specifies detection of watch address references when store instructions are executed (1 → Detect, 0 → Not detect).

0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

Figure 6-11. WatchHi Register



0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

6.2.9 XContext register (20)

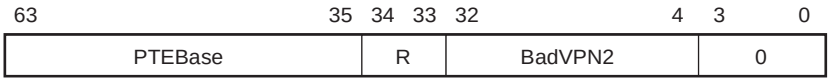
The read/write XContext register contains a pointer to an entry in the page table entry (PTE) array, an operating system data structure that stores virtual-to-physical address translations. If a TLB miss occurs, the operating system loads the untranslated data from the PTE into the TLB to handle the software error.

The XContext register is used by the XTLB Refill exception handler to load TLB entries in 64-bit addressing mode.

The XContext register duplicates some of the information provided in the BadVAddr register, and puts it in a form useful for the XTLB exception handler.

This register is included solely for operating system use. The operating system sets the PTEBase field in the register, as needed. Figure 6-12 shows the format of the XContext register.

Figure 6-12. XContext Register



PTEBase: The PTEBase field is a base address of the PTE entry table.

R: Space type (00 → User, 01→ Supervisor, 11 → Kernel). The setting of this field matches virtual address bits 63 and 62.

BadVPN2: This field holds the value (VPN2) obtained by halving the virtual page number of the most recent virtual address for which translation failed.

0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

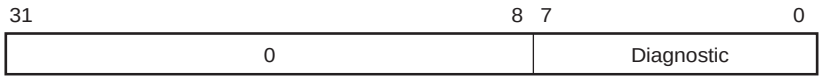
The 29-bit BadVPN2 field has bits 39 to 11 of the virtual address that caused the TLB miss; bit 10 is excluded because a single TLB entry maps to an even-odd page pair. For a 1 KB page size, this format may be used directly to address the pair-table of 8-byte PTEs. For 4 KB-or-more page and PTE sizes, shifting or masking this value produces the appropriate address.

6.2.10 Parity Error register (26)

The Parity Error (PErr) register is a readable/writable register. This register is defined to maintain software-compatibility with the VR4100, and is not used in hardware because the VR4100 Series has no parity.

Figure 6-13 shows the format of the PErr register.

Figure 6-13. Parity Error Register



Diagnostic: 8-bit self diagnostic field.

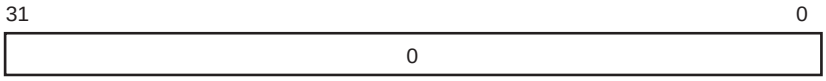
0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

6.2.11 Cache Error register (27)

The Cache Error register is a readable/writable register. This register is defined to maintain software-compatibility with the VR4100, and is not used in hardware because the VR4100 Series has no parity.

Figure 6-14 shows the format of the Cache Error register.

Figure 6-14. Cache Error Register



0: Reserved for future use. Write 0 in a write operation. When this field is read, 0 is read.

6.2.12 ErrorEPC register (30)

The Error Exception Program Counter (ErrorEPC) register is similar to the EPC register. It is used to store the Program Counter value at which the Cold Reset, Soft Reset, or NMI exception has been serviced.

The read/write ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error. The contents of this register change depending on whether execution of MIPS16 instructions is enabled or disabled. Setting the MIPS16EN pin after RTC reset specifies whether the execution of MIPS16 instructions is enabled or disabled.

When the MIPS16 ISA is disabled, this address can be:

- Virtual address of the instruction that caused the exception, or
- Virtual address of the immediately preceding branch or jump instruction, when the instruction associated with the error exception is in a branch delay slot.

When the MIPS16 instruction execution is enabled during a 32-bit instruction execution, this address can be:

- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs, or
- Virtual address of the immediately preceding branch or jump instruction and ISA mode at which an exception occurs when the instruction associated with the exception is in a branch delay slot.

When the MIPS16 instruction execution is enabled during a 16-bit instruction execution, this address can be:

- Virtual address of the instruction that caused the exception and ISA mode at which an exception occurs, or
- Virtual address of the immediately preceding jump instruction or Extend instruction and ISA mode at which an exception occurs when the instruction associated with the exception is in a branch delay slot of the jump instruction or is the instruction following the Extend instruction.

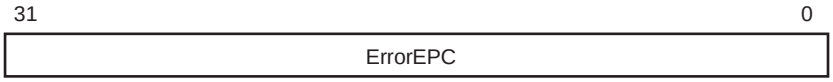
The contents of the ErrorEPC register do not change when the ERL bit of the Status register is set to 1. This prevents the processor when other exceptions occur from overwriting the address of the instruction in this register which causes an error exception.

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

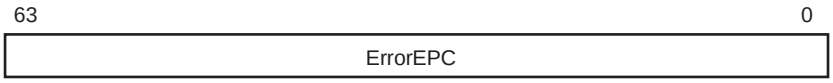
Figure 6-15 shows the format of the ErrorEPC register when the MIPS16ISA is disabled. Figure 6-16 shows the format of the ErrorEPC register when the MIPS16ISA is enabled.

Figure 6-15. ErrorEPC Register (When MIPS16 ISA Is Disabled)

(a) 32-bit Mode



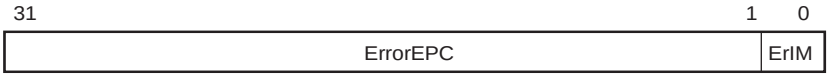
(b) 64-bit Mode



ErrorEPC: Program counter that indicates the restart address after Cold reset, Soft reset, or NMI exception.

Figure 6-16. ErrorEPC Register (When MIPS16 ISA Is Enabled)

(a) 32-bit mode



ErrorEPC: Bits 31 to 1 of virtual restart address after Cold reset, Soft reset, or NMI exception.
ErIM: ISA mode at which an error exception occurs (1 → MIPS16 ISA, 0 → MIPS III ISA).

(b) 64-bit mode

ErrorEPC: Bits 63 to 1 of virtual restart address after Cold reset, Soft reset, or NMI exception.
ErIM: ISA mode at which an error exception occurs (1 → MIPS16 ISA, 0 → MIPS III ISA).

6.3 Overview of Exceptions

When the processor takes an exception, the EXL bit is set to 1, meaning the system is in Kernel mode. After saving the appropriate state, the exception handler typically resets the EXL bit back to 0. The exception handler sets the EXL bit to 1 so that the saved state is not lost upon the occurrence of another exception while the saved state is being restored.

Returning from an exception also resets the EXL bit to 0. For details, see **CHAPTER 9 CPU INSTRUCTION SET DETAILS**.

Remark When the EXL and ERL bits in the Status register are 0, either User, Supervisor, or Kernel operating mode is specified by the KSU bits in the Status register. When either the EXL or ERL bit is set to 1, the processor is in Kernel mode.

6.3.1 Exception types

Exceptions are classified to as follows according to the internal status of the processor retained at the occurrence of an exception.

- Cold Reset
- Soft Reset, NMI
- Remaining processor exceptions (common exceptions)

6.3.2 Exception vector locations

When an exception occurs, the exception vector address is set to the program counter and the processing branches to there from the main program. A program called exception handler that processes exceptions must be placed at the location of the exception vector address.

A vector address is calculated by adding a vector offset to a base address. Each exception type has a different vector address.

64-/32-bit mode exception vectors and their offsets are shown below.

Table 6-3. 32-Bit Mode Exception Vector Base Addresses

Exception	Vector base address (virtual)	Vector offset
Cold Reset Soft Reset NMI	0xBFC0 0000 (BEV is automatically set to 1)	0x0000
TLB Refill (EXL = 0)	0x8000 0000 (BEV = 0)	0x0000
XTLB Refill (EXL = 0)	0xBFC0 0200 (BEV = 1)	0x0080
Others		0x0180

Table 6-4. 64-Bit Mode Exception Vector Base Addresses

Exception	Vector base address (virtual)	Vector offset
Cold Reset Soft Reset NMI	0xFFFF FFFF BFC0 0000 (BEV is automatically set to 1)	0x0000
TLB Refill (EXL = 0)	0xFFFF FFFF 8000 0000 (BEV = 0)	0x0000
XTLB Refill (EXL = 0)	0xFFFF FFFF BFC0 0200 (BEV = 1)	0x0080
Others		0x0180

(1) Vector of Cold Reset, Soft Reset, and NMI exceptions

The Cold Reset, Soft Reset, and NMI exceptions are always branched to the following reset exception vector address (virtual). This address is in an uncached, unmapped space.

- 0xBFC0 0000 in 32-bit mode
- 0xFFFF FFFF BFC0 0000 in 64-bit mode

(2) TLB Refill exception vector

When BEV bit = 0, the vector base address (virtual) for the TLB Refill exception is in kseg0 (unmapped) space.

- 0x8000 0000 in 32-bit mode
- 0xFFFF FFFF 8000 0000 in 64-bit mode

When BEV bit = 1, the vector base address (virtual) for the TLB Refill exception is in kseg1 (uncached, unmapped) space.

- 0xBFC0 0200 in 32-bit mode
- 0xFFFF FFFF BFC0 0200 in 64-bit mode

This is an uncached, non-TLB-mapped space, allowing the exception handler to bypass the cache and TLB.

(3) Common exception vector

Addresses for the remaining exceptions are a combination of a vector offset and a base address.

6.3.3 Priority of exceptions

While more than one exception can occur for a single instruction, only the exception with the highest priority is reported. Table 6-5 lists the priorities.

Table 6-5. Exception Priority Order

Priority	Exceptions
High	Cold Reset
↑	Soft Reset
|	NMI
|	Address Error (instruction fetch)
|	TLB/XTLB Refill (instruction fetch)
|	TLB Invalid (instruction fetch)
|	Bus Error (instruction fetch)
|	System Call
|	Breakpoint
|	Coprocessor Unusable
|	Reserved Instruction
|	Trap
|	Integer Overflow
|	Address Error (data access)
|	TLB/XTLB Refill (data access)
|	TLB Invalid (data access)
|	TLB Modified (data write)
|	Watch
↓	Bus Error (data access)
Low	Interrupt (other than NMI)

Hereafter, handling exceptions by hardware is referred to as “process”, and handling exception by software is referred to as “service”.

6.4 Details of Exceptions

6.4.1 Cold Reset exception

Cause

The Cold Reset exception occurs when the ColdReset# signal (internal) is asserted and then deasserted. This exception is not maskable. The Reset# signal (internal) must be asserted along with the ColdReset# signal (for details, see **Hardware User's Manual** of each processor).

Processing

The CPU provides a special interrupt vector for this exception:

- 0xBFC0 0000 (virtual) in 32-bit mode
- 0xFFFF FFFF BFC0 0000 (virtual) in 64-bit mode

The Cold 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:

- When the MIPS16 instruction execution is disabled while the ERL of Status register is 0, the PC value at which an exception occurs is set to the ErrorEPC register.
  When the MIPS16 instruction execution is enabled while the ERL of Status register is 0, the PC value at which an exception occurs is set to the ErrorEPC register and the ISA mode in which an exception occurs is set to the least significant bit of the ErrorEPC register.
- TS (VR4181 only) and SR of the Status register are cleared to 0.
- ERL and BEV of the Status register are set to 1.
- The Random register is initialized to the value of its upper bound (31).
- The Wired register and the Count register are initialized to 0.
- R and W of the WatchLo register are cleared to 0 (other than VR4181).
- IS and BP of the Config register are cleared to 0 (VR4122, VR4131, and VR4181A only).
- In the VR4121 and VR4181, bits 31 to 28 and bits 22 to 3 of the Config register are set to fixed values.
- In the VR4122, bits 30 to 28, bits 22 to 17, bits 15 to 6, bit 4, and bit 3 of the Config register are set to fixed values.
- In the VR4131 and VR4181A, bits 30 to 28, bits 22 to 17, bits 15 to 6, and bit 3 of the Config register are set to fixed values.
- All other bits are undefined.

Servicing

The Cold Reset exception is serviced by:

- Initializing all processor registers, coprocessor registers, TLB, caches, and the memory system
- Performing diagnostic tests
- Bootstrapping the operating system

6.4.2 Soft Reset exception

Cause

A Soft Reset (sometimes called Warm Reset) occurs when the ColdReset# signal remains deasserted while the Reset# signal goes from assertion to deassertion (for details, see **Hardware User's Manual** of each processor).

A Soft Reset immediately resets all state machines, and sets the SR bit of the Status register. Execution begins at the reset vector when the Reset# is deasserted. This exception is not maskable.

Caution In the VR4100 Series, a Soft Reset never occurs.

Processing

The CPU provides a special interrupt vector for this exception (same location as Cold Reset):

- 0xBFC0 0000 (virtual) in 32-bit mode
- 0xFFFF FFFF BFC0 0000 (virtual) in 64-bit mode

This vector is located within unmapped and uncached address space, so that the cache and TLB need not be initialized to process this exception. The SR bit of the Status register is set to 1 to distinguish this exception from a Cold Reset exception.

When this exception occurs, the contents of all registers are preserved except for the following registers:

- When the MIPS16 instruction execution is disabled, the PC value at which an exception occurs is set to the ErrorEPC register.
  When the MIPS16 instruction execution is enabled, the PC value at which an exception occurs is set to the ErrorEPC register and the ISA mode in which an exception occurs is set to the least significant bit of the ErrorEPC register.
- TS bit of the Status register is cleared to 0 (VR4181 only).
- ERL, SR, and BEV bits of the Status register are set to 1.
- R and W of the WatchLo register are cleared to 0 (other than VR4181).

During a Soft Reset, access to the operating cache or system interface may be aborted. This means that the contents of the cache and memory will be undefined if a Soft Reset occurs.

Servicing

The Soft Reset exception is serviced by:

- Preserving the current processor states for diagnostic tests
- Reinitializing the system in the same way as for a Cold Reset exception

6.4.3 NMI exception

Cause

The Nonmaskable Interrupt (NMI) exception occurs when the NMI signal (internal) becomes active. This interrupt is not maskable; it occurs regardless of the settings of the EXL, ERL, and the IE bits in the Status register (for details, see **CHAPTER 8 CPU CORE INTERRUPTS**).

Processing

The CPU provides a special interrupt vector for this exception:

- 0xBFC0 0000 (virtual) in 32-bit mode
- 0xFFFF FFFF BFC0 0000 (virtual) in 64-bit mode

This vector is located within unmapped and uncached address space so that the cache and TLB need not be initialized to process an NMI interrupt. The SR bit of the Status register is set to 1 to distinguish this exception from a Cold Reset exception.

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

When this exception occurs, the contents of all registers are preserved except for the following registers:

- When the MIPS16 instruction execution is disabled, the PC value at which an exception occurs is set to the ErrorEPC register.
  When the MIPS16 instruction execution is enabled, the PC value at which an exception occurs is set to the ErrorEPC register and the ISA mode in which an exception occurs is set to the least significant bit of the ErrorEPC register.
- The TS bit of the Status register is cleared to 0 (VR4181 only).
- The ERL, SR, and BEV bits of the Status register are set to 1.

Servicing

The NMI exception is serviced by:

- Preserving the current processor states for diagnostic tests
- Reinitializing the system in the same way as for a Cold Reset exception

6.4.4 Address Error exception

Cause

The Address Error exception occurs when an attempt is made to execute one of the following. This exception is not maskable.

- Execution of the LW, LWU, SW, or CACHE instruction for word data that is not located on a word boundary
- Execution of the LH, LHU, or SH instruction for half-word data that is not located on a half-word boundary
- Execution the LD or SD instruction for double-word data that is not located on a double-word boundary
- Referencing the kernel address space in User or Supervisor mode
- Referencing the supervisor space in User mode
- Referencing an address that does not exist in the kernel, user, or supervisor address space in 64-bit Kernel, User, or Supervisor mode
- Branching to an address that was not located on a ward boundary when the MIPS16 instruction is disabled
- Branching to address whose least-significant 2 bits are 10 when the MIPS16 instruction is enabled

Processing

The common exception vector is used for this exception. The AdEL or AdES code in the Cause register is set. If this exception has been caused by an instruction reference or load operation, AdEL is set. If it has been caused by a store operation, AdES is set.

When this exception occurs, the BadVAddr register stores the virtual address that was not properly aligned or was referenced in 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.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

The kernel reports the UNIX™ SIGSEGV (segmentation violation) signal to the current process, and this exception is usually fatal.

6.4.5 TLB exceptions

Three types of TLB exceptions can occur:

- A TLB Refill exception occurs when there is no TLB entry that matches a referenced address.
- A TLB Invalid exception occurs when a TLB entry that matches a referenced virtual address is marked as being invalid (with the V bit set to 0).
- A TLB Modified exception occurs when a TLB entry that matches a virtual address referenced by the store instruction is marked as being valid (with the V bit set to 1) though a write to it is disabled (with the D bit set to 0).

The following three sections describe these TLB exceptions.

(1) TLB Refill exception (32-bit space mode)/XTLB Refill exception (64-bit space mode)

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. When the EXL bit of the Status register is set to 0, either of these two special vectors is referenced. When the EXL bit is set to 1, the common exception vector is referenced.

This exception sets the TLBL or TLBS code in the ExcCode field of the Cause register. If this exception has been caused by an instruction reference or load operation, TLBL is set. If it has been caused by a store operation, TLBS is set.

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.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

To service this exception, the contents of the Context or XContext register are used as a virtual address to fetch memory words containing the physical page frame and access control bits for a pair of TLB entries. The memory word is written into the TLB entry by using the EntryLo0, EntryLo1, or EntryHi register.

It is possible that the physical page frame and access control bits are placed in a page where the virtual address is not resident in the TLB. This condition is processed by allowing a TLB Refill exception in the TLB Refill exception handler. In this case, the common exception vector is used because the EXL bit of the Status register is set to 1.

(2) TLB Invalid exception

Cause

The TLB Invalid exception occurs when the TLB entry that matches with the virtual address to be referenced is invalid (the V bit is set to 0). 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. If this exception has been caused by an instruction reference or load operation, TLBL is set. If it has been caused by a store operation, TLBS 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 Random register normally stores a valid location in which to place the replacement TLB entry. The contents of the EntryLo register are undefined.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

Usually, the V bit of a TLB entry is cleared in the following cases:

- When the virtual address does not exist
- When the virtual address exists, but is not in main memory (a page fault)
- When a trap is required 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 a TLBP (TLB Probe) instruction, and replaced by an entry with its V bit set to 1.

(3) TLB Modified exception

Cause

The TLB Modified exception occurs when the TLB entry that matches with the virtual address referenced by the store instruction is valid (bit V is 1) but is not writable (bit D is 0). This exception is not maskable.

Processing

The common exception vector is used for this exception, and the Mod code in the ExcCode field of 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 are undefined.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

The kernel uses the failed virtual address or virtual page number to identify the corresponding access control bits. 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 (i.e. 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 word data containing the physical page frame and access control bits (with the D bit set to 1) is loaded to the EntryLo register, and the contents of the EntryHi and EntryLo registers are written into the TLB.

6.4.6 Bus Error exception

Cause

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

A Bus Error exception occurs only when a cache miss refill, uncached reference, or unbuffered write occurs synchronously.

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 caused the exception by an instruction reference, load operation, or store operation.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Note that the EPC register may indicate a succeeding instruction instead of the instruction that caused the exception if the Instruction Streaming function is on in the VR4122, VR4131, and VR4181A.

Servicing

The physical address at which the fault occurred can be computed from information available in the System Control Coprocessor (CP0) registers.

- If the IBE code in the Cause register is set (indicating an instruction fetch), the virtual address is contained in the EPC register.
- If the DBE code is set (indicating a load or store), the virtual address of the instruction that caused the exception is saved to the EPC register.

The virtual address of the load and store instruction 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.

At the time of this exception, the kernel reports the UNIX SIGBUS (bus error) signal to the current process, but the exception is usually fatal.

6.4.7 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 ExcCode field of 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 to 1; 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 before returning.

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

6.4.8 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 ExcCode field of the Cause register is set.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

If the BREAK instruction is in a branch delay slot, the BD bit of the Status register is set to 1; otherwise this 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 to 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 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 before returning.

When a Breakpoint exception occurs while executing the MIPS16 instruction, a valve of 2 should be added to the EPC register before returning.

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

6.4.9 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 (Status register bit, CU0 = 0), or
- CP0 instructions, when the unit has not been marked usable (Status register bit, CU0 = 0) and the process executes in 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 ExcCode field of the Cause register is set. The CE bit of the Cause register indicates which of the four coprocessors was referenced.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

The coprocessor unit to which an attempted reference was made is identified by the CE bit of the Cause register. One of the following processing is performed by the handler:

- If the process is entitled access to the coprocessor, the coprocessor is marked usable and the corresponding 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 in the Cause register is set to 1, 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 kernel reports UNIX SIGILL/ILL_PRIVIN_FAULT (illegal instruction/privileged instruction fault) signal to the current process, and this exception is fatal.

6.4.10 Reserved Instruction exception

Cause

The Reserved Instruction exception occurs when an attempt is made to execute one of the following instructions:

- Instruction with an undefined major opcode (bits 31 to 26)
- SPECIAL instruction with an undefined minor opcode (bits 5 to 0)
- REGIMM instruction with an undefined minor opcode (bits 20 to 16)
- 64-bit instructions in 32-bit User or Supervisor mode
- RR instruction with an undefined minor op code (bits 4 to 0) when executing the MIPS16 instruction
- I8 instruction with an undefined minor op code (bits 10 to 8) when executing the MIPS16 instruction

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 ExcCode field of the Cause register is set.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

All currently defined MIPS ISA instructions can be executed. The process executing at the time of this exception is handled by a UNIX SIGILL/ILL_RESOP_FAULT (illegal instruction/reserved operand fault) signal. This error is usually fatal.

6.4.11 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 ExcCode field of the Cause register is set.

The EPC register contains the address of the trap 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 to 1.

Servicing

At the time of a Trap exception, the kernel reports the UNIX SIGFPE/FPE_INTOVF_TRAP (floating-point exception/integer overflow) signal to the current process, but the exception is usually fatal.

6.4.12 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 ExcCode field of 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 to 1.

Servicing

At the time of the exception, the kernel reports the UNIX SIGFPE/FPE_INTOVF_TRAP (floating-point exception/integer overflow) signal to the current process, and this exception is usually fatal.

6.4.13 Watch exception

Cause

A Watch exception occurs when a load or store instruction references the physical address specified by the WatchLo/WatchHi registers. The WatchLo/WatchHi registers specify whether a load or store or both could have initiated this exception.

- When the R bit of the WatchLo register is set to 1: Load instruction
- When the W bit of the WatchLo register is set to 1: Store instruction
- When both the R bit and W bit of the WatchLo register are set to 1: Load instruction or store instruction

The CACHE instruction never causes a Watch exception.

The Watch exception is postponed while the EXL bit in the Status register is set to 1, and Watch exception is maskable by setting the EXL bit in the Status register to 1 or by setting the R or W bit in the WatchLo register to 0.

Processing

The common exception vector is used for this exception, and the WATCH code in the ExcCode field of the Cause register is set.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

The Watch exception is a debugging aid; typically the exception handler transfers control to a debugger, allowing the user to examine the situation. To continue, once the Watch exception must be disabled to execute the faulting instruction. The Watch exception must then be reenabled. The faulting instruction can be executed either by the debugger or by setting breakpoints.

The contents of the WatchLo/WatchHi register after reset are undefined so that they, especially the R and W bits, must be initialized by software, otherwise a Watch exception may occur after reset.

6.4.14 Interrupt exception

Cause

The Interrupt exception occurs when one of the eight interrupt conditions[Note] is asserted. In the VR4100 Series, interrupt requests from internal peripheral units first enter the ICU and are then notified to the CPU core via one of five interrupt sources (Int(4:0)) or NMI.

Each of the eight interrupts can be masked by clearing the corresponding bit in the IM 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 or setting the EXL/ERL bit.

Note They are 1 timer interrupt, 5 ordinary interrupts, and 2 software interrupts.

Of the five ordinary interrupts, Int3 becomes active in the VR4121 and VR4181A only, and Int4 in the VR4181A only.

For details about the Interrupt Control Unit (ICU), refer to **Hardware User's Manual** of each processor.

Processing

The common exception vector is used for this exception, and the Int code in the ExcCode field of 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 cleared) if the interrupt request signal is asserted (or deasserted) before this register is read.

When the MIPS16 instruction is disabled, the EPC register contains the address of the instruction that caused the exception. However, if this instruction is in a branch delay slot, the EPC register contains the address of the preceding jump or branch instruction, and the BD bit of the Cause register is set to 1.

When the MIPS16 instruction is enabled, the EPC register contains the address of the instruction that caused the exception, and the least significant bit stores the ISA mode in which an exception occurs. However, if this instruction is in a branch delay slot or is the instruction following the Extend instruction, the EPC register contains the address of the preceding jump or Extend instruction, and the BD bit of the Cause register is set to 1.

Servicing

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

If the interrupt is caused by hardware, the interrupt condition is cleared by deactivating the corresponding interrupt request signal.

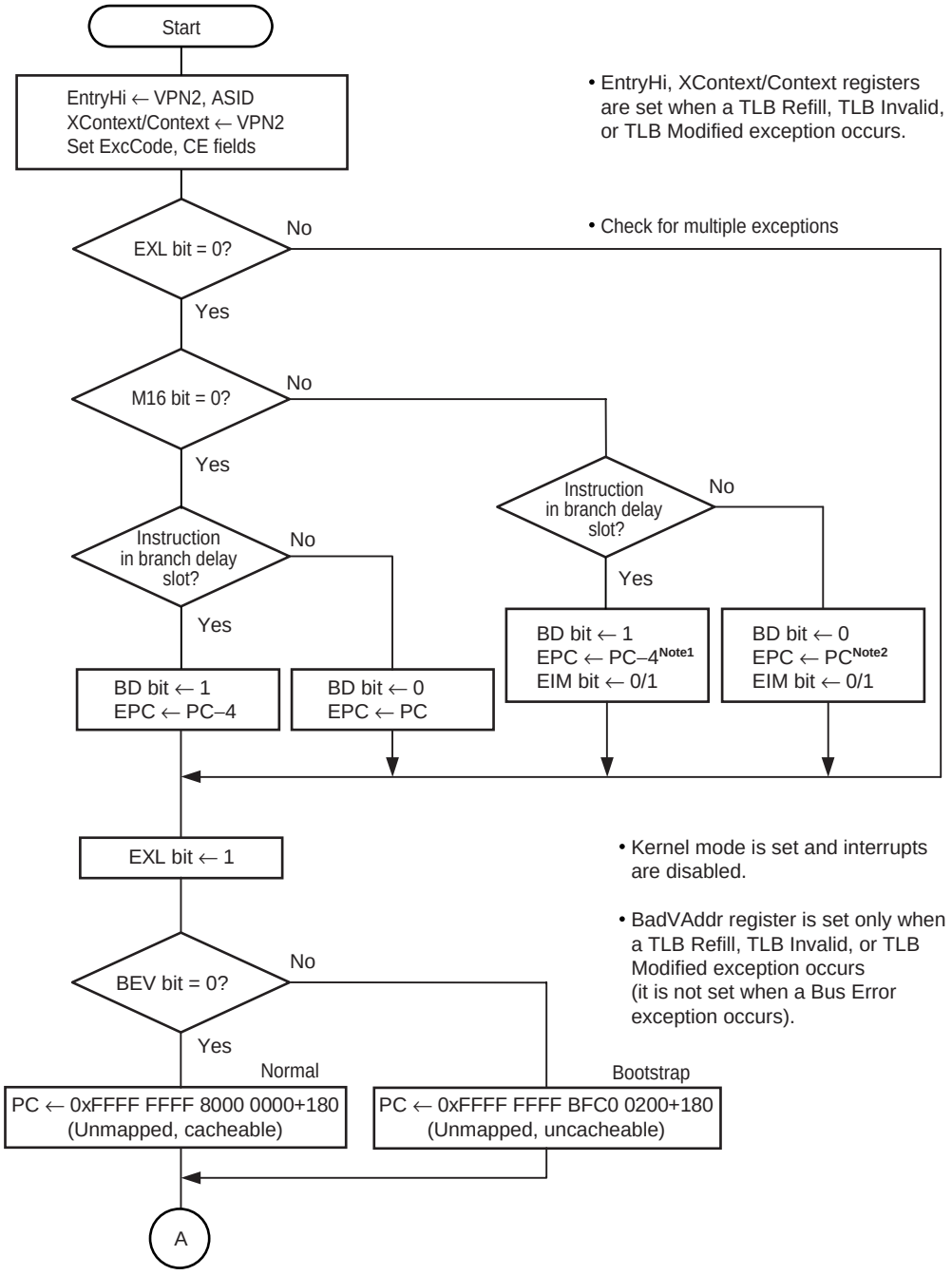
6.5 Exception Processing and Servicing Flowcharts

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

- Common exceptions and a guideline to their exception handler
- TLB/XTLB Refill exception and a guideline to their exception handler
- Cold Reset, Soft Reset and NMI exceptions, and a guideline to their handler.

Figure 6-17. Common Exception Handling (1/2)

(a) Processing by hardware



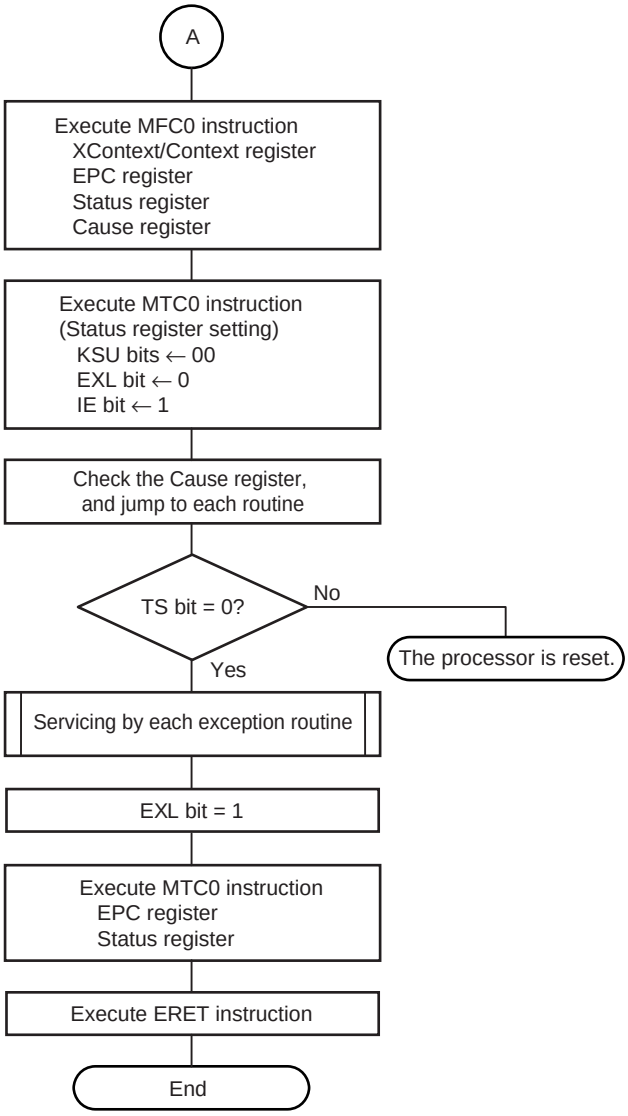
Notes 1. PC – 2 when the JR or JALR instruction of MIPS16 instructions

2. PC – 2 when the Extend instruction of MIPS16 instructions

Remark The interrupts can be masked by setting the IE or IM bit. The Watch exception can be set to pending state by setting the EXL bit to 1.

Figure 6-17. Common Exception Handling (2/2)

(b) Servicing by software



- The occurrence of TLB Refill, TLB Invalid, and TLB Modified exceptions is disabled by using an unmapped space.
- The occurrence of the Watch and Interrupt exceptions is disabled setting EXL = 1.
- The Cold Reset, Soft Reset, and NMI exceptions are enabled.
- Other exceptions are avoided in the OS programs.

- In Kernel mode, interrupts are enabled.

- After EXL = 0 is set, all exceptions are enabled (although the interrupt exception can be masked by the IE and IM bits).

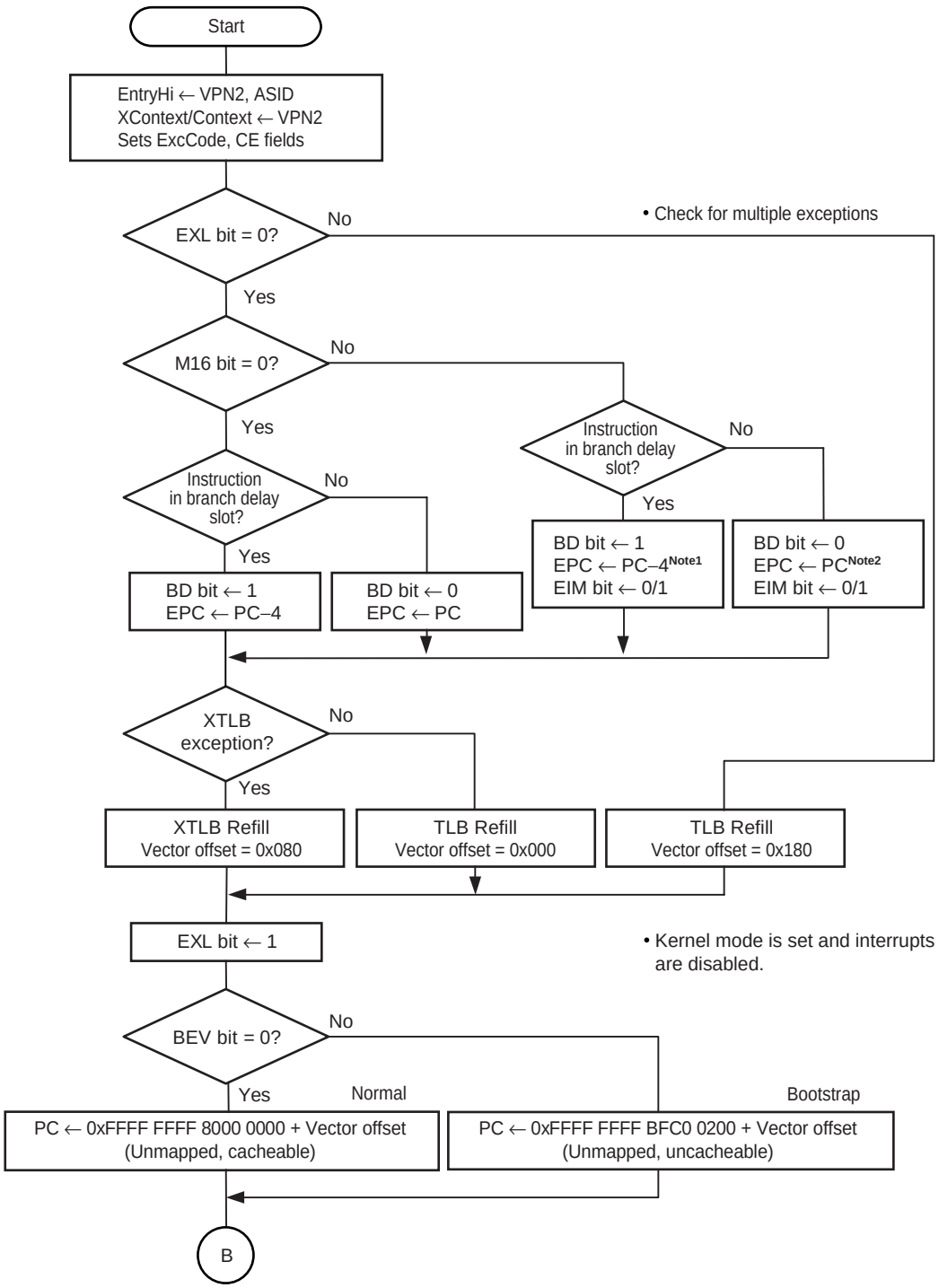
- VR4181 only.

- The register files are saved.

- The execution of the ERET instruction is disabled in the delay slots for the other jump instructions.
- The processor does not execute an instruction in the branch delay slot for the ERET instruction.
- PC ← EPC register, EXL bit ← 0

Figure 6-18. TLB/XTLB Refill Exception Handling (1/2)

(a) Processing by hardware

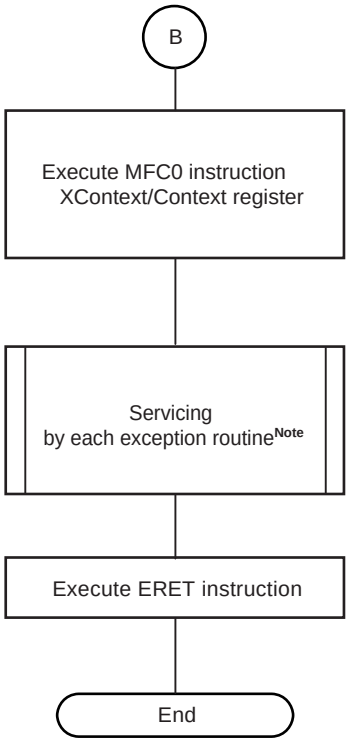


Notes 1. PC – 2 when the JR or JALR instruction of MIPS16 instructions

2. PC – 2 when the Extend instruction of MIPS16 instructions

Figure 6-18. TLB/XTLB Refill Exception Handling (2/2)

(b) Servicing by software



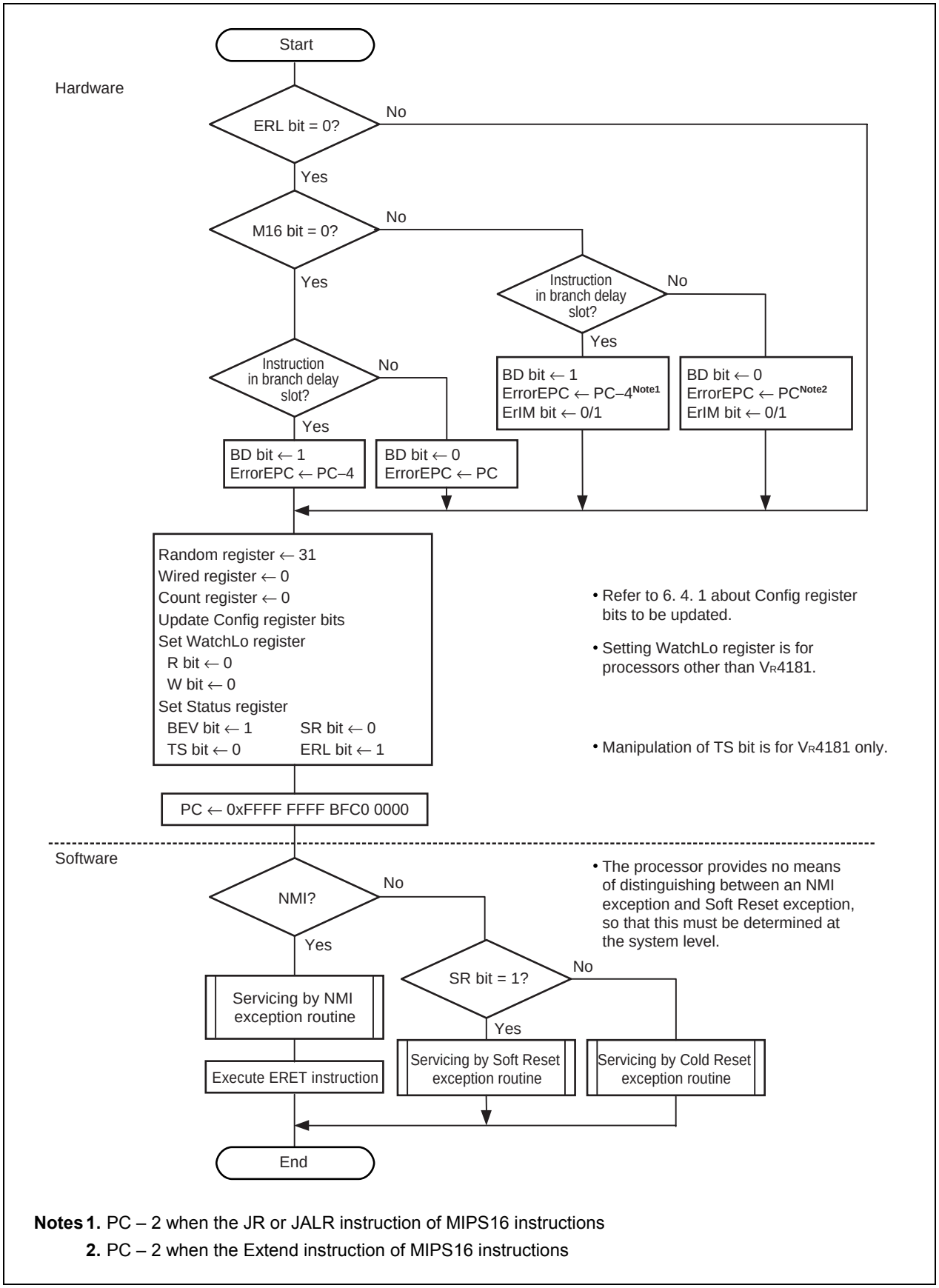
- The occurrence of TLB Refill, TLB Invalid, and TLB Modified exceptions is disabled by using an unmapped space.
- The occurrence of the Watch and Interrupt exceptions is disabled by setting EXL = 1.
- However, the Cold Reset, Soft Reset, and NMI exceptions are enabled.
- Other exceptions are avoided in the OS programs.

- The physical address for a virtual address that is loaded into the Context register is loaded into the EntryLo register and written to the TLB.

- The execution of the ERET instruction is not allowed in the branch delay slots for other jump instructions.
- The processor does not execute an instruction in the branch delay slot for the ERET instruction.
- PC ← EPC register, EXL bit ← 0

Note As long as a data/instruction address exists in the mapping space, another TLB Refill exception may occur. In such a case, EXL = 1 is set, causing a jump to the common exception vector. In this case, the common exception handler handles the TLB miss, the ERET instruction returns control to the user program, then a TLB Refill exception is generated again.

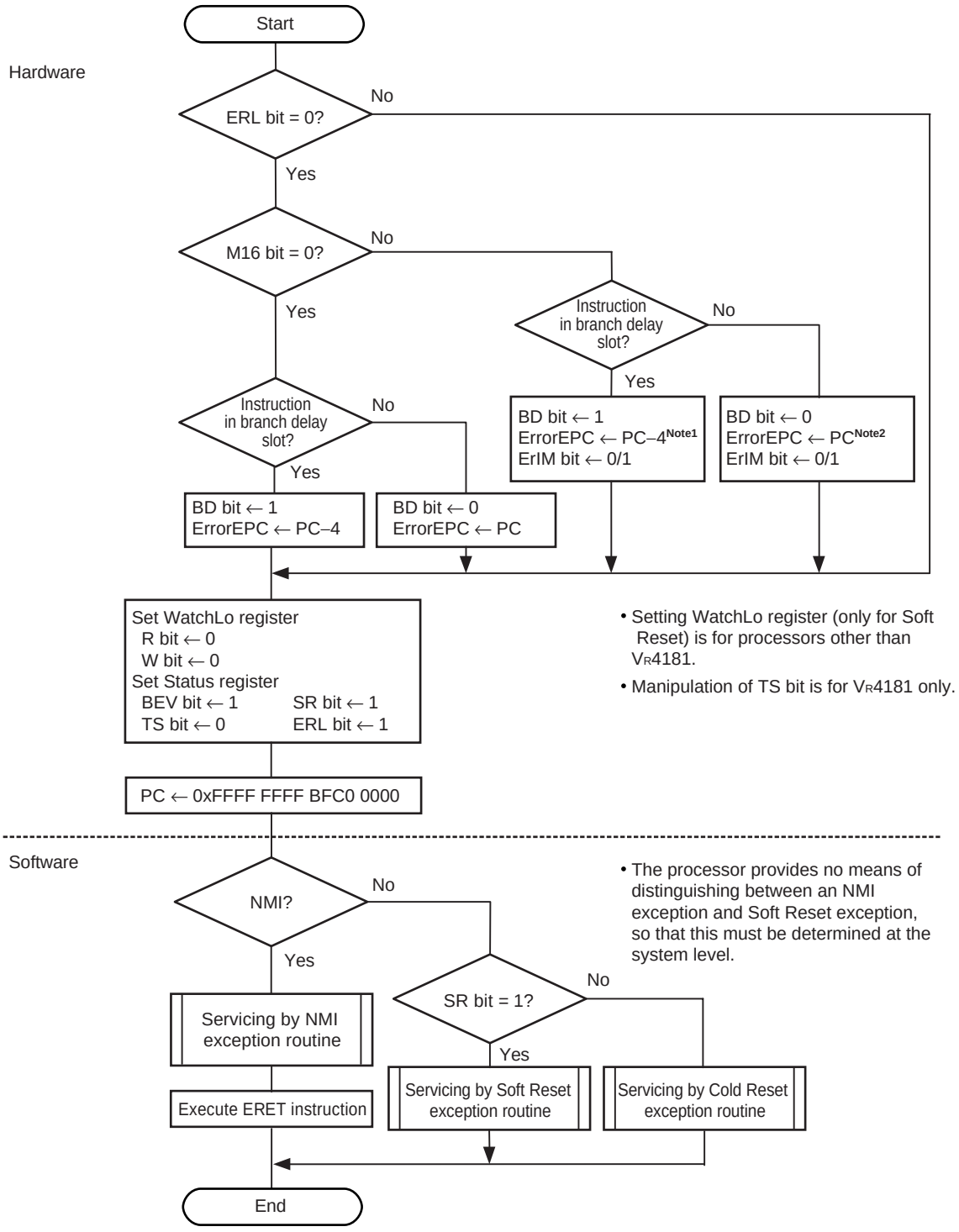
Figure 6-19. Cold Reset Exception Handling



Notes 1. PC – 2 when the JR or JALR instruction of MIPS16 instructions

2. PC – 2 when the Extend instruction of MIPS16 instructions

Figure 6-20. Soft Reset and NMI Exception Handling



Notes 1. PC – 2 when the JR or JALR instruction of MIPS16 instructions

2. PC – 2 when the Extend instruction of MIPS16 instructions

CHAPTER 7 CACHE MEMORY

This chapter describes in detail the cache memory of the VR4100 Series: its place in the CPU core 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.

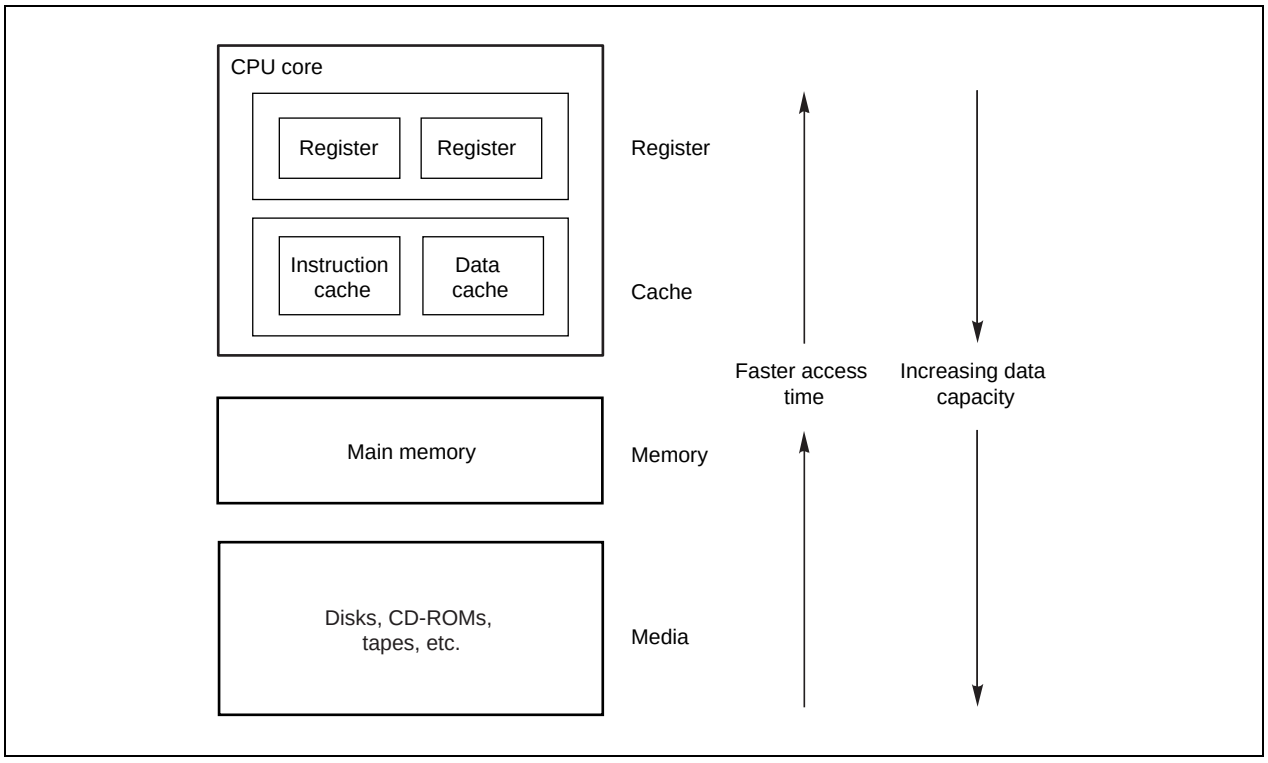
These terms are used interchangeably throughout this book.

7.1 Memory Organization

Figure 7-1 shows the CPU core 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 7-1 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 7-1. Logical Hierarchy of Memory



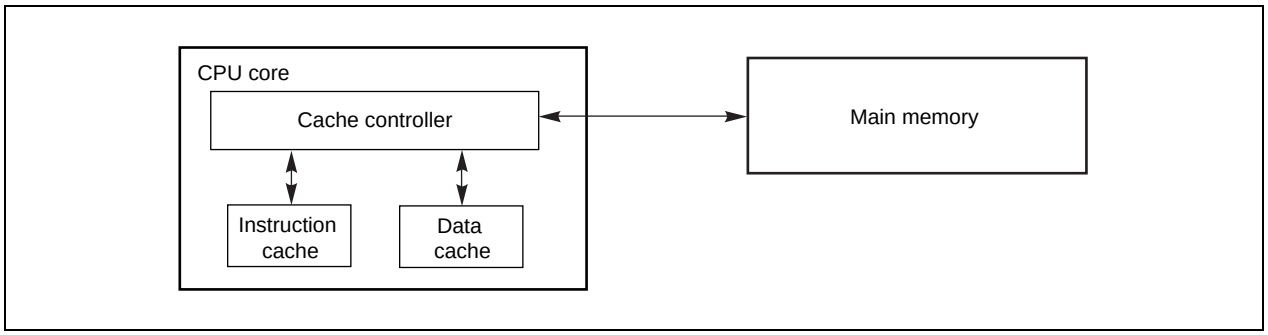
7.1.1 On-chip caches

The CPU core 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.

2 PCycles are needed to write data. However, 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 7-2 provides a relationship between cache and memory.

Figure 7-2. On-chip Caches and Main Memory



On-chip caches have the following characteristics:

- indexed with a virtual address
- holds physical address with a tag
- maintains coherency to memory with writeback

The cache data of the VR4121, VR4122, VR4181, and VR4181A are directly mapped; on the other hand those of the VR4131 are mapped in 2-way set associative format. In addition, the caches of the VR4131 have line lock function.

7.2 Cache Organization

This section describes the organization of the on-chip data and instruction caches.

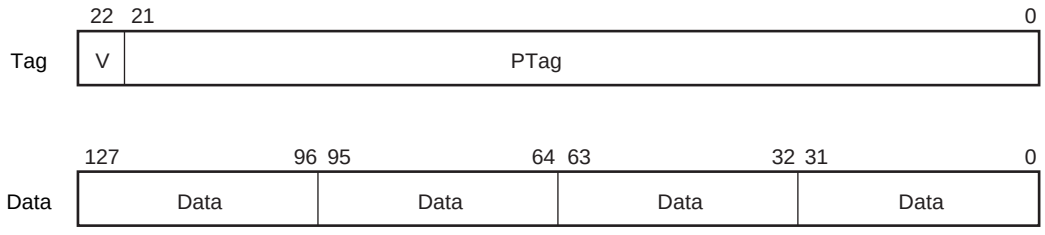
A cache consists of blocks called cache lines, which is the smallest unit of information that can be fetched from main memory to a cache. A cache line itself has tag and data fields. Two types of line size can be selectable by setting the Config register of the CP0 for the instruction cache line of the VR4122 and for the instruction/data cache line of the VR4131.

7.2.1 Instruction cache line

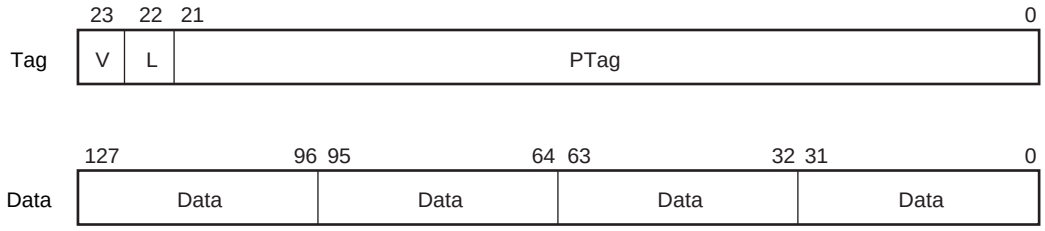
Figure 7-3 shows the format of a 4-word (16-byte) I-cache line.

Figure 7-3. Instruction Cache Line Format

(a) VR4121, VR4122, VR4181



(b) VR4131



V : Valid bit (line status)
L : Lock bit (line lock status)
Ptag : Physical tag (bits 31 to 10 of physical address)
Data : Cache data

Remarks 1. In the VR4181A, the data field has 256 bits since the line size is 8 words (32 bytes), though the tag format is the same as that of the VR4121, VR4122, and VR4181.

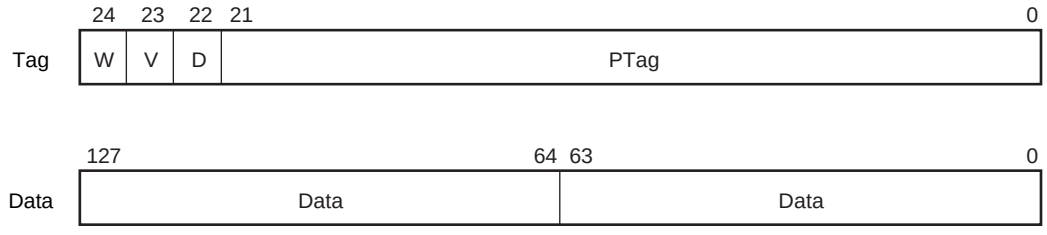
2. When the line size is specified as 8 words (32 bytes) in the VR4122 or VR4131, the data field becomes 256 bits wide.

7.2.2 Data cache line

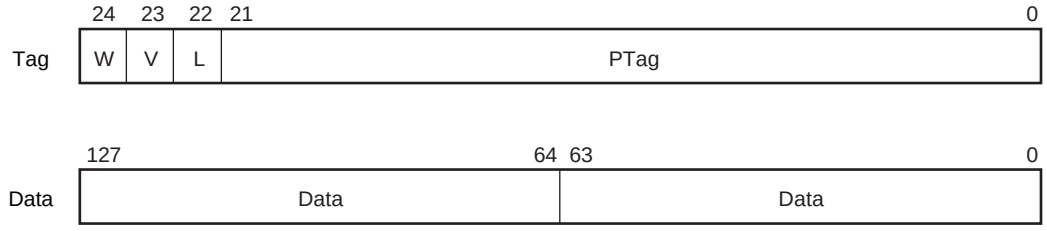
Figure 7-4 shows the format of a 4-word (16-byte) D-cache line.

Figure 7-4. Data Cache Line Format

(a) VR4121, VR4122, VR4181



(b) VR4131



W : Write-back bit (set if cache line has been written)
V : Valid bit (line status)
D : Dirty bit (write status)
L : Lock bit (line lock status)
Ptag : Physical tag (bits 31 to 10 of physical address)
Data : D-cache data

Remarks 1. In the VR4181A, the data field has 256 bits since the line size is 8 words (32 bytes), though the tag format is the same as that of the VR4121, VR4122, and VR4181.

2. When the line size is specified as 8 words (32 bytes) in the VR4131, the data field becomes 256 bits wide.

7.2.3 Placement of cache data

The cache data of the VR4121, VR4122, VR4181, and VR4181A are directly mapped; on the other hand those of the VR4131 are mapped in 2-way set associative format.

(1) Direct mapping

In this format, a cache is dealt with one block of memory space, and cache lines are placed linearly.

(2) 2-way set associative

In this format, the memory space of a cache is divided into two blocks (ways), and two cache lines are placed in the same index (of different ways).

7.3 Cache Operations

As described earlier, caches provide fast temporary data storage, and they make the speedup of memory accesses transparent to the user. In general, the CPU core accesses cache-resident instructions or data through the following procedure:

1. The CPU core, through the on-chip cache controller, attempts to access the next instruction or data in the appropriate cache.
2. The cache controller checks to see if this instruction or data is present in the cache.
   - If the instruction/data is present, the CPU core retrieves it. This is called a cache hit.
   - If the instruction/data is not present in the cache, the cache controller must retrieve it from memory. This is called a cache miss.
3. The CPU core retrieves the instruction/data from the cache and operation continues.

It is possible for the same data to be in two places simultaneously: main memory and cache. This data is kept consistent through the use of a writeback methodology; that is, modified data is not written back to memory until the cache line is to be replaced.

7.3.1 Cache data coherency

The CPU core of the VR4100 Series manages its data cache by using a writeback policy; that is, it stores write data into the cache, instead of writing it directly to memory. Some time later this data is independently written into memory. In the VR4100 Series implementation, a modified cache line is not written back to memory until the cache line is to be replaced.

When the CPU core writes a cache line back to memory, it does not ordinarily retain a copy of the cache line, and the state of the cache line is changed to invalid.

Remark Contrary to the writeback, the write-through cache policy stores write data into the memory and cache simultaneously.

(1) VR4121, VR4122, VR4181, and VR4181A

On a store miss writeback, data tag is checked and data is transferred to the write buffer. If an error is detected in the data field, the writeback is not terminated; the erroneous data is still written out to main memory. If an error is detected in the tag field, the writeback bus cycle is not issued.

The cache data may not be checked during CACHE operation.

(2) VR4131

On a store miss writeback, data tag is checked, a refill request is issued, and data is transferred to the write buffer. The writeback is performed after the refill is completed.

7.3.2 Replacement of cache line

When a cache miss occurs or when the Fill operation (for instruction cache only) or the Fetch_and_Lock operation (for VR4131 only) of CACHE instruction is executed, one of the cache lines is overwritten with data that is read from main memory. Such an overwriting is called replacement of a cache line.

The on-chip caches of the VR4131 are 2-way set associative memory where two cache lines are placed to one index. When a cache miss occurs, the way to be replaced is determined by the LRU (Least recently used) algorithm. It is indicated in the TagLo register of the CP0.

The on-chip caches of the VR4131 also have the line lock function. If a line is set locked on its placement, it will not be replaced even when a cache miss occurs. Cache line locking is set or cancelled with CACHE instruction, and locking status is indicated in the TagLo register of the CP0.

7.3.3 Accessing the caches

CACHE instruction is used to change cache line states or to write back cache data (for details, refer to **CHAPTER 9 CPU INSTRUCTION SET DETAILS**).

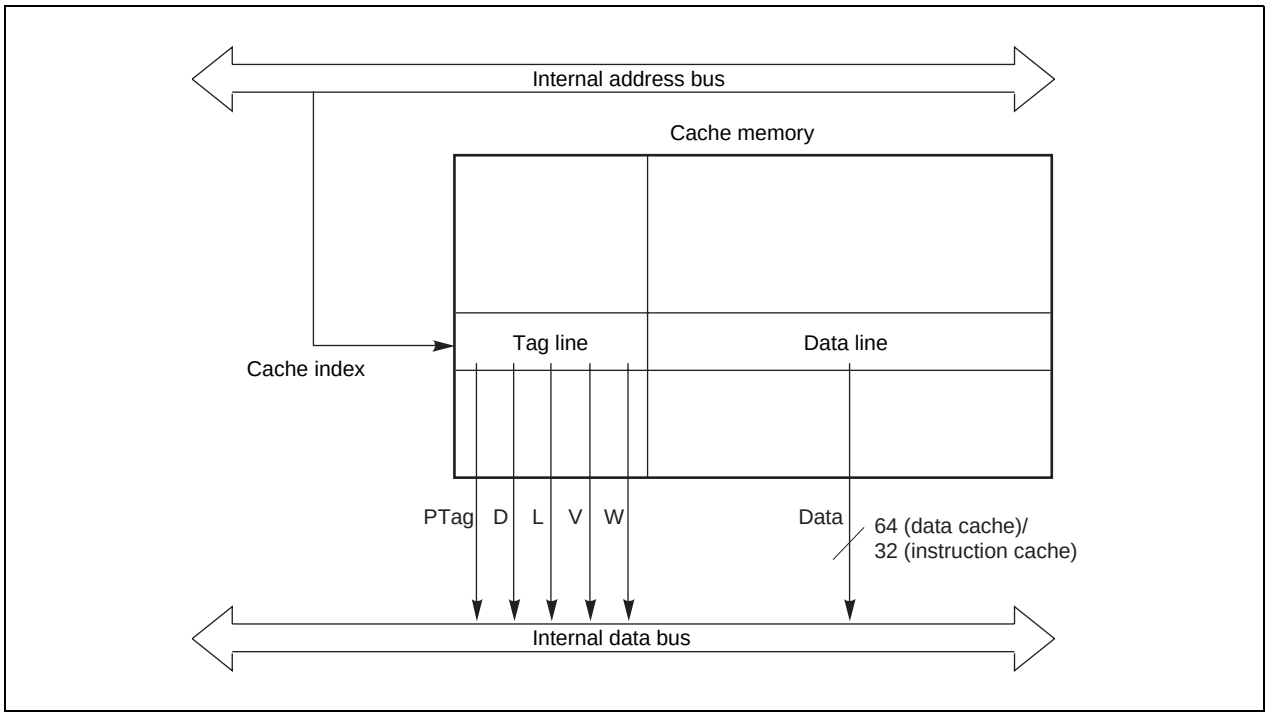
Some bits of the virtual address (VA) are used to index into the caches. The number of virtual address bits used to index the instruction and data caches depends on the cache size. In addition, bit 13 of the virtual address specifies the way to be accessed in the VR4131.

Table 7-1. Cache Size, Line Size, and Index

Processor	Cache	Cache size	Line size	Index
VR4121	Instruction	16 KB	4 words	VA(13:4)
	Data	8 KB	4 words	VA(12:4)
VR4122	Instruction	32 KB	4 words or 8 words	VA(14:4)
	Data	16 KB	4 words	VA(13:4)
VR4131	Instruction	16 KB	4 words or 8 words	VA(12:4)
	Data	16 KB	4 words or 8 words	VA(12:4)
VR4181	Instruction	4 KB	4 words	VA(11:4)
	Data	4 KB	4 words	VA(11:4)
VR4181A	Instruction	8 KB	8 words	VA(12:5)
	Data	8 KB	8 words	VA(12:5)

Figure 7-5 shows index into caches and data output.

Figure 7-5. Cache Index and Data Output



7.4 Cache States

There are three cache line states that indicate validity and consistency with main memory of line data.

(1) Instruction cache

The instruction cache supports two cache states:

- Invalid: a cache line that does not contain valid information must be marked invalid, and cannot be used.
- Valid: a cache line that contains valid data.

(2) Data cache

The data cache supports three cache states:

- Invalid: a cache line that does not contain valid information must be marked invalid, and cannot be used.
- Valid clean: a cache line that contains data that has not changed since it was loaded from memory.
- Valid dirty: a cache line containing data that has changed since it was loaded from memory.

The state of a valid cache line may be modified when the processor executes some operations of CACHE instruction. CACHE instruction and its operations are described in **CHAPTER 9 CPU INSTRUCTION SET DETAILS**.

7.4.1 Cache state transition diagrams

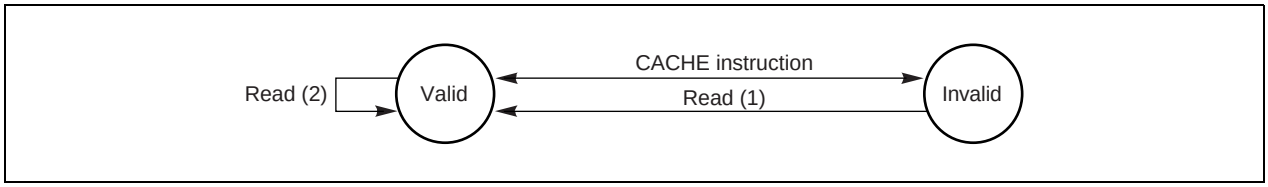
The following section describes the cache state diagrams for the data and instruction cache lines. These state diagrams do not cover the initial state of the system, since the initial state is system-dependent.

(1) Instruction cache state transition

The following diagram illustrates the instruction cache state transition sequence.

- Read (1) indicates a read operation from main memory to cache, inducing a cache state transition.
- Read (2) indicates a read operation from cache to the CPU core, which induces no cache state transition.

Figure 7-6. Instruction Cache State Diagram

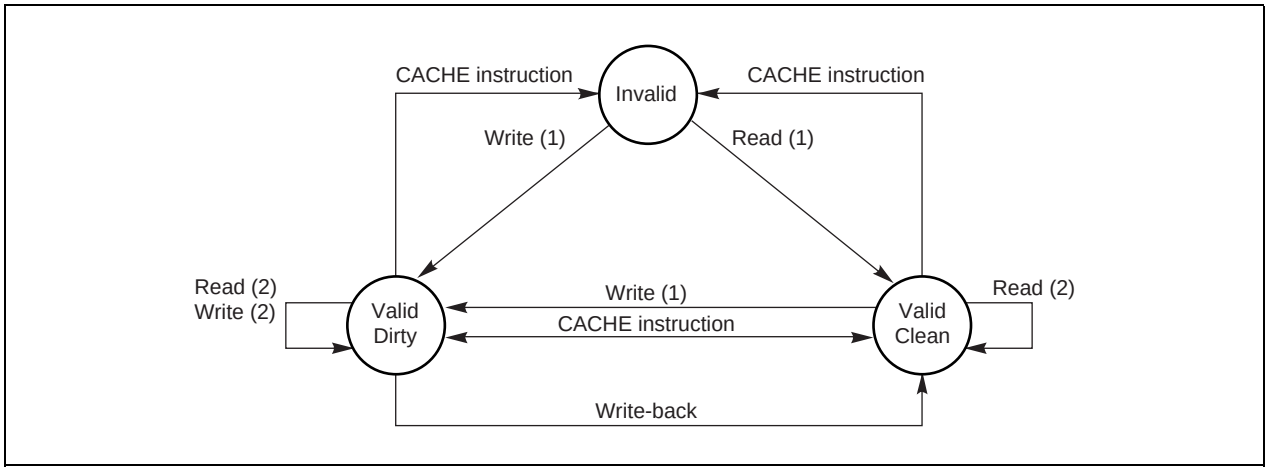


(2) Data cache state transition

The following diagram illustrates the data cache state transition sequence. A load or store operation may include one or more of the atomic read and/or write operations shown in the state diagram below, which may cause cache state transitions.

- Read (1) indicates a read operation from main memory to cache, inducing a cache state transition.
- Write (1) indicates a write operation from CPU core to cache, inducing a cache state transition.
- Read (2) indicates a read operation from cache to the CPU core, which induces no cache state transition.
- Write (2) indicates a write operation from CPU core to cache, which induces no cache state transition.

Figure 7-7. Data Cache State Diagram



7.5 Cache Access Flow

Figures 7-8 to 7-23 show operation flows for various cache accesses.

Figure 7-8. Flow on Instruction Fetch

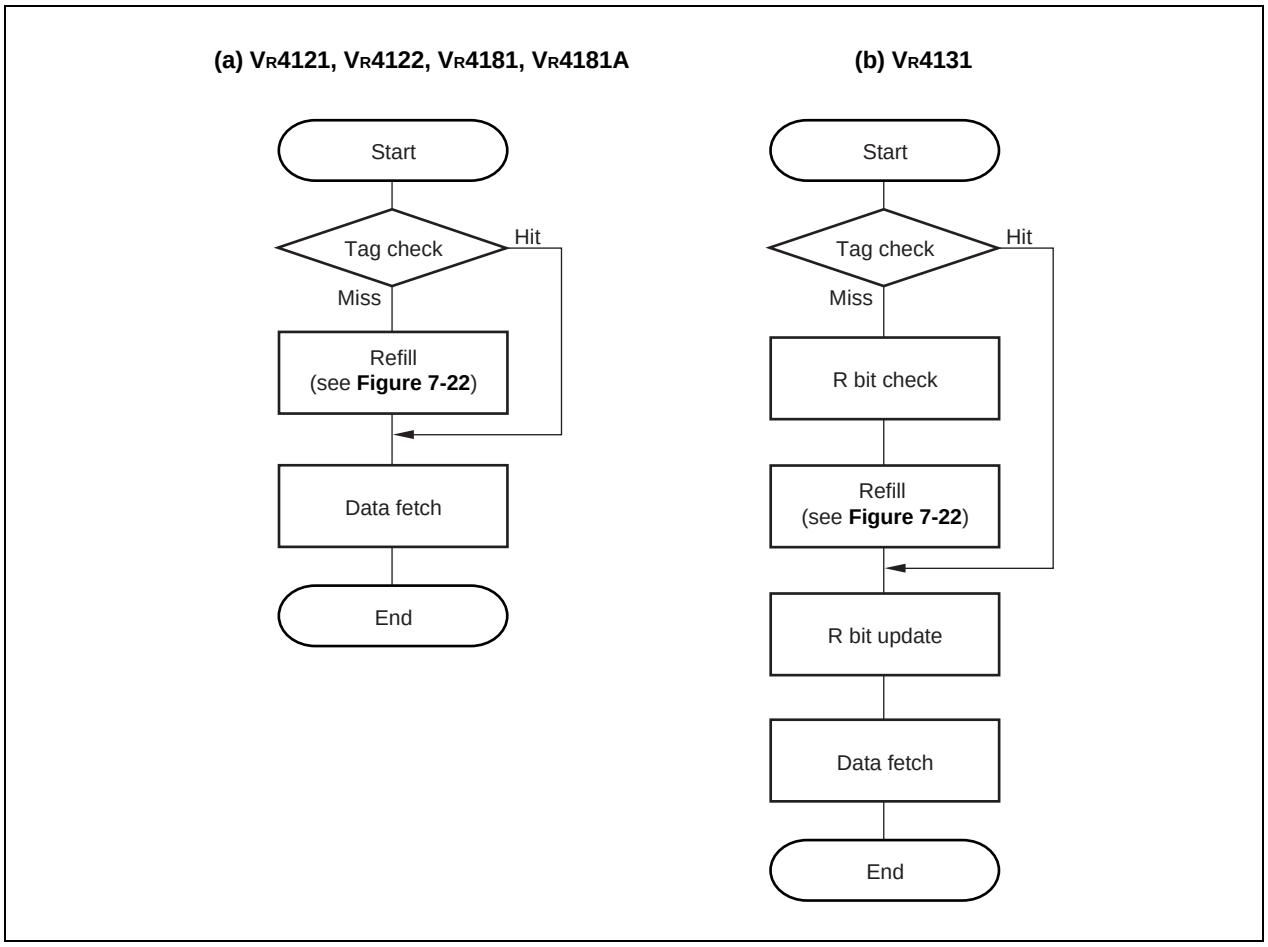


Figure 7-9. Flow on Load Operations

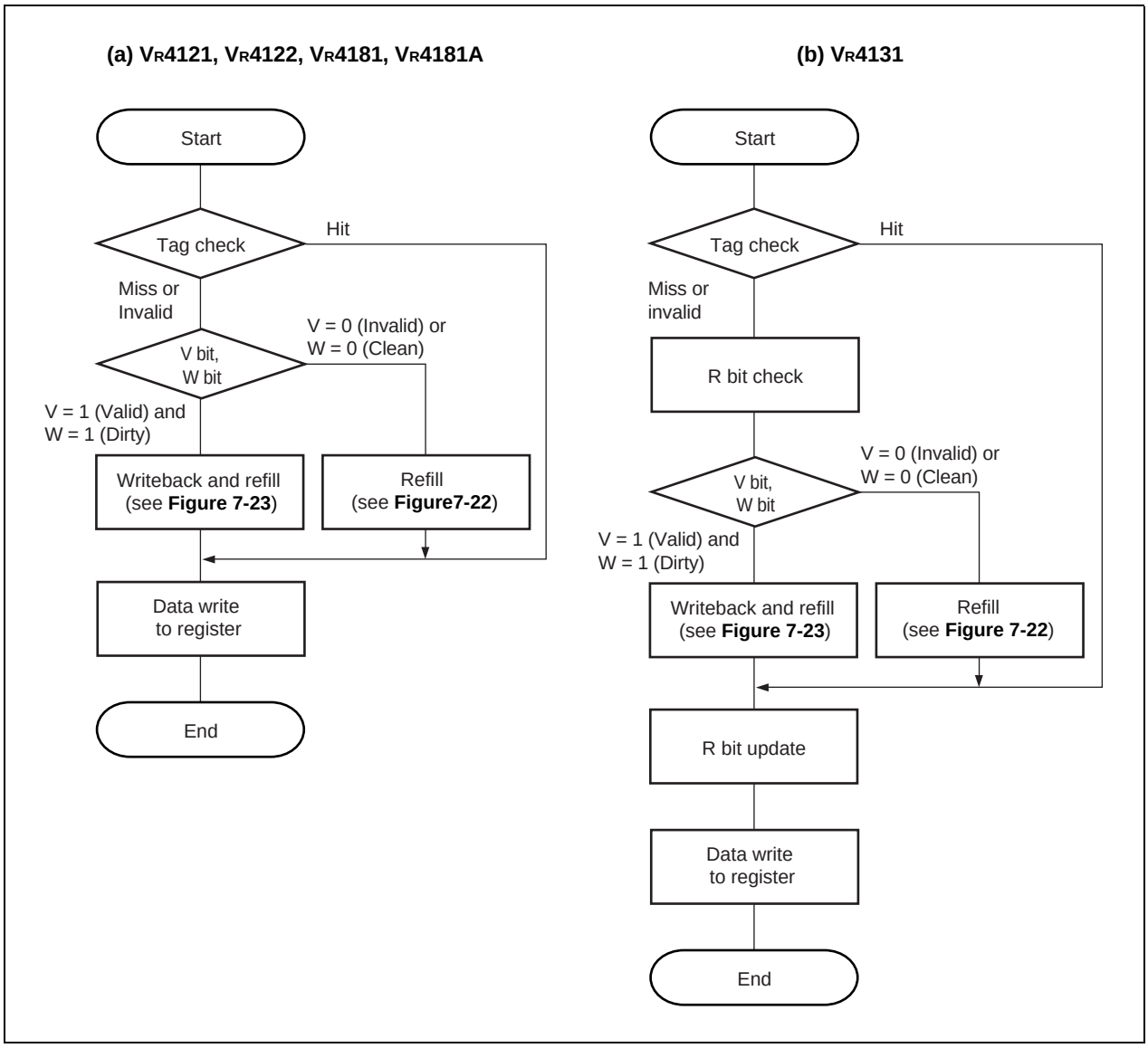


Figure 7-10. Flow on Store Operations

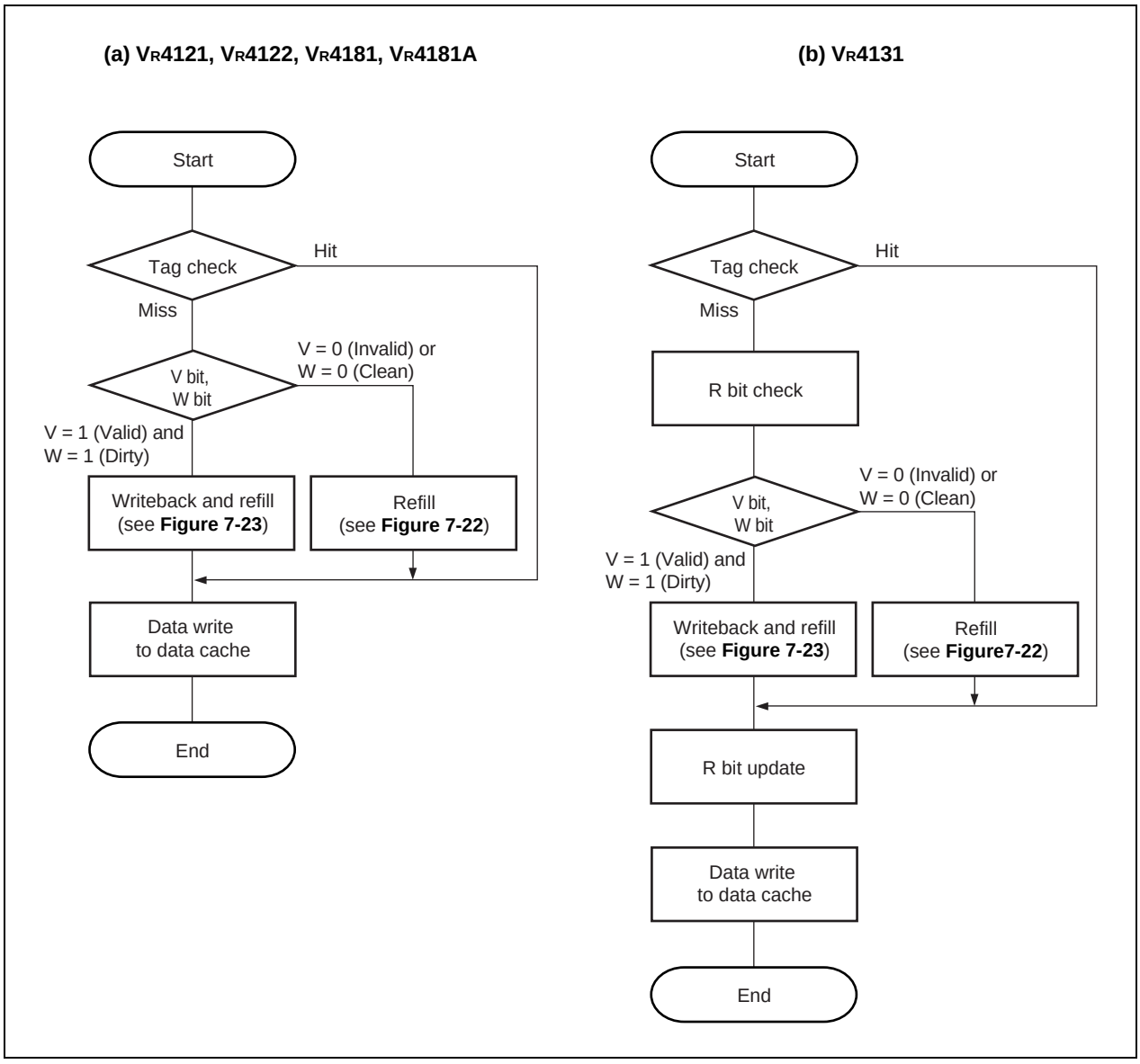


Figure 7-11. Flow on Index_Invalidate Operations

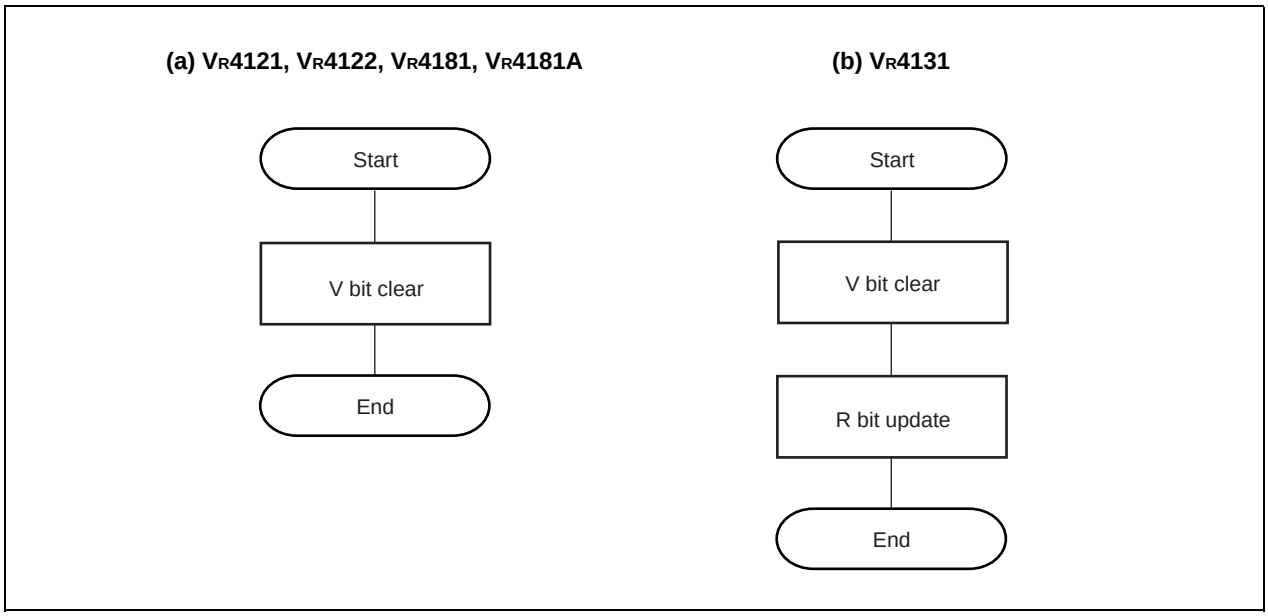


Figure 7-12. Flow on Index_Writeback_Invalidate Operations

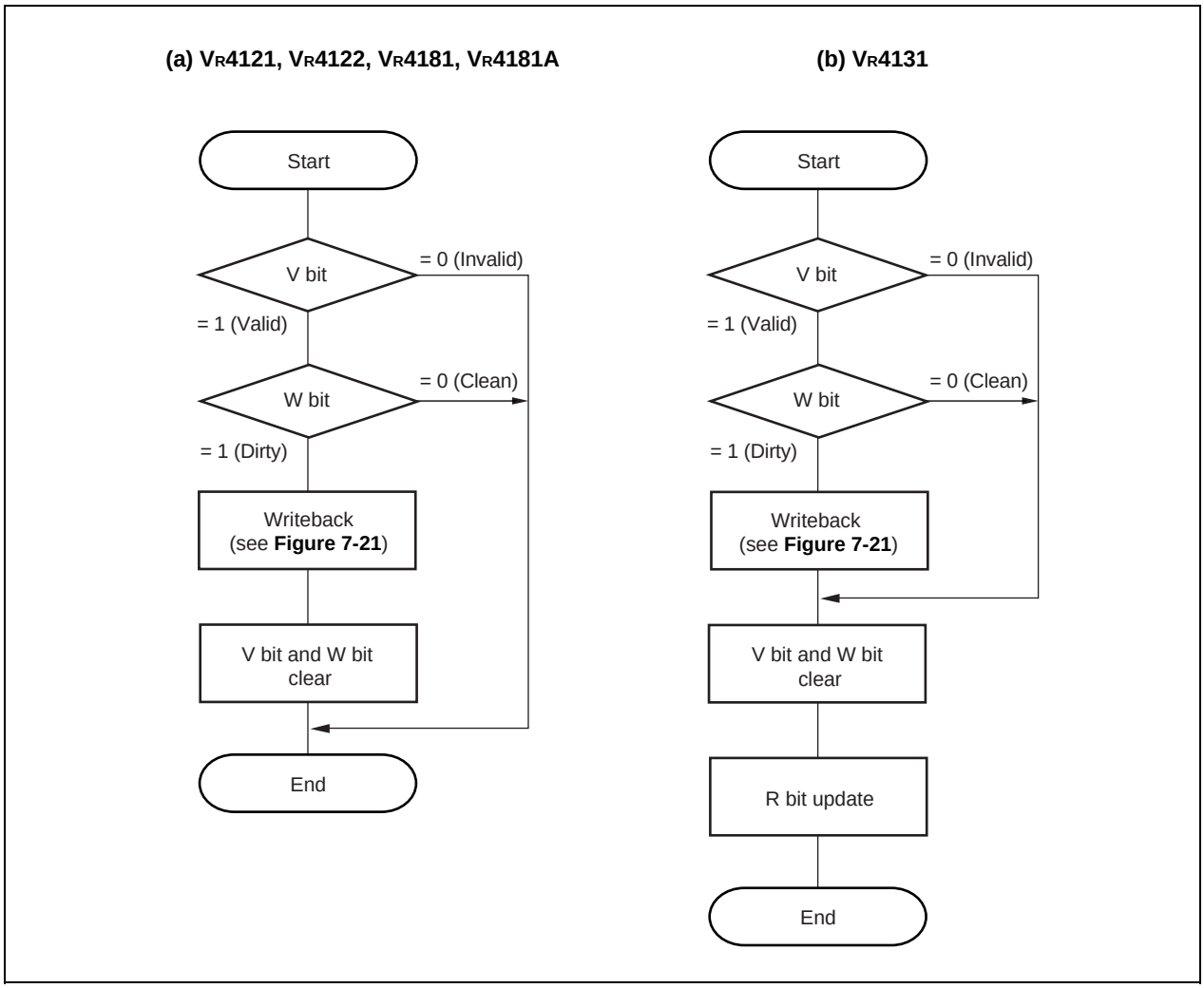


Figure 7-13. Flow on Index_Load_Tag Operations

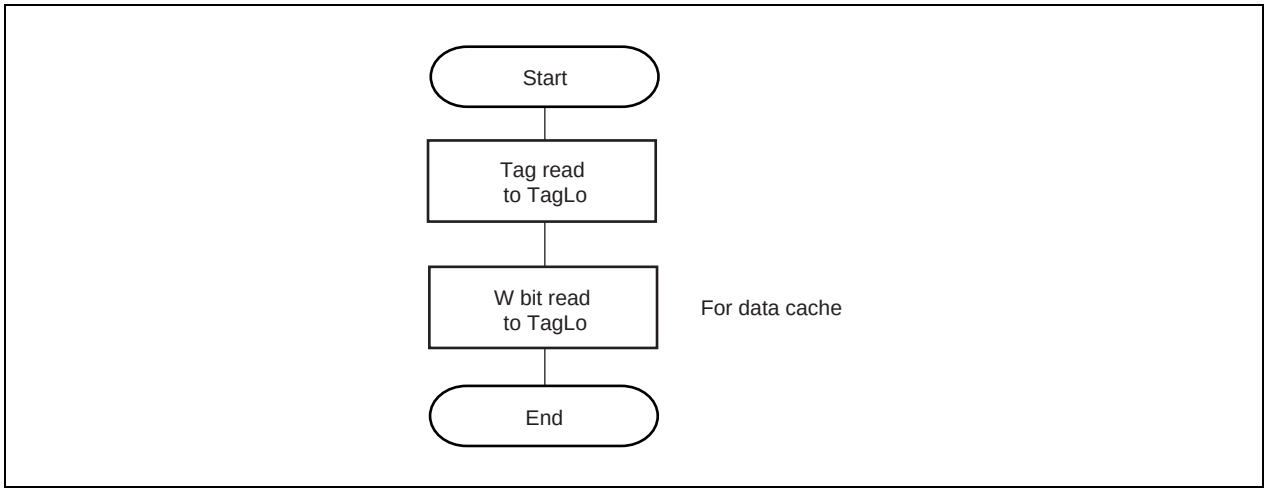


Figure 7-14. Flow on Index_Store_Tag Operations

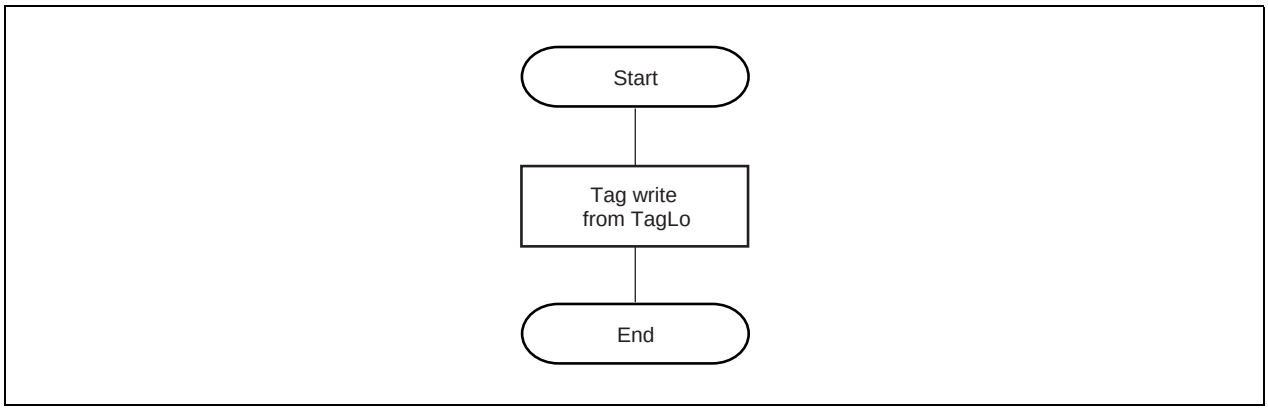


Figure 7-15. Flow on Create_Dirty Operations

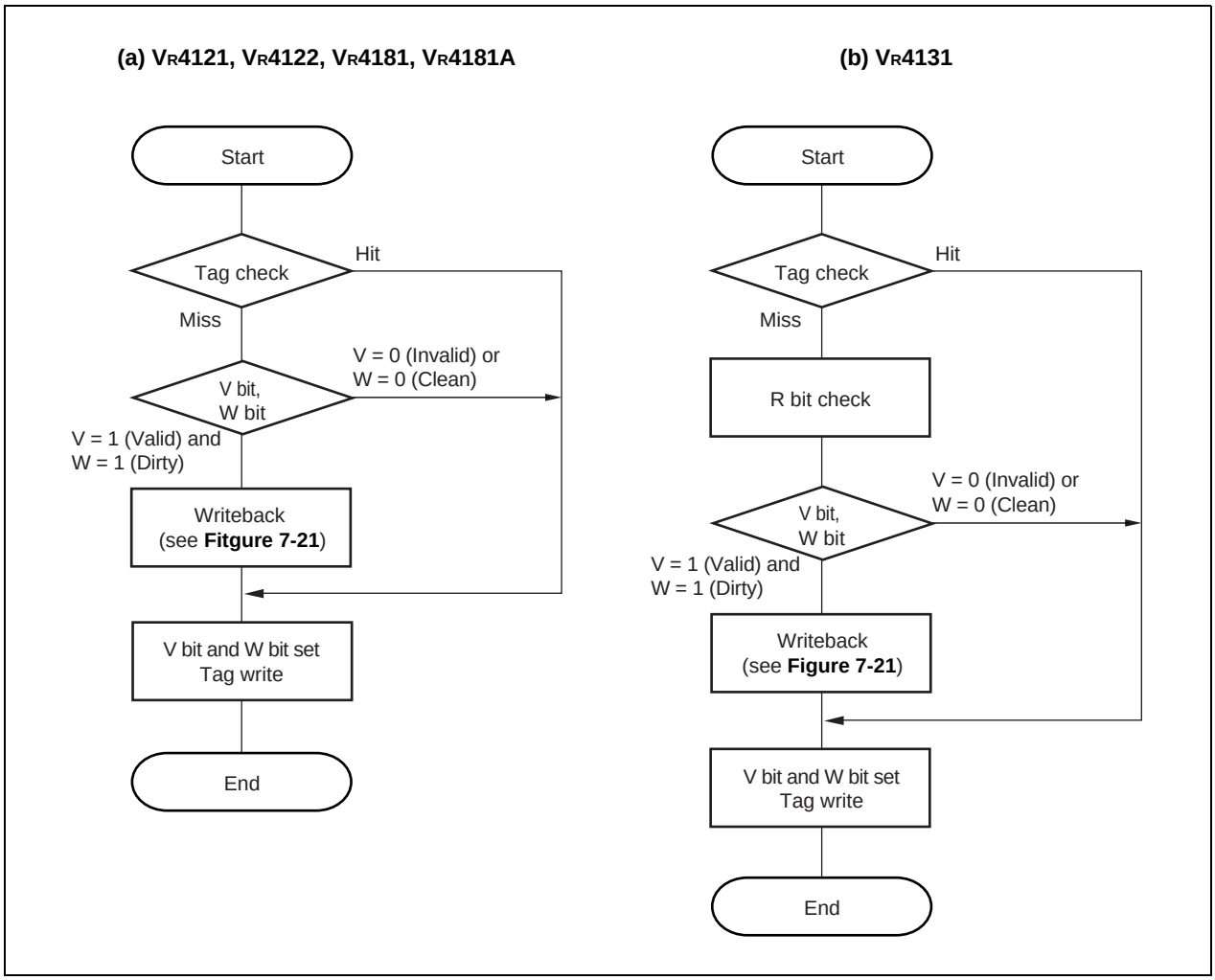


Figure 7-16. Flow on Hit_Invalidate Operations

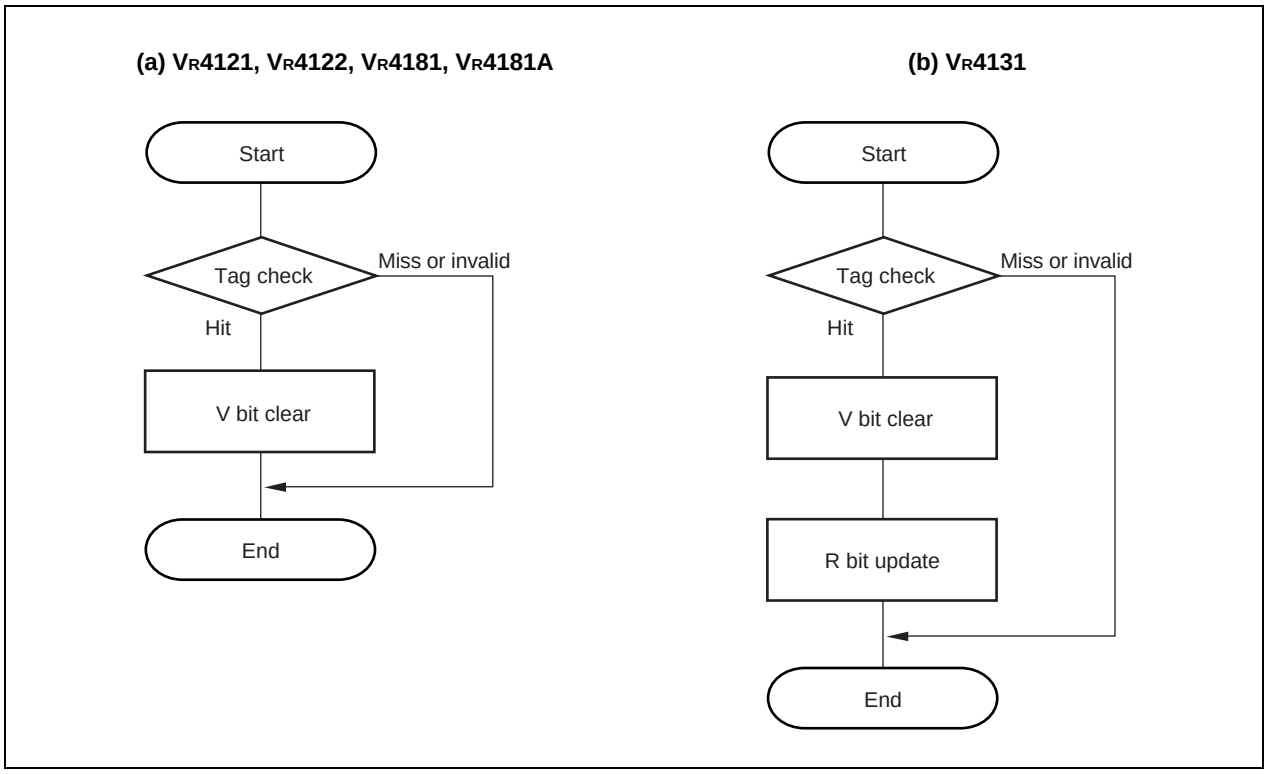


Figure 7-17. Flow on Hit_Writeback_Invalidate Operations

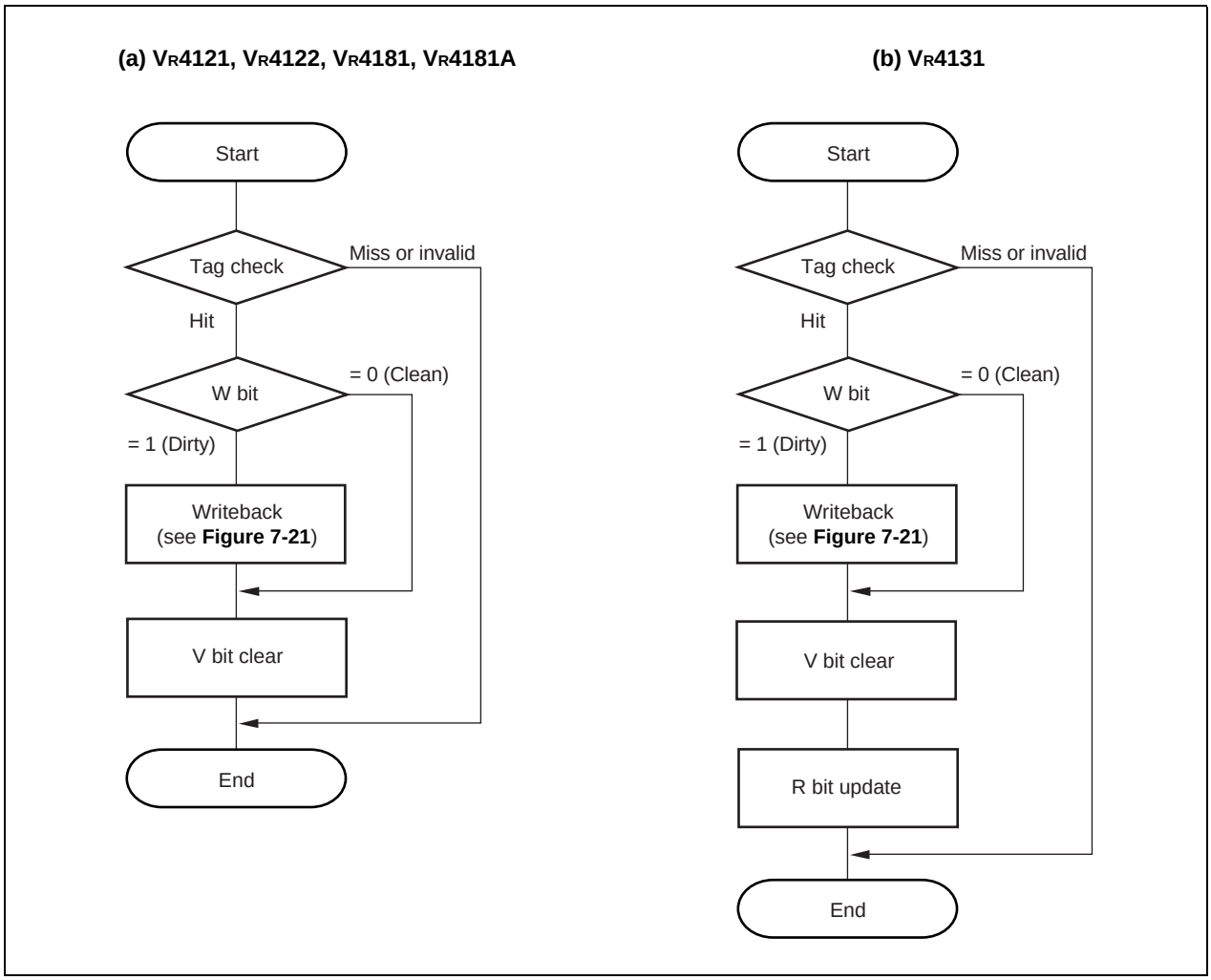


Figure 7-18. Flow on Fill Operations

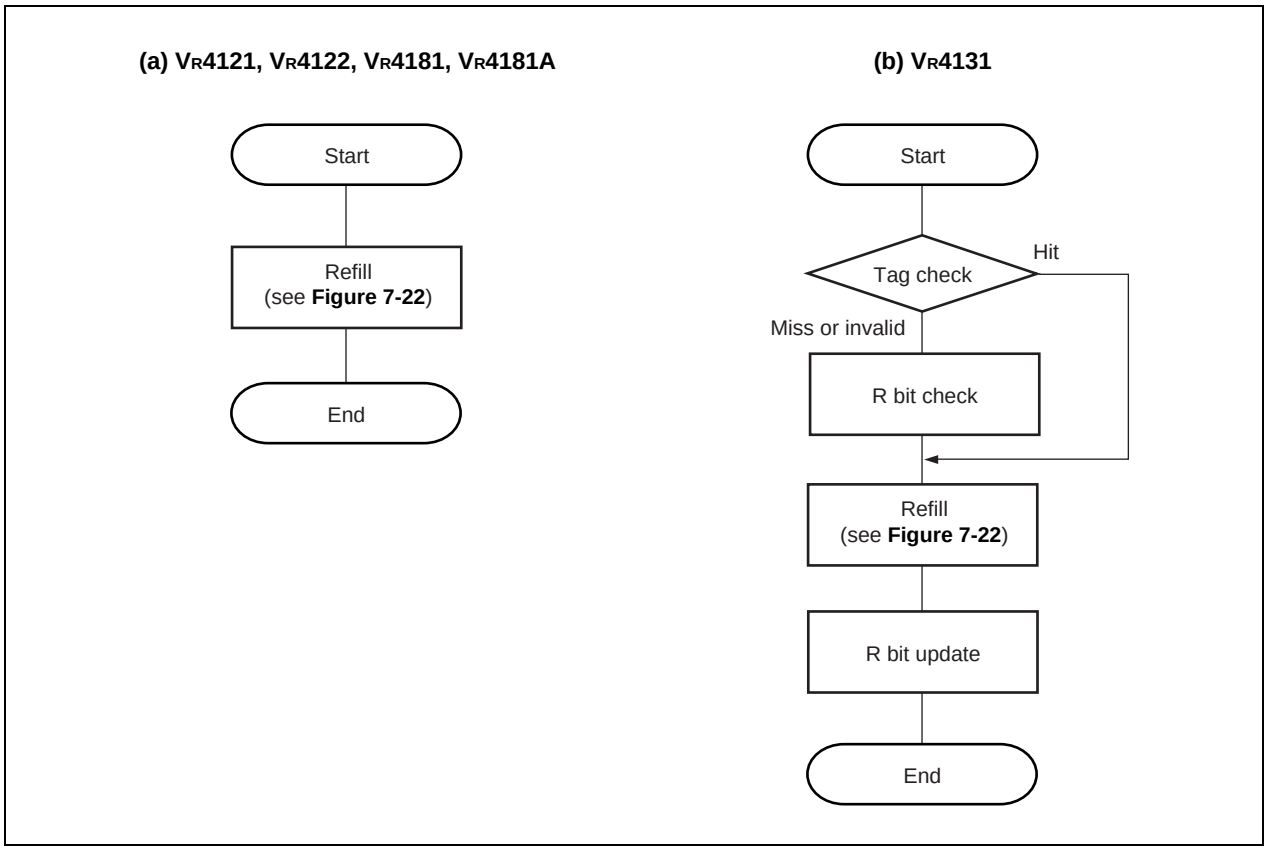


Figure 7-19. Flow on Hit_Writeback Operations

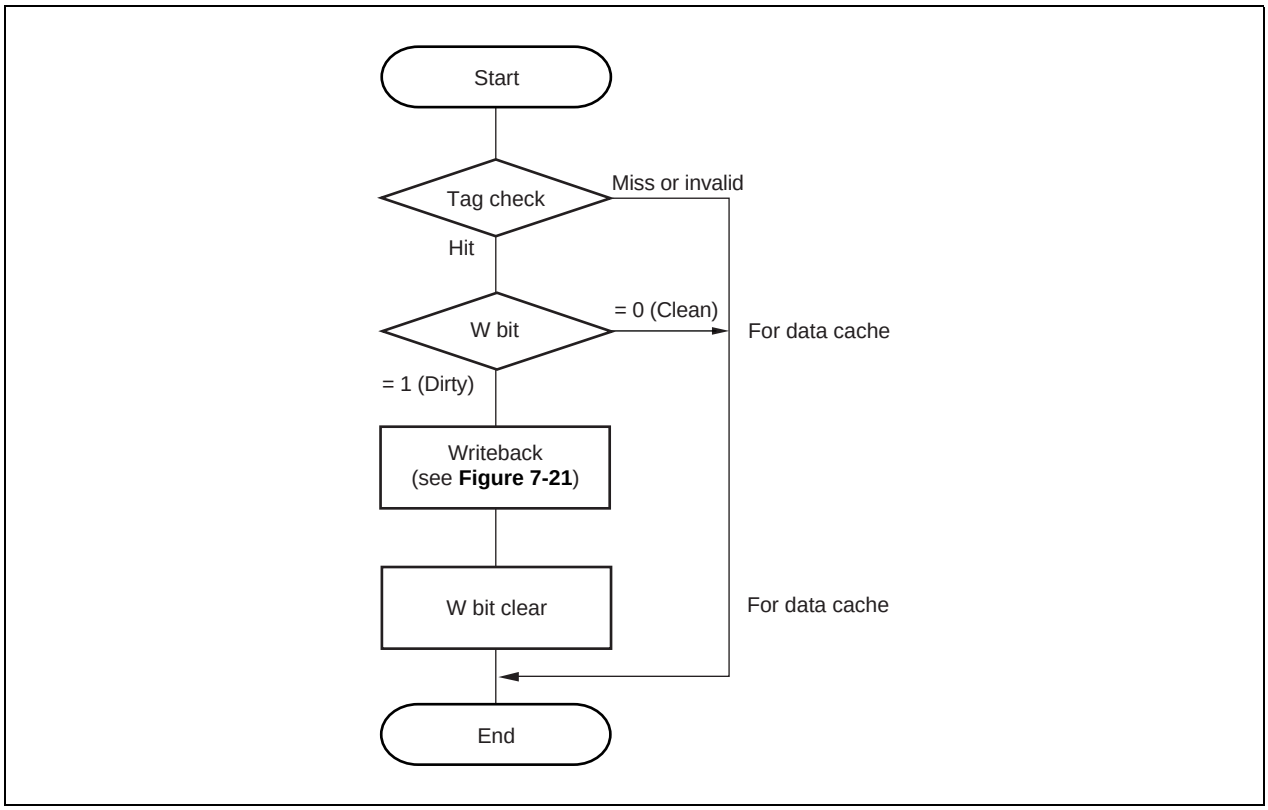


Figure 7-20. Flow on Fetch_and_Lock Operations (VR4131 only)

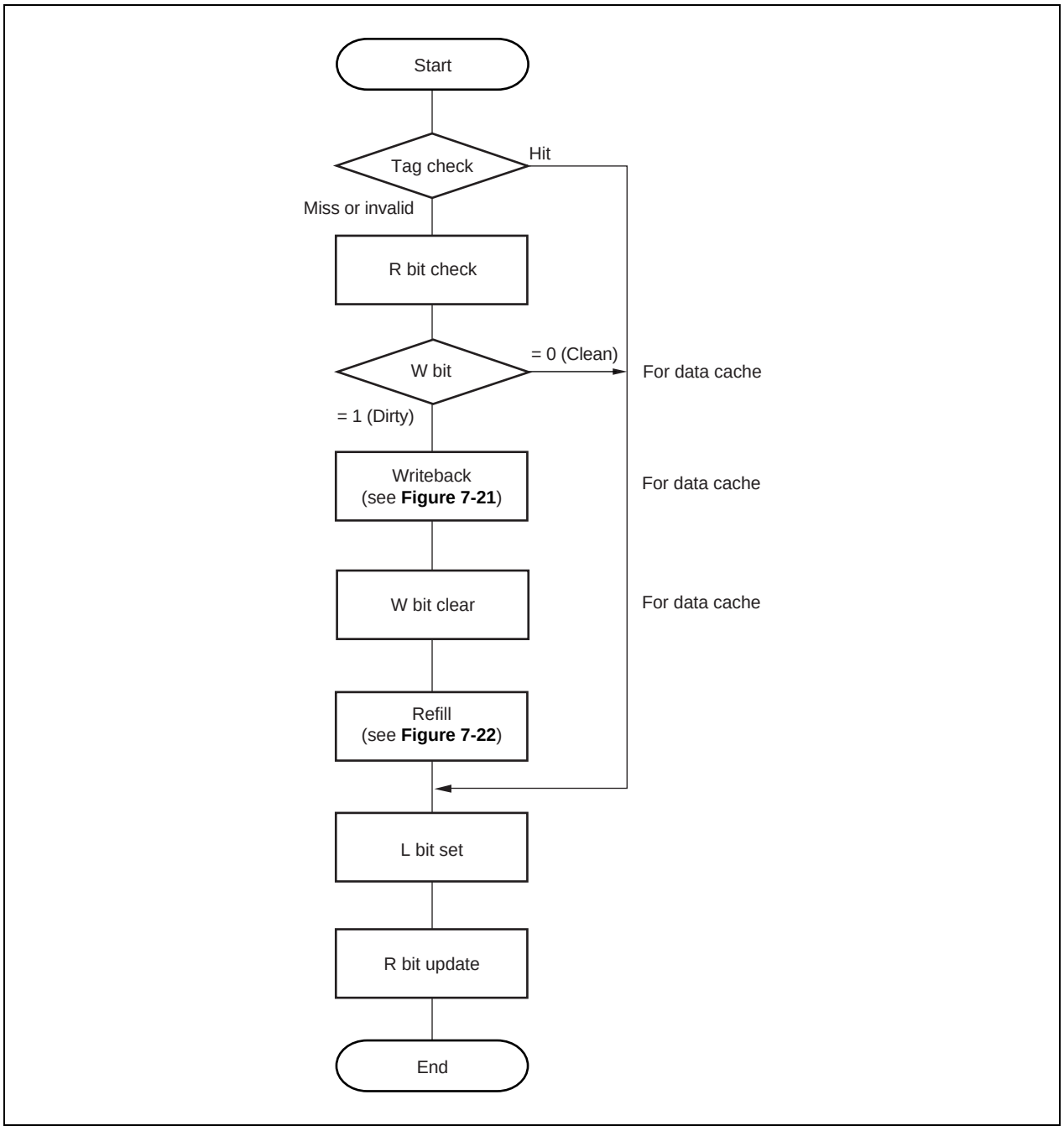


Figure 7-21. Writeback Flow

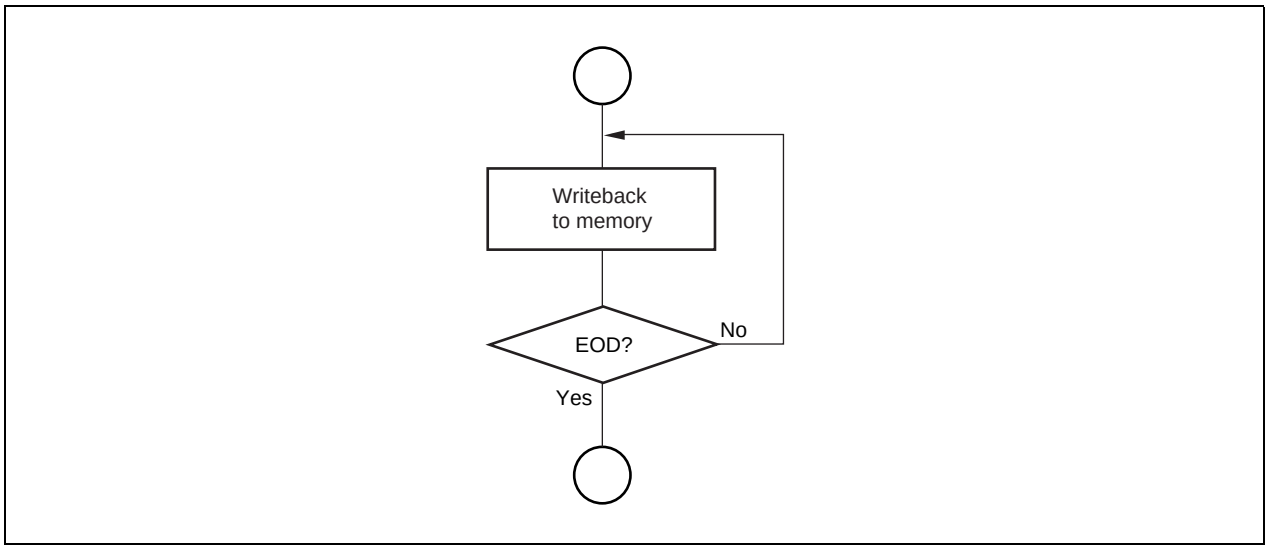


Figure 7-22. Refill Flow

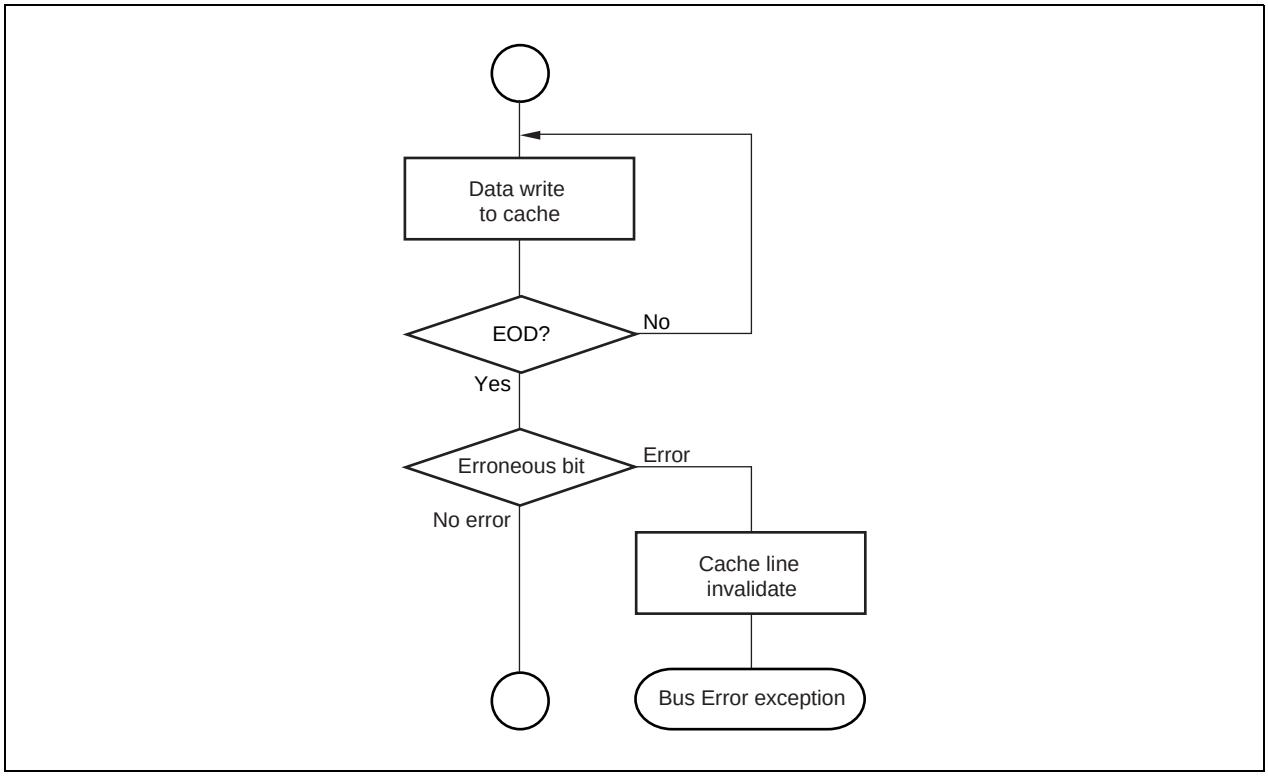
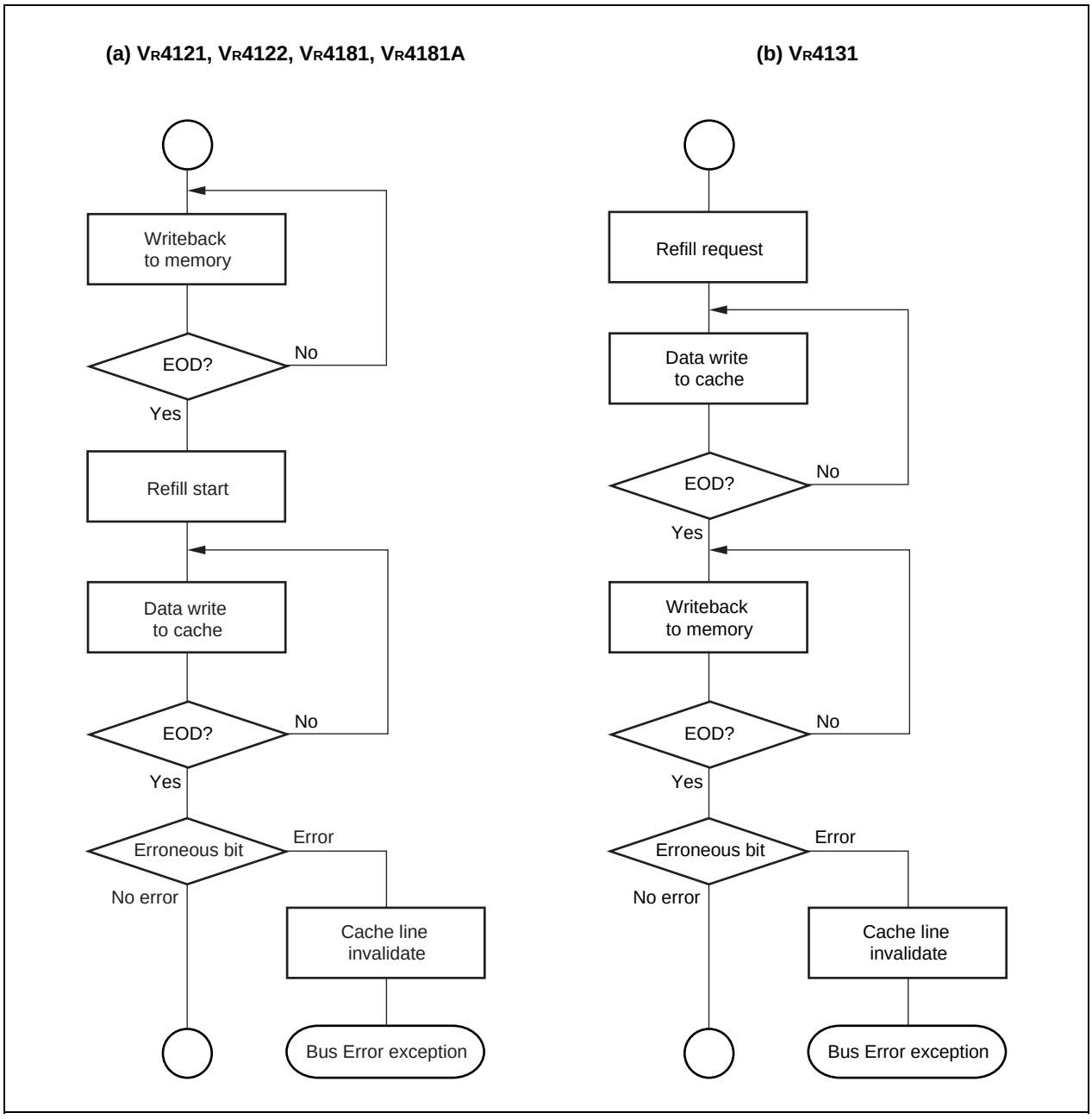


Figure 7-23. Writeback & Refill Flow



7.6 Manipulation of the Caches by an External Agent

The VR4100 Series does not provide any mechanisms for an external agent to examine and manipulate the state and contents of the caches.

7.7 Initialization of the Caches

The caches of the VR4100 Series also need an initialization on reset or such cases. For procedures and program examples of initialization, refer to **VR Series Programming Guide Application Note**.

CHAPTER 8 CPU CORE INTERRUPTS

Four types of interrupt are available on the CPU core of the VR4100 Series. These are:

- one non-maskable interrupt, NMI
- five ordinary interrupts
- two software interrupts
- one timer interrupt

For the interrupt request input to the CPU core from on-chip peripheral units, see **Hardware User's Manual** of each product.

8.1 Types of Interrupt Request

8.1.1 Non-maskable interrupt (NMI)

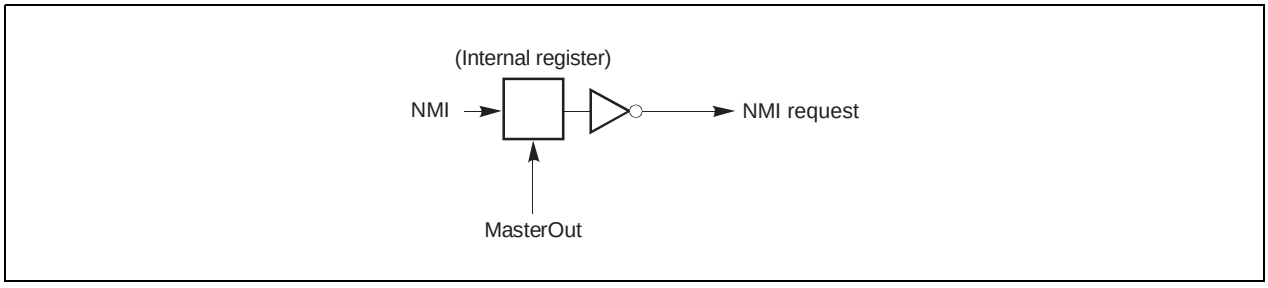
The non-maskable interrupt is acknowledged by asserting the NMI signal (internal), forcing the processor to branch to the Reset Exception vector. This signal is latched into an internal register at the rising edge of MasterOut (internal), as shown in Figure 8-1.

NMI only takes effect when the processor pipeline is running.

This interrupt cannot be masked.

Figure 8-1 shows the internal service of the NMI signal. The NMI signal is latched into an internal register by the rising edge of MasterOut. The latched signal is inverted to be transferred to inside the device as an NMI request.

Figure 8-1. Non-maskable Interrupt Signal



8.1.2 Ordinary interrupts

Ordinary interrupts are acknowledged by asserting the Int(4:0) signals (internal). However, Int3 occurs in the VR4121 and VR4181A only, and Int4 in the VR4181A only.

This interrupt request can be masked with the IM (6:2), IE, EXL, and ERL fields of the Status register.

8.1.3 Software interrupts generated in CPU core

Software interrupts generated in the CPU core use bits 1 and 0 of the IP (interrupt pending) field in the Cause register. These may be written by software, but there is no hardware mechanism to set or clear these bits.

After the processing of a software interrupt exception, corresponding bit of the IP field in the Cause register must be cleared before enabling multiple interrupts or until the operation returns to normal routine.

This interrupt request is maskable through the IM (1:0), IE, EXL, and ERL fields of the Status register.

8.1.4 Timer interrupt

The timer interrupt uses bit 7 of the IP (interrupt pending) field of the Cause register. This bit is set automatically whenever the value of the Count register equals the value of the Compare register, and an interrupt request is acknowledged.

This interrupt is maskable through IM7, IE, EXL, and ERL fields of the Status register.

8.2 Acknowledging Interrupts

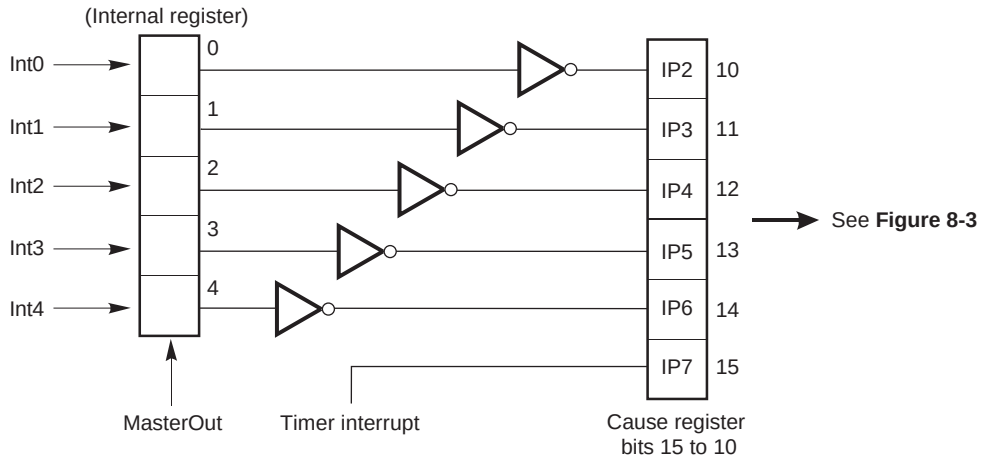
8.2.1 Detecting hardware interrupts

Figure 8-2 shows how the hardware interrupts are readable through the Cause register.

- The timer interrupt signal of the CPU core is directly readable as bit 15 (IP7) of the Cause register.
- The Int(4:0) signals are directly readable as bits 14 to 10 (IP(6:2)) of the Cause register.

IP(1:0) of the Cause register are used for software interrupt requests. There is no hardware mechanism for setting or clearing the software interrupts.

Figure 8-2. Hardware Interrupt Signals



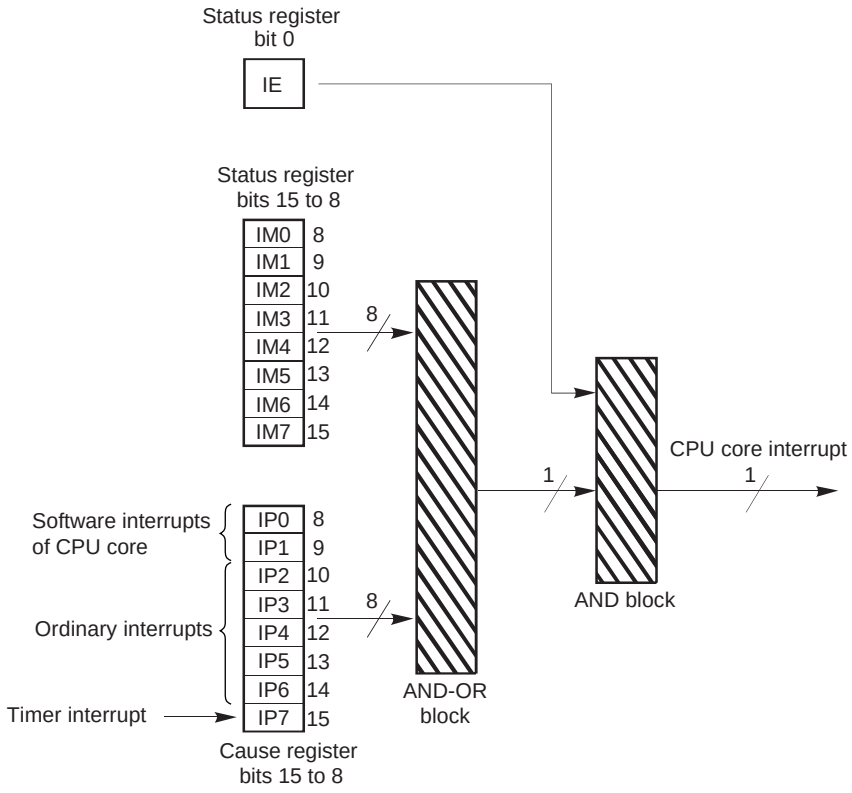
Remark Int3 occurs in the VR4121 and VR4181A only, and Int4 in the VR4181A only.

8.2.2 Masking interrupt signals

Figure 8-3 shows the masking of the CPU core interrupt signals.

- Cause register bits 15 to 8 (IP(7:0)) are AND-ORed with Status register interrupt mask bits 15 to 8 (IM(7:0)) to mask individual interrupts.
- Status register bit 0 is a global Interrupt Enable (IE) bit. It is ANDed with the output of the AND-OR logic to produce the CPU core interrupt signal. The EXL bit in the Status register also enables these interrupts.

Figure 8-3. Masking of the Interrupt Request Signals



Bit	Function	Setting
IE	Whole interrupts enable	1 : Enable 0 : Disable
IM(7:0)	Interrupt mask	Each bit 1 : Enable 0 : Disable
IP(7:0)	Interrupt request	Each bit 1 : Pending 0 : Not pending

CHAPTER 9 CPU INSTRUCTION SET DETAILS

This chapter provides a detailed description of the operation of each VR4100 Series instruction in both 32- and 64-bit modes. The instructions are listed in alphabetical order.

9.1 Instruction Notation Conventions

In this chapter, all variable subfields in an instruction format (such as *rs*, *rt*, *immediate*, etc.) are shown in lowercase names.

For the sake of clarity, we sometimes use an alias for a variable subfield in the formats of specific instructions. For example, we use *rs* = *base* in the format for load and store instructions. Such an alias is always lower case, since it refers to a variable subfield.

Figures with the actual bit encoding for all the mnemonics are located at the end of this chapter (**9.4 CPU Instruction Opcode Bit Encoding**), and the bit encoding also accompanies each instruction.

In the instruction descriptions that follow, the Operation section describes the operation performed by each instruction using a high-level language notation. The VR4100 Series can operate as either a 32- or 64-bit microprocessor and the operation for both modes is included with the instruction description.

Special symbols used in the notation are described in Table 9-1.

Table 9-1. CPU Instruction Operation Notations

Symbol	Meaning
<-	Assignment.
||	Bit string concatenation.
x^y	Replication of bit value *x* into a *y*-bit string. *x* is always a single-bit value.
$x_{y...z}$	Selection of bits *y* through *z* of bit string *x*. Little-endian bit notation is always used. If *y* is less than *z*, this expression is an empty (zero length) bit string.
+	2's complement or floating-point addition.
-	2's complement or floating-point subtraction.
*	2's complement or floating-point multiplication.
div	2's complement integer division.
mod	2's complement modulo.
/	Floating-point division.
<	2's complement less than comparison.
and	Bit-wise logical AND.
or	Bit-wise logical OR.
xor	Bit-wise logical XOR.
nor	Bit-wise logical NOR.
GPR [*x*]	General-Register *x*. The content of GPR [0] is always zero. Attempts to alter the content of GPR [0] have no effect.
CPR [*z, x*]	Coprocessor unit *z*, general register *x*.
CCR [*z, x*]	Coprocessor unit *z*, control register *x*.
COC [*z*]	Coprocessor unit *z* condition signal.
BigEndianMem	Big-endian mode as configured at reset (0 → Little, 1 → Big). Specifies the endianness of the memory interface (see **Table 9-2**), and the endianness of Kernel and Supervisor mode execution. However, this value is always 0 in the VR4121, VR4122, VR4181, and VR4181A since they support the little endian order only.
ReverseEndian	Signal to reverse the endianness of load and store instructions. This feature is available in User mode only, and is effected by setting the RE bit of the Status register. Thus, ReverseEndian may be computed as (SR_{25} and User mode). However, this value is always 0 since the VR4100 Series does not support the reverse of the endianness.
BigEndianCPU	The endianness for load and store instructions (0 → Little, 1 → Big). In User mode, this endianness may be reversed by setting SR_{25}. Thus, BigEndianCPU may be computed as BigEndianMem XOR ReverseEndian. However, this value is always 0 in the VR4121, VR4122, VR4181, and VR4181A since they support the little endian order only.
T + *i* :	Indicates the time steps between operations. Each of the statements within a time step are defined to be executed in sequential order (as modified by conditional and loop constructs). Operations which are marked T + *i* : are executed at instruction cycle *i* relative to the start of execution of the instruction. Thus, an instruction which starts at time *j* executes operations marked T + *i* : at time *i* + *j*. The interpretation of the order of execution between two instructions or two operations that execute at the same time should be pessimistic; the order is not defined.

The following examples illustrate the application of some of the instruction notation conventions:

Example #1:

GPR [rt] $\leftarrow$ immediate || 0^{16}

Sixteen zero bits are concatenated with an immediate value (typically 16 bits), and the 32-bit string (with the lower 16 bits set to zero) is assigned to General-purpose register *rt*.

Example #2:

$(immediate_{15})^{16}$ || $immediate_{15...0}$

Bit 15 (the sign bit) of an immediate value is extended for 16 bit positions, and the result is concatenated with bits 15 through 0 of the immediate value to form a 32-bit sign extended value.

9.2 Notes on Using CPU Instructions

9.2.1 Load and Store instructions

In the VR4100 Series implementation, the instruction immediately following a Load may use the loaded contents of the register. In such cases, the hardware interlocks, requiring additional real cycles, so scheduling load delay slots is still desirable, although not required for functional code.

In the Load and Store descriptions, the functions listed in Table 9-2 are used to summarize the handling of virtual addresses and physical memory.

Table 9-2. Load and Store Common Functions

Function	Meaning
Address Translation	Uses the TLB to find the physical address given the virtual address. The function fails and an exception is taken if the required translation is not present in the TLB.
Load Memory	Uses the cache and main memory to find the contents of the word containing the specified physical address. The low-order three bits of the address and the Access Type field indicate which of each of the four bytes within the data word need to be returned. If the cache is enabled for this access, the entire word is returned and loaded into the cache. If the specified data is short of word length, the data position to which the contents of the specified data is stored is determined considering the endian mode and reverse endian mode.
Store Memory	Uses the cache, write buffer, and main memory to store the word or part of word specified as data in the word containing the specified physical address. The low-order three bits of the address and the Access Type field indicate which of each of the four bytes within the data word should be stored. If the specified data is short of word length, the data position to which the contents of the specified data is stored is determined considering the endian mode and reverse endian mode.

As shown in Table 9-3, the Access Type field indicates the size of the data item to be loaded or stored. Regardless of access type or byte-numbering order (endianness), the address specifies the byte that has the smallest byte address in the addressed field. For a big-endian machine, this is the leftmost byte and contains the sign for a 2's complement number; for a little-endian machine, this is the rightmost byte.

Table 9-3. Access Type Specifications for Loads/Stores

Access type mnemonic	Value in internal command	Meaning
DOUBLEWORD	7	8 bytes (64 bits)
SEPTIBYTE	6	7 bytes (56 bits)
SEXTIBYTE	5	6 bytes (48 bits)
QUINTIBYTE	4	5 bytes (40 bits)
WORD	3	4 bytes (32 bits)
TRIPLEBYTE	2	3 bytes (24 bits)
HALFWORD	1	2 bytes (16 bits)
BYTE	0	1 byte (8 bits)

The bytes within the addressed doubleword that are used can be determined directly from the access type and the three low-order bits of the address.

9.2.2 Jump and Branch instructions

All Jump and Branch instructions have an architectural delay of exactly one instruction. That is, the instruction immediately following a Jump or Branch (that is, occupying the delay slot) is always executed while the target instruction is being fetched from storage. A delay slot may not itself be occupied by a Jump or Branch instruction; however, this error is not detected and the results of such an operation are undefined.

If an exception or interrupt prevents the completion of a legal instruction during a delay slot, the hardware sets the EPC register to point at the Jump or Branch instruction that precedes it. When the code is restarted, both the Jump or Branch instructions and the instruction in the delay slot are reexecuted.

Because Jump and Branch instructions may be restarted after exceptions or interrupts, they must be restartable. Therefore, when a Jump or Branch instruction stores a return link value, register *r31* (the register in which the link is stored) may not be used as a source register.

Since instructions must be word-aligned, a Jump Register or Jump and Link Register instruction must use a register which contains an address whose two low-order bits (low-order one bit in the 16-bit mode) are zero. If these low-order bits are not zero, an address exception will occur when the jump target instruction is subsequently fetched.

9.2.3 System control coprocessor (CP0) instructions

There are some special limitations imposed on operations involving CP0 that is incorporated within the CPU. Although Load and Store instructions to transfer data to/from coprocessors and to move control to/from coprocessor instructions are generally permitted by the MIPS architecture, CP0 is given a somewhat protected status since it has responsibility for exception handling and memory management. Therefore, the move to/from coprocessor instructions are the only valid mechanism for writing to and reading from the CP0 registers.

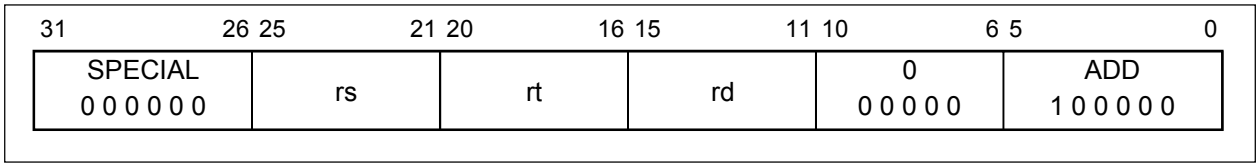
Several CP0 instructions are defined to directly read, write, and probe TLB entries and to modify the operating modes in preparation for returning to User mode or interrupt-enabled states.

9.3 CPU Instructions

This section describes the functions of CPU instructions in detail for both 32-bit address mode and 64-bit address mode.

The exception that may occur by executing each instruction is shown in the last of each instruction's description. For details of exceptions and their processes, see **CHAPTER 6 EXCEPTION PROCESSING**.

ADD Add ADD



Format:

ADD rd, rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*. In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

An overflow exception occurs if the carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

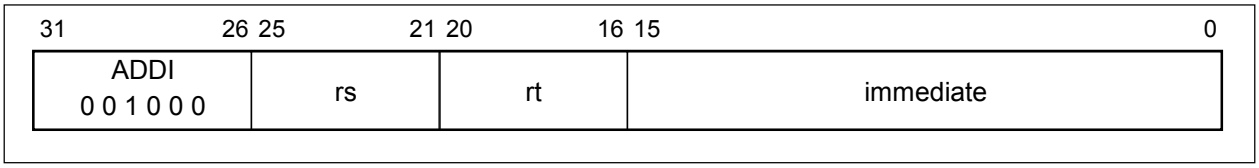
Operation:

```
32      T:   GPR [rd] ← GPR [rs] + GPR [rt]

64      T:   temp ← GPR [rs] + GPR [rt]
             GPR [rd] ← (temp31)^32 || temp31...0
```

Exceptions:

Integer overflow exception

ADDI Add Immediate ADDI



Format:

ADDI rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.* In 64-bit mode, the operand must be valid sign-extended, 32-bit values.

An overflow exception occurs if carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rt* is not modified when an integer overflow exception occurs.

Restrictions:

If the value of general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

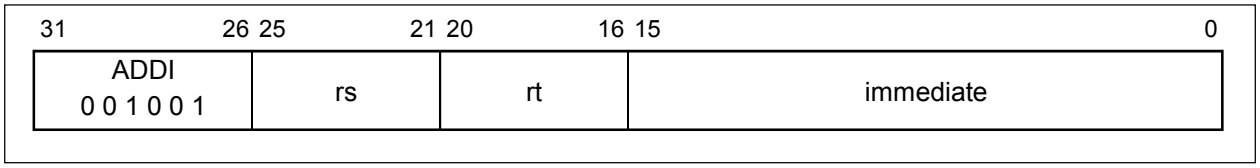
Operation:

32	T:	GPR [rt] ← GPR [rs] + $(immediate_{15})^{16}$ || $immediate_{15...0}$
64	T:	temp ← GPR [rs] + $(immediate_{15})^{48}$ || $immediate_{15...0}$ GPR [rt] ← $(temp_{31})^{32}$ || $temp_{31...0}$

Exceptions:

Integer overflow exception

ADDIU Add Immediate Unsigned ADDIU



Format:

ADDIU rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt*. No integer overflow exception occurs under any circumstances. In 64-bit mode, the operand must be valid sign-extended, 32-bit values.

The only difference between this instruction and the ADDI instruction is that ADDIU never causes an integer overflow exception.

Restrictions:

If the value of general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

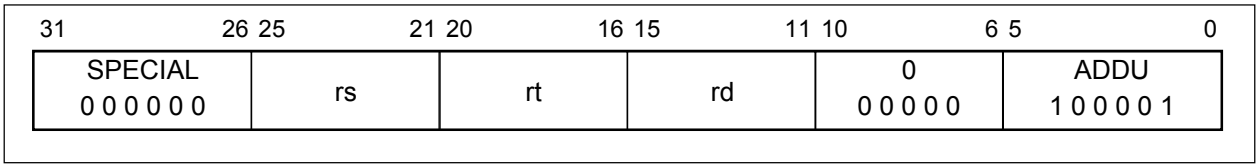
Operation:

```
32      T:   GPR [rt] ← GPR [rs] + (immediate15)^16 || immediate15...0

64      T:   temp ← GPR [rs] + (immediate15)^48 || immediate15...0
             GPR [rt] ← (temp31)^32 || temp31...0
```

Exceptions:

None

ADDU Add Unsigned ADDU



Format:

ADDU rd, rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*. No integer overflow exception occurs under any circumstances. In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the ADD instruction is that ADDU never causes an integer overflow exception.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

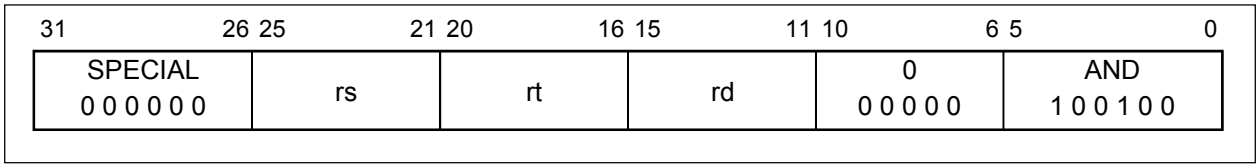
Operation:

```
32      T:   GPR [rt] ← GPR [rs] + GPR [rt]

64      T:   temp ← GPR [rs] + GPR [rt]
             GPR [rd] ← (temp31)^32 || temp31...0
```

Exceptions:

None

AND AND AND



Format:

AND rd, rs, rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical AND operation. The result is placed into general register *rd*.

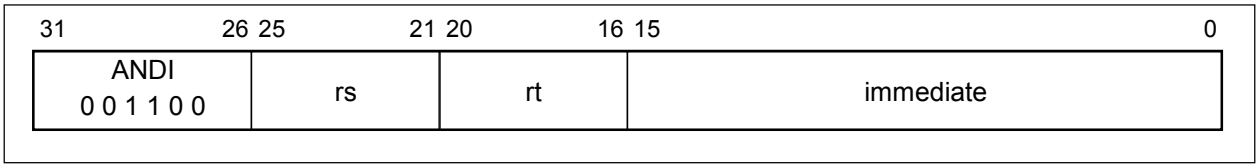
Operation:

32	T:	GPR [rd] ← GPR [rs] and GPR [rt]
64	T:	GPR [rd] ← GPR [rs] and GPR [rt]

Exceptions:

None

ANDI AND Immediate ANDI



Format:

ANDI rt, rs, immediate

Description:

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical AND operation. The result is placed into general register *rt*.

Operation:

32	T:	GPR [rt] $\leftarrow 0^{16}$ || (immediate and GPR $[rs]_{15...0}$)
64	T:	GPR [rt] $\leftarrow 0^{48}$ || (immediate and GPR $[rs]_{15...0}$)

Exceptions:

None

BC0F Branch on Coprocessor 0 False BC0F

31 26	25 21	20 16	15 0
COPz 0 1 0 0 X X Note	BC 0 1 0 0 0	BCF 0 0 0 0 0	offset

Format:

BC0F offset

Description:

A 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. If the Coprocessor 0's condition signal (CpCond), as sampled during the previous instruction, is false, then the program branches to the target address with a delay of one instruction.

Because the condition signal is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition signal.

Operation:

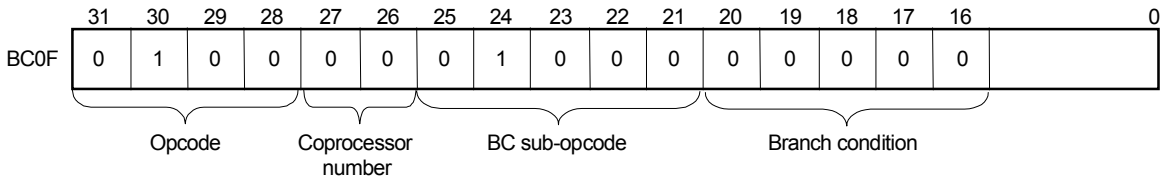
```
32      T–1: condition ← not SR18
        T:   target ← (offset15)^14 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             endif

64      T–1: condition ← not SR18
        T:   target ← (offset15)^46 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             endif
```

Exceptions:

Coprocessor unusable exception

Note See the opcode table below, or **9.4 CPU Instruction Opcode Bit Encoding**.

Opcode Table:



BC0FL Branch on Coprocessor 0 False Likely BFC0FL

31 26	25 21	20 16	15 0
COPz 0 1 0 0 X X Note	BC 0 1 0 0 0	BCFL 0 0 0 1 0	offset

Format:

BC0FL offset

Description:

A 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. If the Coprocessor 0's condition signal (CpCond), as sampled during the previous instruction, is false, the target address is branched to with a delay of one instruction.

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

Because the condition signal is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition signal.

Operation:

```
32      T–1: condition ← not SR18
        T:   target ← (offset15)^14 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             else
                 NullifyCurrentInstruction
             endif

64      T–1: condition ← not SR18
        T:   target ← (offset15)^46 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             else
                 NullifyCurrentInstruction
             endif
```

Exceptions:

Coprocessor unusable exception

Note See the opcode table below, or **9.4 CPU Instruction Opcode Bit Encoding**.

Opcode Table:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	0
BC0FL	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	

Opcode | Coprocessor number | BC sub-opcode | Branch condition

BC0T Branch on Coprocessor 0 True BC0T

31 26	25 21	20 16	15 0
COPz 0 1 0 0 X X [Note]	BC 0 1 0 0 0	BCT 0 0 0 0 1	offset

Format:

BC0T offset

Description:

A 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. If the Coprocessor 0's condition signal (CpCond), as sampled during the previous instruction, is true, then the program branches to the target address, with a delay of one instruction.

Because the condition signal is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition signal.

Operation:

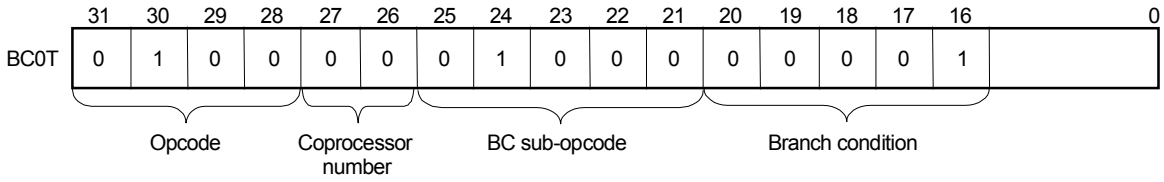
```
32      T–1: condition ← SR18
        T:   target ← (offset15)^14 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             endif

64      T–1: condition ← SR18
        T:   target ← (offset15)^46 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             endif
```

Exceptions:

Coprocessor unusable exception

Note See the opcode table below, or **9.4 CPU Instruction Opcode Bit Encoding**.

Opcode Table:



BC0TL Branch on Coprocessor 0 True Likely BC0TL

31 26	25 21	20 16	15 0
COPz 0 1 0 0 X X Note	BC 0 1 0 0 0	BCTL 0 0 0 1 1	offset

Format:

BC0TL offset

Description:

A 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. If the Coprocessor 0's condition signal (CpCond), as sampled during the previous instruction, is true, the target address is branched to with a delay of one instruction.

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

Because the condition signal is sampled during the previous instruction, there must be at least one instruction between this instruction and a coprocessor instruction that changes the condition signal.

Operation:

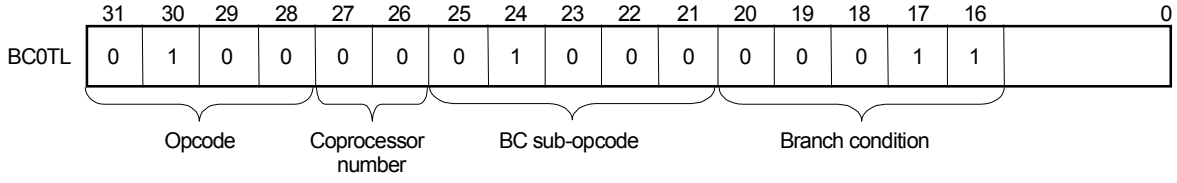
```
32      T–1: condition ← SR18
        T:   target ← (offset15)^14 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             else
                 NullifyCurrentInstruction
             endif

64      T–1: condition ← SR18
        T:   target ← (offset15)^46 || offset || 0^2
        T+1: if condition then
                 PC ← PC + target
             else
                 NullifyCurrentInstruction
             endif
```

Exceptions:

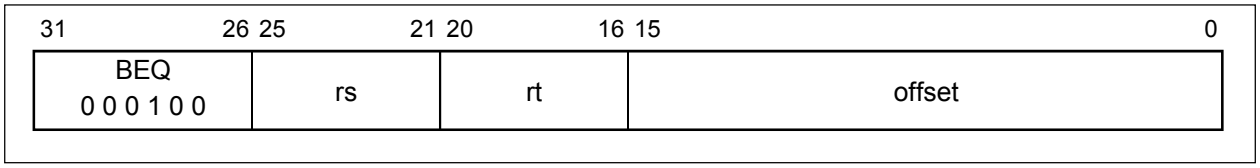
Coprocessor unusable exception

Note See the opcode table below, or **9.4 CPU Instruction Opcode Bit Encoding**.

Opcode Table:



BEQ Branch on Equal BEQ



Format:

BEQ rs, rt, offset

Description:

A 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. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are equal, then the program branches to the target address, with a delay of one instruction.

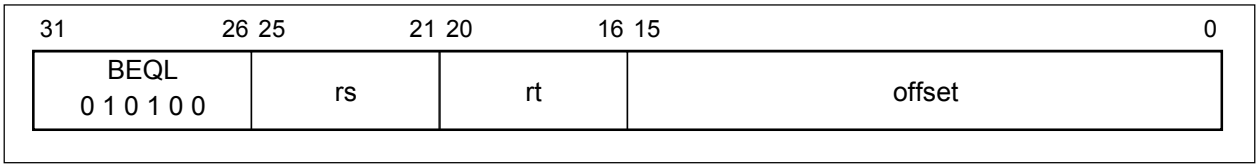
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs] = GPR [rt])
        T+1:  if condition then
                    PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs] = GPR [rt])
        T+1:  if condition then
                    PC ← PC + target
              endif
```

Exceptions:

None

BEQL Branch on Equal Likely BEQL



Format:

BEQL rs, rt, offset

Description:

A 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. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are equal, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

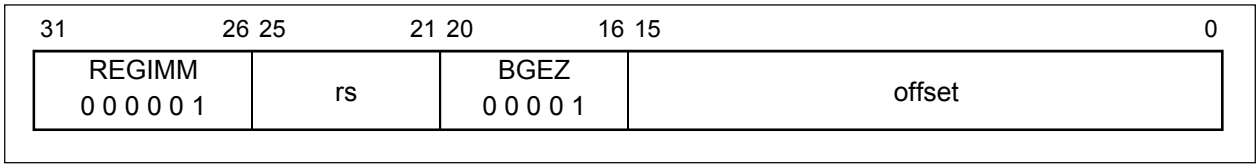
Operation:

```
32      T:   target ← (offset15)^14 || offset || 0^2
             condition ← (GPR [rs] = GPR [rt])
        T+1: if condition then
                  PC ← PC + target
             else
                  NullifyCurrentInstruction
             endif

64      T:   target ← (offset15)^46 || offset || 0^2
             condition ← (GPR [rs] = GPR [rt])
        T+1: if condition then
                  PC ← PC + target
             else
                  NullifyCurrentInstruction
             endif
```

Exceptions:

None

BGEZ Branch on Greater than or Equal to Zero BGEZ



Format:

BGEZ rs, offset

Description:

A 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. If the contents of general register *rs* have the sign bit cleared, then the program branches to the target address, with a delay of one instruction.

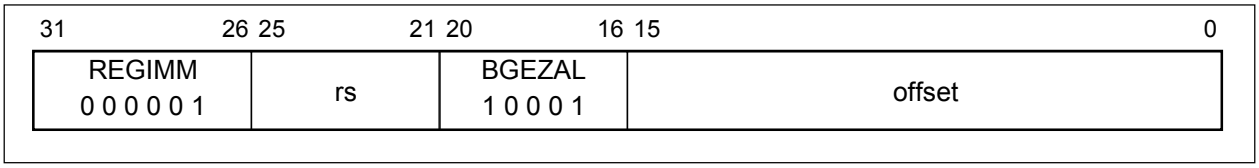
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 0)
        T+1:  if condition then
                  PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 0)
        T+1:  if condition then
                  PC ← PC + target
              endif
```

Exceptions:

None

BGEZAL Branch on Greater than or Equal to Zero And Link BGEZAL



Format:

BGEZAL rs, offset

Description:

A 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. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit cleared, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *r31*, because such an instruction is not restartable. An attempt to execute such an instruction is not trapped, however.

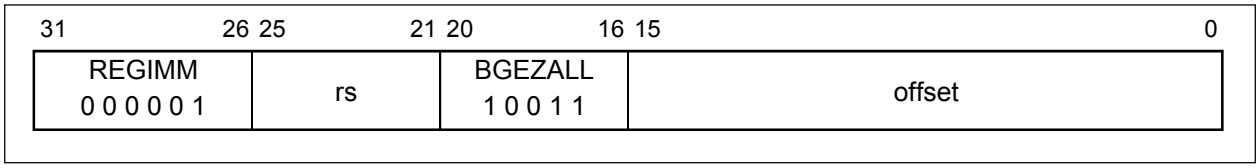
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 0)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 0)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              endif
```

Exceptions:

None

BGEZALL Branch on Greater than or Equal to Zero And Link Likely BGEZALL



Format:

BGEZALL rs, offset

Description:

A 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. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit cleared, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *r31*, because such an instruction is not restartable. An attempt to execute such an instruction is not trapped, however. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

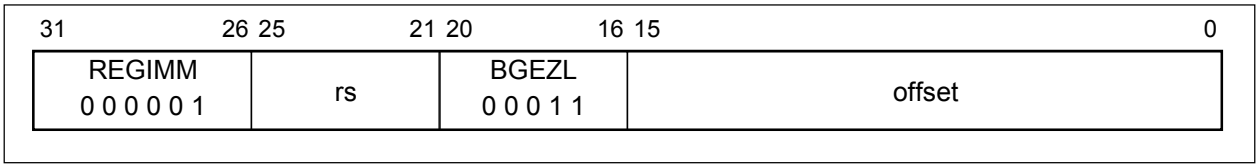
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 0)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              else
                    NullifyCurrentInstruction
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 0)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              else
                    NullifyCurrentInstruction
              endif
```

Exceptions:

None

BGEZL Branch on Greater than or Equal to Zero Likely BGEZL



Format:

BGEZL rs, offset

Description:

A 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. If the contents of general register *rs* have the sign bit cleared, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

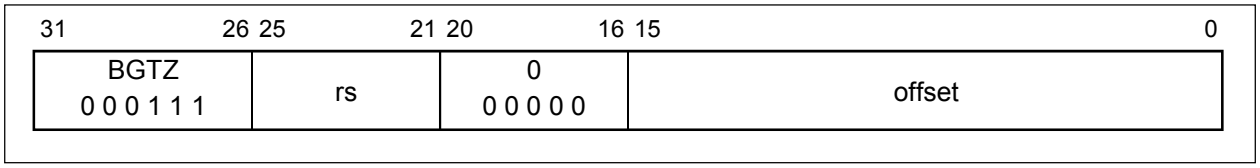
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 0)
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 0)
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif
```

Exceptions:

None

BGTZ Branch on Greater than Zero BGTZ



Format:

BGTZ rs, offset

Description:

A 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. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit cleared and are not equal to zero, then the program branches to the target address, with a delay of one instruction.

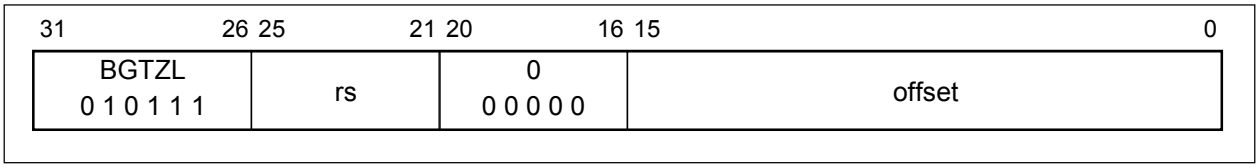
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 0) or (GPR [rs] ≠ 0^32)
        T+1:  if condition then
                    PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 0) or (GPR [rs] ≠ 0^64)
        T+1:  if condition then
                    PC ← PC + target
              endif
```

Exceptions:

None

BGTZL Branch on Greater than Zero Likely BGTZL



Format:

BGTZL rs, offset

Description:

A 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. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit cleared and are not equal to zero, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

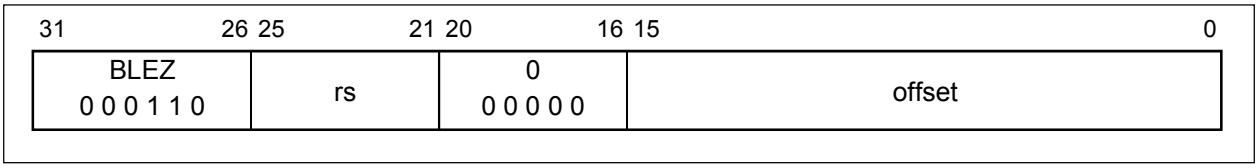
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 0) or (GPR [rs] ≠ 0^32)
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 0) or (GPR [rs] ≠ 0^64)
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif
```

Exceptions:

None

BLEZ Branch on Less than or Equal to Zero BLEZ



Format:

BLEZ rs, offset

Description:

A 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. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit set or are equal to zero, then the program branches to the target address, with a delay of one instruction.

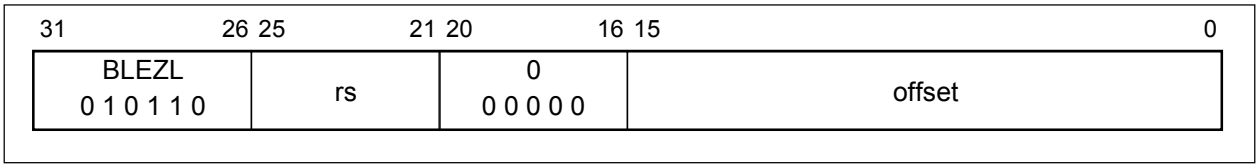
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 1) or (GPR [rs] = 0^32)
        T+1:  if condition then
                    PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 1) or (GPR [rs] = 0^64)
        T+1:  if condition then
                    PC ← PC + target
              endif
```

Exceptions:

None

BLEZL Branch on Less than or Equal to Zero Likely BLEZL



Format:

BLEZL rs, offset

Description:

A 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. The contents of general register *rs* is compared to zero. If the contents of general register *rs* have the sign bit set or are equal to zero, then the program branches to the target address, with a delay of one instruction.

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

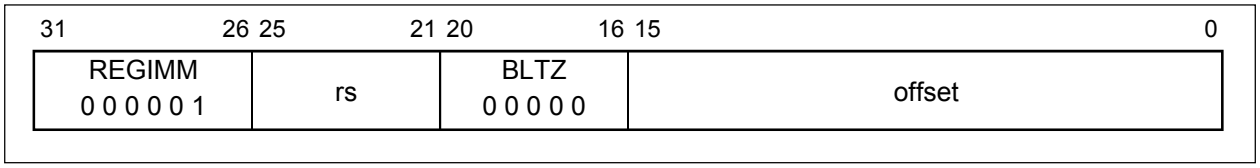
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 1) or (GPR [rs] = 0^32)
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 1) or (GPR [rs] = 0^64)
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif
```

Exceptions:

None

BLTZ Branch on Less than Zero BLTZ



Format:

BLTZ rs, offset

Description:

A 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. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction.

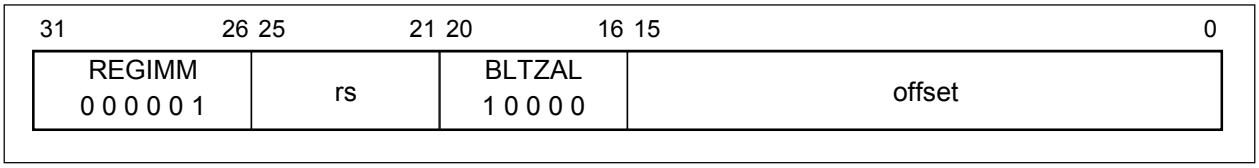
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 1)
        T+1:  if condition then
                  PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 1)
        T+1:  if condition then
                  PC ← PC + target
              endif
```

Exceptions:

None

BLTZAL Branch on Less than Zero and Link BLTZAL



Format:

BLTZAL rs, offset

Description:

A 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. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *r31*, because such an instruction is not restartable. An attempt to execute such an instruction is not trapped, however.

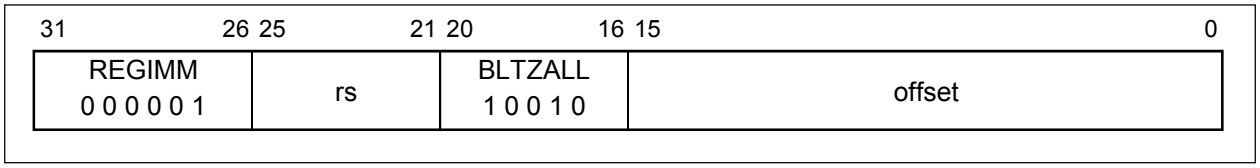
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 1)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 1)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              endif
```

Exceptions:

None

BLTZALL Branch on Less than Zero and Link Likely BLTZALL



Format:

BLTZALL rs, offset

Description:

A 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. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *r31*, because such an instruction is not restartable. An attempt to execute such an instruction is not trapped, however. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

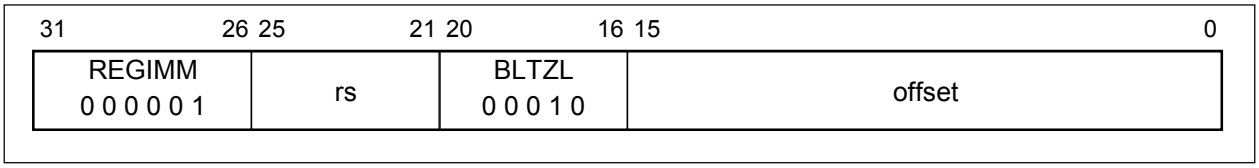
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs]31 = 1)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              else
                    NullifyCurrentInstruction
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs]63 = 1)
              GPR [31] ← PC + 8
        T+1:  if condition then
                    PC ← PC + target
              else
                    NullifyCurrentInstruction
              endif
```

Exceptions:

None

BLTZL Branch on Less than Zero Likely BLTZL



Format:

BLTZ rs, offset

Description:

A 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. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

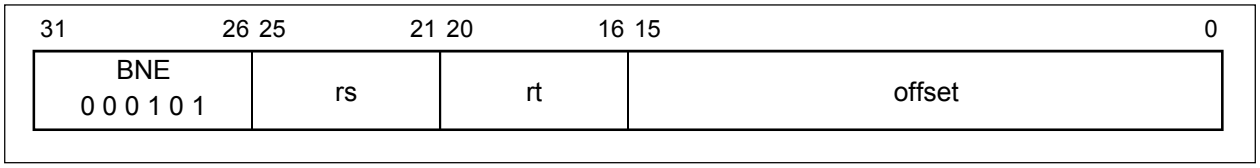
Operation:

```
32      T:   target ← (offset15)^14 || offset || 0^2
             condition ← (GPR [rs]31 = 1)
        T+1: if condition then
                  PC ← PC + target
             else
                  NullifyCurrentInstruction
             endif

64      T:   target ← (offset15)^46 || offset || 0^2
             condition ← (GPR [rs]63 = 1)
        T+1: if condition then
                  PC ← PC + target
             else
                  NullifyCurrentInstruction
             endif
```

Exceptions:

None

BNE Branch on Not Equal BNE



Format:

BNE rs, rt, offset

Description:

A 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. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

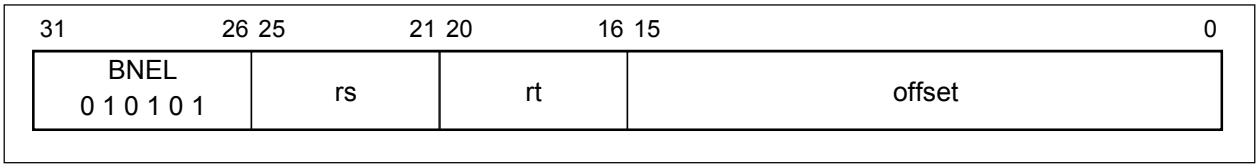
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs] ≠ GPR [rt])
        T+1:  if condition then
                    PC ← PC + target
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs] ≠ GPR [rt])
        T+1:  if condition then
                    PC ← PC + target
              endif
```

Exceptions:

None

BNEL Branch on Not Equal Likely BNEL



Format:

BNEL rs, rt, offset

Description:

A 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. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

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

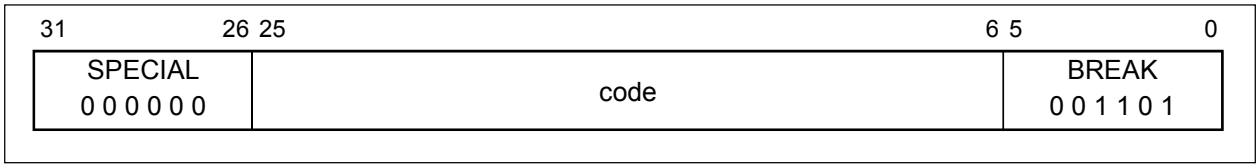
Operation:

```
32      T:    target ← (offset15)^14 || offset || 0^2
              condition ← (GPR [rs] ≠ GPR [rt])
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif

64      T:    target ← (offset15)^46 || offset || 0^2
              condition ← (GPR [rs] ≠ GPR [rt])
        T+1:  if condition then
                  PC ← PC + target
              else
                  NullifyCurrentInstruction
              endif
```

Exceptions:

None

BREAK Breakpoint BREAK



Format:

BREAK

Description:

A breakpoint trap occurs, immediately and unconditionally transferring control to the exception handler.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

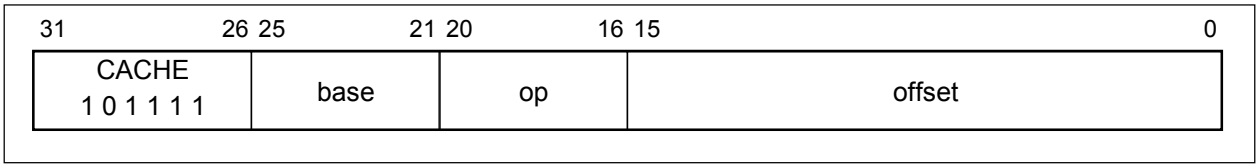
Operation:

```
32, 64   T:     BreakpointException
```

Exceptions:

Breakpoint exception

CACHE — Cache Operation — CACHE



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 5-bit sub-opcode *op* specifies a cache operation for that address.

If CP0 is not usable (User or Supervisor mode) and the CP0 enable bit in the Status register is cleared, a coprocessor unusable exception is taken. The operation of this instruction on any operation/cache combination not listed below, or on a secondary cache, 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 cache of $2^{CACHEBITS}$ bytes with $2^{LINEBITS}$ bytes per tag, $vAddr_{CACHEBITS...LINEBITS}$ in the VR4121, VR4122, VR4181, and VR4181A or $vAddr_{CACHEBITS-2...LINEBITS}$ in the VR4131 specifies the block. In the VR4131, bit 31 of the virtual address indicates the way of cache to be used.

The Hit operation translates the virtual address to a physical address using the TLB, 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.

CACHE Cache (Continued) CACHE

Write back from a primary cache goes to memory. 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) to unmapped addresses may be used to avoid TLB exceptions. This operation never causes a TLB Modified exception.

Bits 17 and 16 ($op_{1..0}$) of the instruction code specify the cache as follows:

$op_{1..0}$	Name	Cache
0	I	Instruction cache
1	D	Data cache
2	—	Reserved
3	—	Reserved

CACHE — Cache (Continued) — CACHE

Bits 20 to 18 ($op_{4..2}$) of the instruction specify the operation as follows:

$op_{4..2}$	Cache	Name	Operation
0	I	Index_Invalidate	Set the cache state of the cache block to Invalid. This operation can also be used to cancel lock of a cache block in the VR4131.
0	D	Index_Write_ Back_Invalidate	Examine the cache state and W bit of the primary data cache block at the index specified by the virtual address. If the state is not Invalid and the W bit is set, then write back the block to memory. The address to write is taken from the primary cache tag. Set cache state of primary cache block to Invalid. This operation can also be used to cancel lock of a cache block in the VR4131.
1	I, D	Index_Load_Tag	Read the tag for the cache block at the specified index and place it into the TagLo register of the CP0.
2	I, D	Index_Store_ Tag	Write the tag for the cache block at the specified index from the TagLo register of the CP0.
3	D	Create_Dirty_ Exclusive	This operation is used to avoid loading data needlessly from 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 memory. In all cases, set the cache state to Dirty.
4	I, D	Hit_Invalidate	If the cache block contains the specified address, mark the cache block Invalid. This operation can also be used to cancel lock of a cache block in the VR4131.
5	D	Hit_Write_Back Invalidate	If the cache block contains the specified address, write back the data if it is dirty, and mark the cache block Invalid.
5	I	Fill	Fill the primary instruction cache block from memory. This operation can also be used to cancel lock of a cache block in the VR4131.
6	D	Hit_Write_Back	If the cache block contains the specified address, and the W bit is set, write back the data to memory and clear the W bit.
6	I	Hit_Write_Back	If the cache block contains the specified address, write back the data unconditionally.
7	I, D	Fetch_and_Lock	For the VR4131 only. If the cache block contains the specified address, fill the cache block from memory. Locks the cache line regardless of refilling the cache block.

CACHE — Cache (Continued) — CACHE

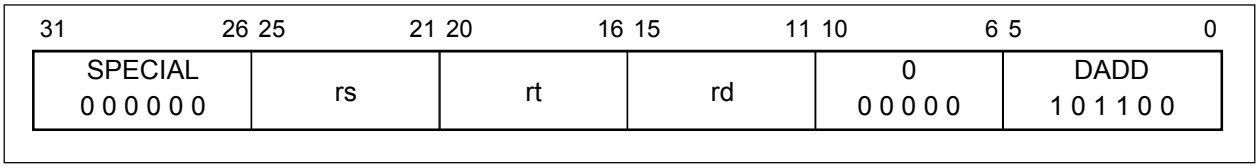
Operation:

```
32, 64   T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
              (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
              CacheOp (op, vAddr, pAddr)
```

Exceptions:

Coprocessor unusable exception
TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

DADD Doubleword Add DADD



Format:

DADD rd, rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*.

An integer overflow exception occurs if the carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

Operation:

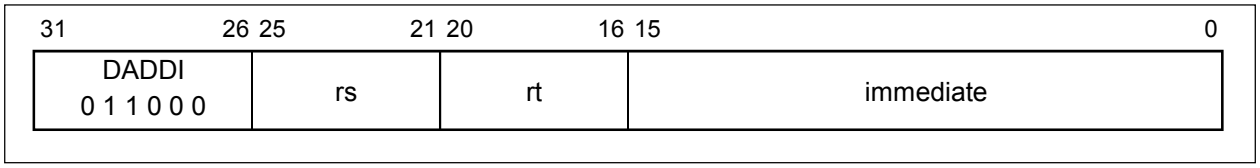
```
32, 64   T:    GPR [rd] ← GPR [rs] + GPR [rt]
```

Exceptions:

Integer overflow exception

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DADDI Doubleword Add Immediate DADDI



Format:

DADDI rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt*.

An integer overflow exception occurs if carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rt* is not modified when an integer overflow exception occurs.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

Operation:

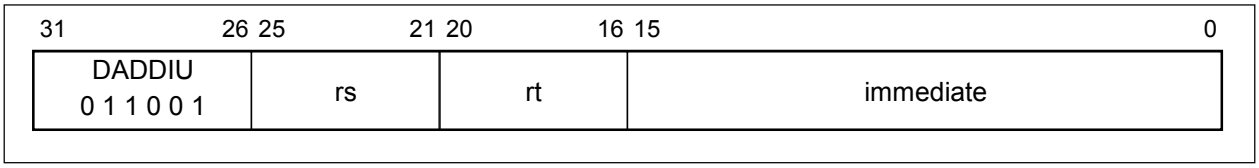
```
32, 64   T:    GPR [rt] ← GPR [rs] + (immediate15)^48 || immediate15...0
```

Exceptions:

Integer overflow exception

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DADDIU Doubleword Add Immediate Unsigned DADDIU



Format:

DADDIU rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.*

The only difference between this instruction and the DADDI instruction is that DADDIU never causes an overflow exception.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

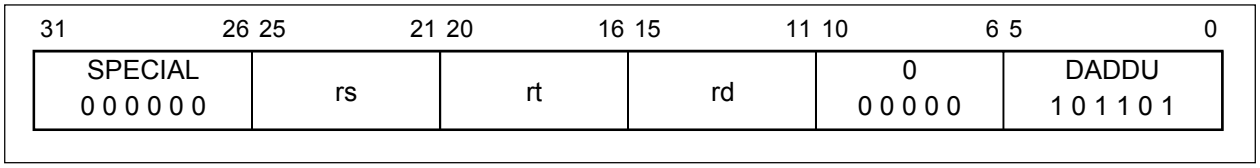
Operation:

```
64      T:   GPR [rt] ← GPR [rs] + (immediate15)^48 || immediate15...0
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DADDU Doubleword Add Unsigned DADDU



Format:

DADDU rd, rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*.

The only difference between this instruction and the DADD instruction is that DADDU never causes an overflow exception.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

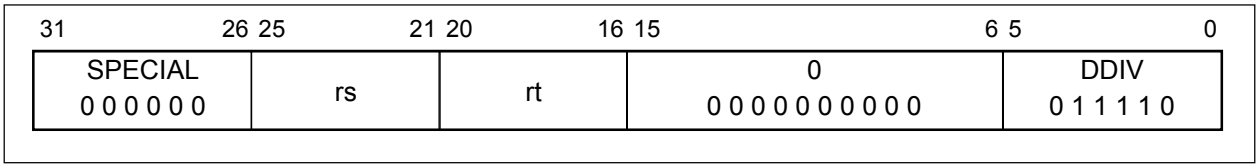
Operation:

```
64      T:    GPR [rd] ← GPR [rs] + GPR [rt]
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DDIV Doubleword Divide DDIV



Format:

DDIV rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as 2's complement values. No overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

This instruction is typically followed by additional instructions to check for a zero divisor and for overflow.

When the operation completes, the doubleword quotient of the result is loaded into special register *LO*, and the doubleword remainder of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

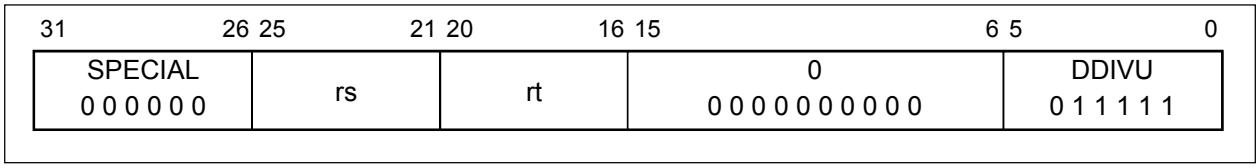
Operation:

```
32, 64  T−2: LO ← undefined
             HI ← undefined
        T−1: LO ← undefined
             HI ← undefined
        T:   LO ← GPR [rs] div GPR [rt]
             HI ← GPR [rs] mod GPR [rt]
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DDIVU Doubleword Divide Unsigned DDIVU



Format:

DDIVU rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as unsigned values. No integer overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

This instruction may be followed by additional instructions to check for a zero divisor.

When the operation completes, the doubleword quotient of the result is loaded into special register *LO*, and the doubleword remainder of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

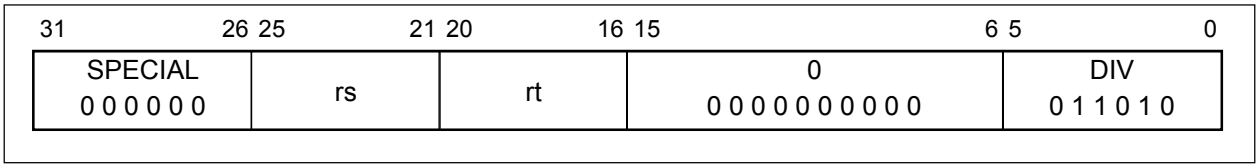
Operation:

```
64      T–2: LO ← undefined
             HI ← undefined
        T–1: LO ← undefined
             HI ← undefined
        T:   LO ← (0 || GPR [rs]) div (0 || GPR [rt])
             HI ← (0 || GPR [rs]) mod (0 || GPR [rt])
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DIV — Divide — DIV



Format:

DIV rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as 2's complement values. No overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

This instruction is typically followed by additional instructions to check for a zero divisor and for overflow.

When the operation completes, the doubleword quotient of the result is loaded into special register *LO*, and the doubleword remainder of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

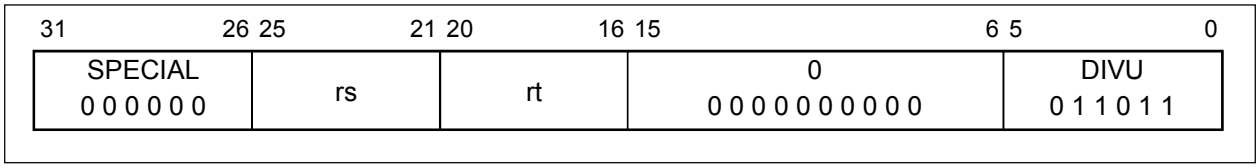
Operation:

```
32      T−2: LO ← undefined
             HI ← undefined
        T−1: LO ← undefined
             HI ← undefined
        T:   LO ← GPR [rs] div GPR [rt]
             HI ← GPR [rs] mod GPR [rt]

64      T−2: LO ← undefined
             HI ← undefined
        T−1: LO ← undefined
             HI ← undefined
        T:   q  ← GPR [rs] 31...0 div GPR [rt] 31...0
             r  ← GPR [rs] 31...0 mod GPR [rt] 31...0
             LO ← (q 31)^32 || q 31...0
             HI ← (r 31)^32 || r 31...0
```

Exceptions:

None

DIVU Divide Unsigned DIVU



Format:

DIVU rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt,* treating both operands as unsigned values. No integer overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

This instruction is typically followed by additional instructions to check for a zero divisor.

When the operation completes, the doubleword quotient of the result is loaded into special register *LO*, and the doubleword remainder of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

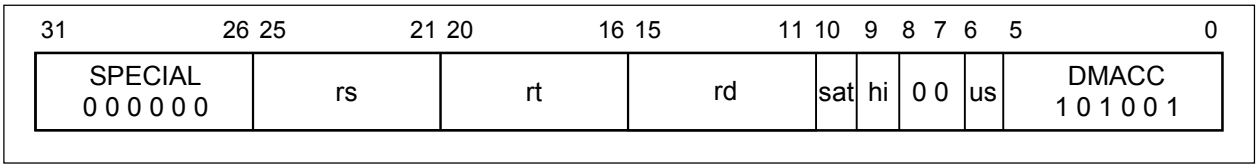
Operation:

```
32      T–2: LO ← undefined
             HI ← undefined
        T–1: LO ← undefined
             HI ← undefined
        T:   LO ← (0 || GPR [rs]) div (0 || GPR [rt])
             HI ← (0 || GPR [rs]) mod (0 || GPR [rt])

64      T–2: LO ← undefined
             HI ← undefined
        T–1: LO ← undefined
             HI ← undefined
        T:   q  ← (0 || GPR [rs]31...0 ) div (0 || GPR [rt]31...0)
             r  ← (0 || GPR [rs]31...0 ) mod (0 || GPR [rt]31...0)
             LO ← (q31)32 || q31...0
             HI ← (r31)32 || r31...0
```

Exceptions:

None

DMACC Doubleword Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) DMACC



Format:

DMACC rd, rs, rt
DMACCU rd, rs, rt
DMACCHI rd, rs, rt
DMACCHIU rd, rs, rt
DMACCS rd, rs, rt
DMACCUS rd, rs, rt
DMACCHIS rd, rs, rt
DMACCHIUS rd, rs, rt

Description:

The mnemonics of the DMACC instruction differ as shown in the table below by the setting of the *sat*, *hi*, or *us* bits.

Mnemonic	sat	hi	us
DMACC	0	0	0
DMACCU	0	0	1
DMACCHI	0	1	0
DMACCHIU	0	1	1
DMACCS	1	0	0
DMACCUS	1	0	1
DMACCHIS	1	1	0
DMACCHIUS	1	1	1

The number of valid bits in the operands differs depending on whether saturation processing is executed (*sat* = 1) or not (*sat* = 0).

- **When saturation processing is executed (*sat* = 1): DMACCS, DMACCUS, DMACCHIS, and DMACCHIUS instructions**

  The contents of general register *rs* are multiplied by the contents of general register *rt*. If *us* = 1, the contents of both operands are handled as 16-bit unsigned data. If *us* = 0, the contents are handled as 16-bit signed integers. Sign/zero extension by software is required for bits 16 to 31 in the operands.

DMACC Doubleword Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) DMACC

The product of this multiply operation is added to the value in special register *LO*. If *us* = 1, this add operation handles the values being added as 32-bit unsigned data. If *us* = 0, the values are handled as 32-bit signed integers. Sign/zero extension by software is required for bits 32 to 63 in special register *LO*.

After saturation processing of 32 bits has been performed (refer to the table below), the sum from this add operation is loaded to special register *LO*. When *hi* = 1, data that is the same as the data loaded to special register *HI* is also loaded to general register *rd*. When *hi* = 0, data that is the same as the data loaded to special register *LO* is also loaded to general register *rd*. Overflow exceptions do not occur.

- **When saturation processing is not executed (*sat* = 0): DMACC, DMACCU, DMACCHI, and DMACCHIU instructions**

  The contents of general register *rs* are multiplied by the contents of general register *rt*. If *us* = 1, the contents of both operands are handled as 32-bit unsigned data. If *us* = 0, the contents are handled as 32-bit signed integers. Sign/zero extension by software is required for bits 32 to 63 in the operands.

  The product of this multiply operation is added to the value in special register *LO*. If *us* = 1, this add operation handles the values being added as 64-bit unsigned data. If *us* = 0, the values are handled as 64-bit signed integers.

  The sum from this add operation is loaded to special register *LO*. When *hi* = 1, data that is the same as the data loaded to special register *HI* is also loaded to general register *rd*. When *hi* = 0, data that is the same as the data loaded to special register *LO* is also loaded to general register *rd*. Overflow exceptions do not occur.

These operations are defined for 64-bit mode and 32-bit Kernel mode. A reserved instruction exception occurs if one of these instructions is executed during 32-bit User/Supervisor mode.

DMACC Doubleword Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) DMACC

The correspondence of *us* and *sat* settings and values stored during saturation processing is shown below, along with the hazard cycles required between execution of the instruction for manipulating the *HI* and *LO* registers and execution of the DMACC instruction.

Values Stored During Saturation Processing

us	sat	Overflow	Underflow
0	0	Store calculation result as is	Store calculation result as is
1	0	Store calculation result as is	Store calculation result as is
0	1	0x0000 0000 7FFF FFFF	0xFFFF FFFF 8000 0000
1	1	0xFFFF FFFF FFFF FFFF	None

Hazard Cycle Counts

Instruction	Cycle Count
MULT, MULTU	**Note1**
DMULT, DMULTU	3
DIV, DIVU	36
DDIV, DDIVU	68
MFHI, MFLO	**Note2**
MTHI, MTLO	0
MACC	0
DMACC	0

Notes 1. VR4121, VR4122 … 1
VR4131 … 0
VR4181A … 1

2. VR4121, VR4122 … 2
VR4131 … 0
VR4181A … 2

Operation:

```
32, 64, sat = 0, hi = 0, us = 0 (DMACC instruction)
        T:  temp1 ← ((GPR[rs]31)^32 || GPR [rs]) * ((GPR[rt]31)^32 || GPR [rt])
            temp2 ← temp1 + LO
            LO ← temp2
            GPR[rd] ← LO
32, 64, sat = 0, hi = 0, us = 1 (DMACCU instruction)
        T:  temp1 ← (0^32 || GPR [rs]) * (0^32 || GPR [rt])
            temp2 ← temp1 + LO
            LO ← temp2
            GPR[rd] ← LO
32, 64, sat = 0, hi = 1, us = 0 (DMACCHI instruction)
        T:  temp1 ← ((GPR[rs]31)^32 || GPR [rs]) * ((GPR[rt]31)^32 || GPR [rt])
            temp2 ← temp1 + LO
            LO ← temp2
            GPR[rd] ← HI
```

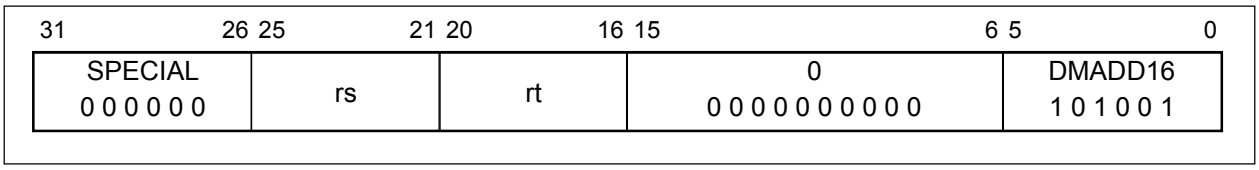
DMACC Doubleword Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) DMACC

```
32, 64, sat = 0, hi = 1, us = 1 (DMACCHIU instruction)
        T:    temp1 ← (0^32 || GPR [rs]) * (0^32 || GPR [rt])
              temp2 ← temp1 + LO
              LO ← temp2
              GPR[rd] ← HI
32, 64, sat = 1, hi = 0, us = 0 (DMACCS instruction)
        T:    temp1 ← ((GPR[rs]31)^32 || GPR [rs]) * ((GPR[rt]31)^32 || GPR [rt])
              temp2 ← saturation(temp1 + LO)
              LO ← temp2
              GPR[rd] ← LO
32, 64, sat = 1, hi = 0, us = 1 (DMACCUS instruction)
        T:    temp1 ← (0^32 || GPR [rs]) * (0^32 || GPR [rt])
              temp2 ← saturation(temp1 + LO)
              LO ← temp2
              GPR[rd] ← LO
32, 64, sat = 1, hi = 1, us = 0 (DMACCHIS instruction)
        T:    temp1 ← ((GPR[rs]31)^32 || GPR [rs]) * ((GPR[rt]31)^32 || GPR [rt])
              temp2 ← saturation(temp1 + LO)
              LO ← temp2
              GPR[rd] ← HI
32, 64, sat = 1, hi = 1, us = 1 (DMACCHIUS instruction)
        T:    temp1 ← (0^32 || GPR [rs]) * (0^32 || GPR [rt])
              temp2 ← saturation(temp1 + LO)
              LO ← temp2
              GPR[rd] ← HI
```

Exceptions:

Reserved instruction exception (in 32-bit User/Supervisor mode)

DMADD16 Doubleword Multiply and Add 16-bit Integer (for VR4181 only) DMADD16



Format:

DMADD16 rs, rt

Description:

The contents of general registers *rs* and *rt* are multiplied, treating both operands as 16-bit 2's complement values. Bits 62 to 15 of the operand must be sign-extended values.

This multiplied result and the contents of special register *LO* are added to form the result as a signed integer. When the operation completes, the doubleword result is loaded into special register *LO*.

No integer overflow exception occurs under any circumstances.

This operation is defined for the VR4181 operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

The following table shows hazard cycles between DMADD16 and other instructions.

Instruction sequence	No. of cycles
MULT/MULTU → DMADD16	1 Cycle
DMULT/DMULTU → DMADD16	4 Cycles
DIV/DIVU → DMADD16	36 Cycles
DDIV/DDIVU → DMADD16	68 Cycles
MFHI/MFLO → DMADD16	2 Cycles
MADD16 → DMADD16	0 Cycles
DMADD16 → DMADD16	0 Cycles

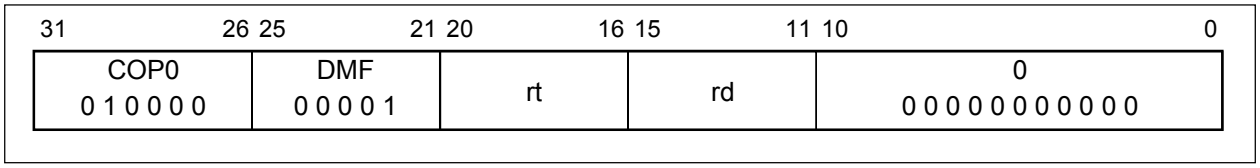
Operation:

```
32, 64   T−2: LO ← undefined
              HI ← undefined
         T−1: LO ← undefined
              HI ← undefined
         T:   temp ← GPR [rs] * GPR [rt]
              temp ← temp + LO
              LO ← temp
              HI ← undefined
```

Exceptions:

Reserved instruction exception (VR4181 in 32-bit User mode, VR4181 in 32-bit Supervisor mode)

DMFC0 Doubleword Move from System Control Coprocessor DMFC0



Format:

DMFC0 rt, rd

Description:

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

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode 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:

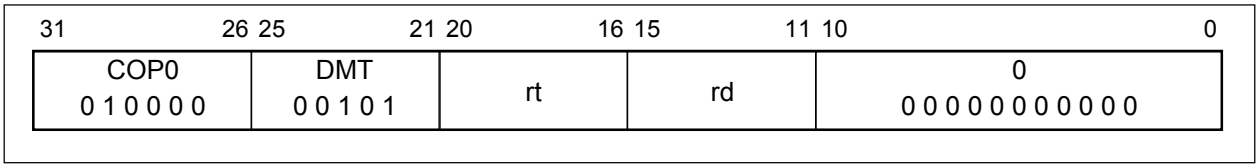
```
32, 64  T:    data ← CPR [0, rd]
        T+1:  GPR [rt] ← data
```

Exceptions:

Coprocessor unusable exception (User and Supervisor mode if CP0 not enabled)

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DMTC0 Doubleword Move to System Control Coprocessor DMTC0



Format:

DMTC0 rt, rd

Description:

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

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

All 64-bits of the coprocessor register destination 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:

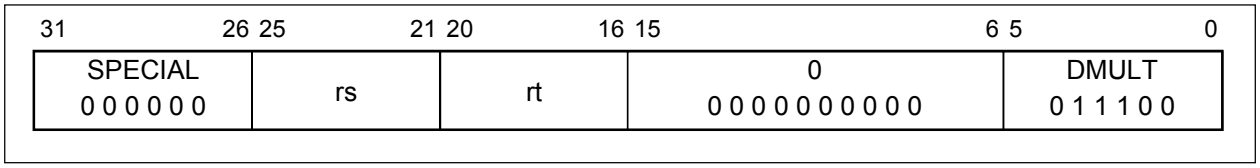
```
32, 64  T:    data ← GPR [rt]
        T+1:  CPR [0, rd] ← data
```

Exceptions:

Coprocessor unusable exception (User and Supervisor mode if CP0 not enabled)

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DMULT Doubleword Multiply DMULT



Format:

DMULT rs, rt

Description:

The contents of general registers *rs* and *rt* are multiplied, treating both operands as 2's complement values. No integer overflow exception occurs under any circumstances.

When the operation completes, the low-order doubleword of the result is loaded into special register *LO*, and the high-order doubleword of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two other instructions.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

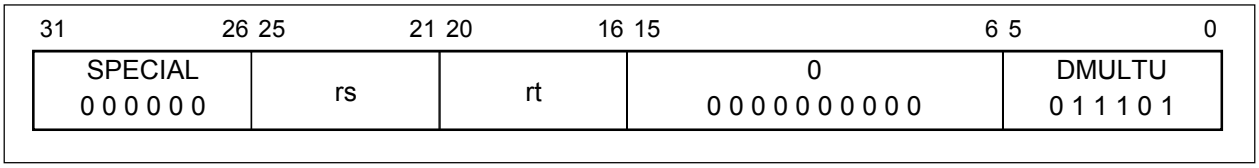
Operation:

```
32, 64  T–2: LO  ← undefined
             HI  ← undefined
        T–1: LO  ← undefined
             HI  ← undefined
        T:   t   ← GPR [rs] * GPR [rt]
             LO  ← t 63...0
             HI  ← t 127...64
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DMULTU Doubleword Multiply Unsigned DMULTU



Format:

DMULTU rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are multiplied, treating both operands as unsigned values. No overflow exception occurs under any circumstances.

When the operation completes, the low-order doubleword of the result is loaded into special register *LO*, and the high-order doubleword of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two instructions.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

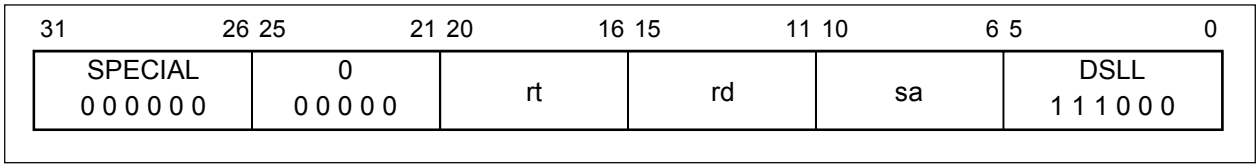
Operation:

```
32, 64  T−2: LO  ← undefined
             HI  ← undefined
        T−1: LO  ← undefined
             HI  ← undefined
        T:   t   ← (0 || GPR [rs]) * (0 || GPR [rt])
             LO  ← t 63...0
             HI  ← t 127...64
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSLL Doubleword Shift Left Logical DSLL



Format:

DSLL rd, rt, sa

Description:

The contents of general register *rt* are shifted left by *sa* bits, inserting zeros into the low-order bits. The result is placed in general register *rd.*

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

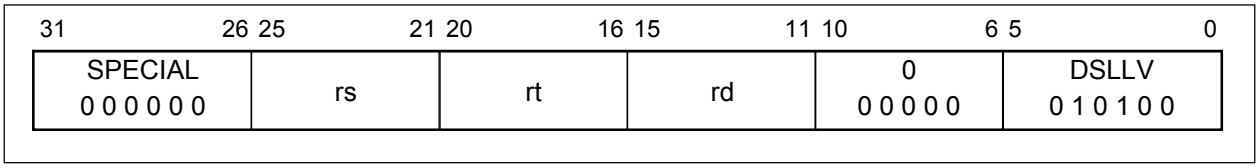
Operation:

```
32, 64   T:   s ← 0 || sa
              GPR [rd] ← GPR [rt] 63 – s...0 || 0^s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSLLV Doubleword Shift Left Logical Variable DSLLV



Format:

DSLLV rd, rt, rs

Description:

The contents of general register *rt* are shifted left by the number of bits specified by the low-order six bits contained in general register *rs*, inserting zeros into the low-order bits. The result is placed in general register *rd*. This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

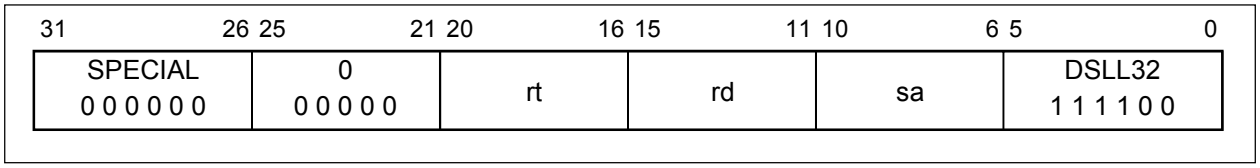
Operation:

```
32, 64   T:   s ← GPR [rs] 5...0
              GPR [rd] ← GPR [rt] 63 – s...0 || 0^s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSLL32 Doubleword Shift Left Logical + 32 DSLL32



Format:

DSLL32 rd, rt, sa

Description:

The contents of general register *rt* are shifted left by *32* + *sa* bits, inserting zeros into the low-order bits. The result is placed in general register *rd*.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

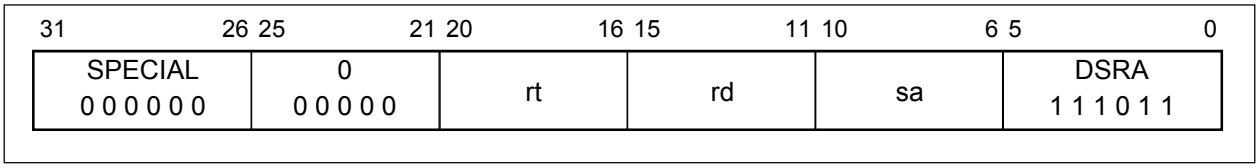
Operation:

```
32, 64   T:   s ← 1 || sa
              GPR [rd] ← GPR [rt] 63 – s...0 || 0^s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSRA Doubleword Shift Right Arithmetic DSRA



Format:

DSRA rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, sign-extending the high-order bits. The result is placed in general register *rd*.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

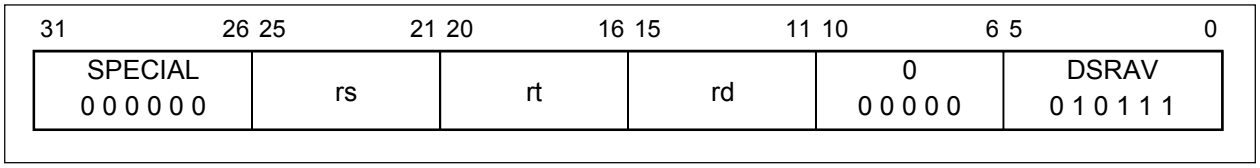
Operation:

```
32, 64   T:   s ← 0 || sa
              GPR [rd] ← (GPR [rt]63)^s || GPR [rt]63...s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSRAV Doubleword Shift Right Arithmetic Variable DSRAV



Format:

DSRAV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order six bits of general register *rs*, sign-extending the high-order bits. The result is placed in general register *rd*.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

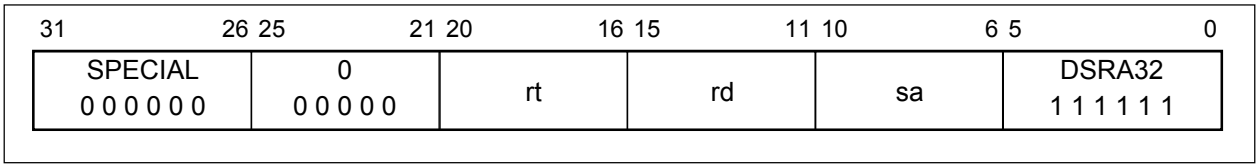
Operation:

```
32, 64   T:   s ← GPR [rs]5...0
              GPR [rd] ← (GPR [rt]63)^s || GPR [rt]63...s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSRA32 Doubleword Shift Right Arithmetic + 32 DSRA32



Format:

DSRA32 rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *32 + sa* bits, sign-extending the high-order bits. The result is placed in general register *rd*.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

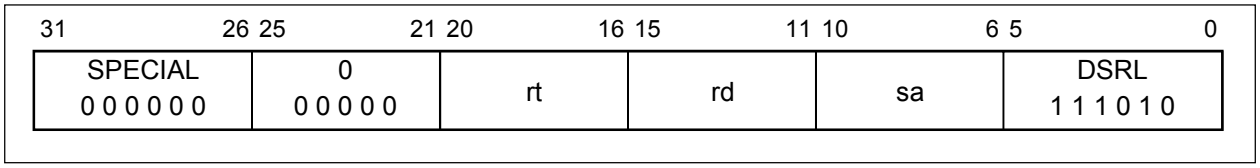
Operation:

```
32, 64   T:   s ← 1 || sa
              GPR [rd] ← (GPR [rt]63)^s || GPR [rt]63...s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSRL Doubleword Shift Right Logical DSRL



31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0	rt	rd	sa	DSRL 1 1 1 0 1 0

Format:

DSRL rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, inserting zeros into the high-order bits. The result is placed in general register *rd*.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

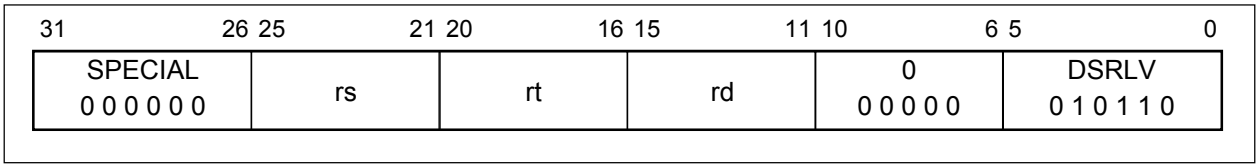
Operation:

```
32, 64   T:   s ← 0 || sa
              GPR [rd] ← 0^s || GPR [rt]63...s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSRLV Doubleword Shift Right Logical Variable DSRLV



Format:

DSRLV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order six bits of general register *rs,* inserting zeros into the high-order bits. The result is placed in general register *rd*.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

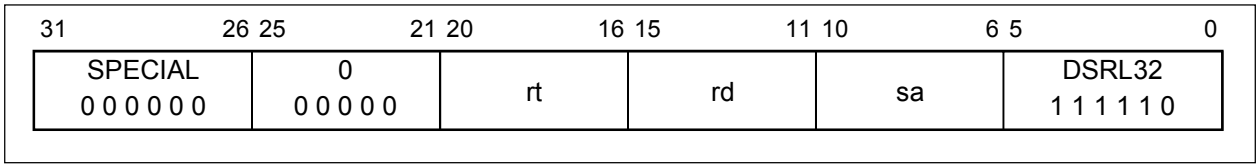
Operation:

```
32, 64   T:   s ← GPR [rs]5...0
              GPR [rd] ← 0^s || GPR [rt]63...s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSRL32 Doubleword Shift Right Logical + 32 DSRL32



Format:

DSRL32 rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *32 + sa* bits, inserting zeros into the high-order bits. The result is placed in general register *rd*.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

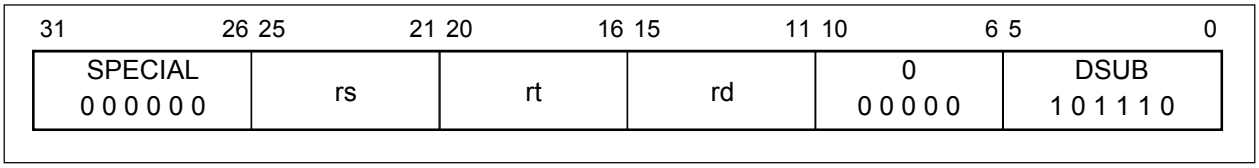
Operation:

```
32, 64   T:   s ← 1 || sa
              GPR [rd] ← 0^s || GPR [rt]63...s
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSUB Doubleword Subtract DSUB



Format:

DSUB rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd.*

An integer overflow exception takes place if the carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

Operation:

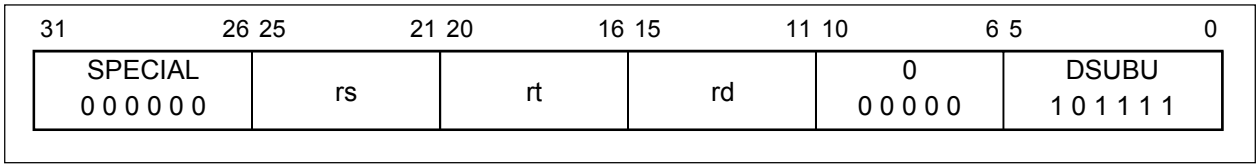
```
32, 64   T:    GPR [rd] ← GPR [rs] – GPR [rt]
```

Exceptions:

Integer overflow exception

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

DSUBU Doubleword Subtract Unsigned DSUBU



Format:

DSUBU rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd*.

The only difference between this instruction and the DSUB instruction is that DSUBU never traps on overflow.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

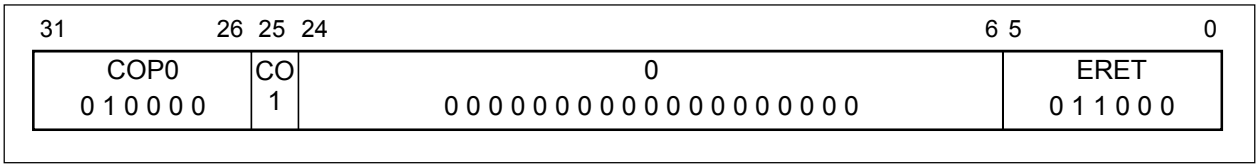
Operation:

```
32, 64   T:    GPR [rd] ← GPR [rs] – GPR [rt]
```

Exceptions:

Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

ERET Exception Return ERET



Format:

ERET

Description:

ERET is the 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 register and clear the *ERL* bit of the Status register (SR_2 = 0). Otherwise (SR_2 = 0), load the PC from the EPC register, and clear the *EXL* bit of the Status register (SR_1 = 0).

When MIPS16 instructions are enabled, the value of clearing the least significant bit of the EPC or ErrorEPC register to 0 is loaded to PC. This means the content of the least significant bit is reflected on the ISA mode bit (internal).

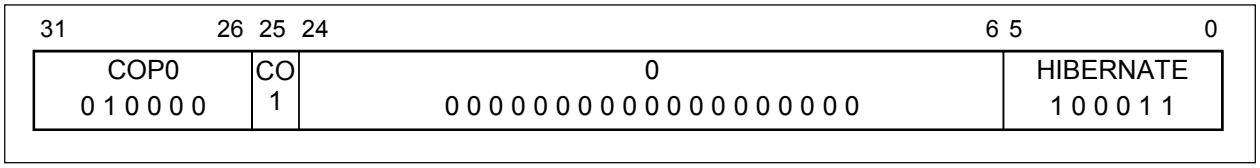
Operation:

```
32, 64   T:   if SR2 = 1 then
                  if MIPS16EN = 1 then
                      PC ← ErrorEPC63...1 || 0
                      ISA MODE ← ErrorEPC0
                  else
                      PC ← ErrorEPC
                  endif
                  SR ← SR31...3 || 0 || SR1...0
              else
                  if MIPS16EN = 1 then
                      PC ← EPC63...1 || 0
                      ISA MODE ← EPC0
                  else
                      PC ← EPC
                  endif
                  SR ← SR31...2 || 0 || SR0
              endif
```

Exceptions:

Coprocessor unusable exception

HIBERNATE Hibernate HIBERNATE



Format:

HIBERNATE

Description:

HIBERNATE instruction starts mode transition from Fullspeed mode to Hibernate mode.

When the HIBERNATE instruction finishes the WB stage, the VR4100 Series wait by the SysAD bus is idle state, and then fix the all clocks generated by the CPU core to high level, thus freezing the pipeline.

Once the VR4100 Series is in Hibernate mode, the Cold Reset sequence will cause the VR4100 Series to exit Hibernate mode and to enter Fullspeed mode.

Operation:

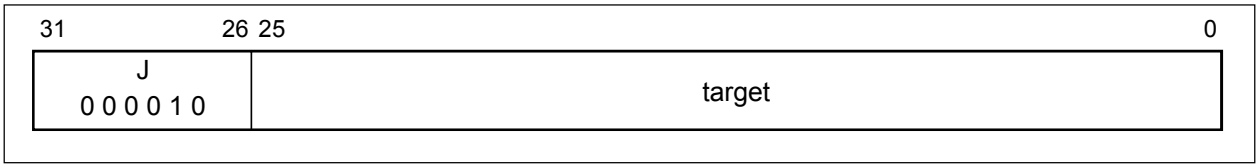
```
32, 64  T:
        T+1: Hibernate operation ( )
```

Exceptions:

Coprocessor unusable exception

Remark Refer to **Hardware User's Manual** of each product for details about the operation of the peripheral units at mode transition.

J Jump J



Format:

J target

Description:

The 26-bit target address is shifted left by two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction.

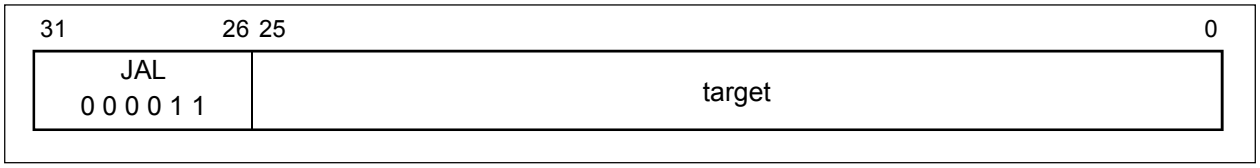
Operation:

```
32      T:    temp ← target
        T+1:  PC ← PC31...28 || temp || 0^2

64      T:    temp ← target
        T+1:  PC ← PC63...28 || temp || 0^2
```

Exceptions:

None

JAL Jump And Link JAL



Format:

JAL target

Description:

The 26-bit target address is shifted left by two bits and combined with the high-order four bits of the address of the delay slot. The program unconditionally jumps to this calculated address with a delay of one instruction. The address of the instruction immediately after a delay slot is placed in the link register (*r31*). When MIPS16 instructions are enabled, the value of bit 0 of *r31* indicates the ISA mode bit (internal) before jump.

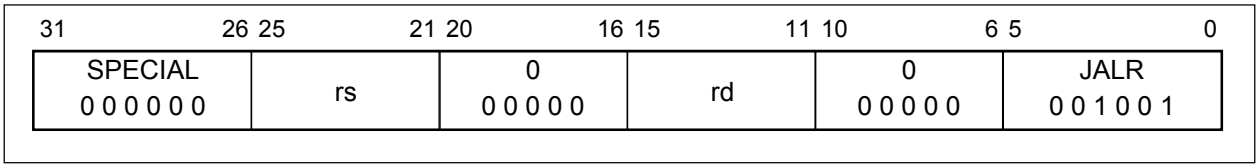
Operation:

```
32      T:    temp ← target
              if MIPS16EN = 1 then
                   GPR [31] ← (PC + 8)31...1 || ISA MODE
              else
                   GPR [31] ← PC + 8
              endif
        T+1: PC ← PC31...28 || temp || 0^2

64      T:    temp ← target
              if MIPS16EN = 1 then
                   GPR [31] ← (PC + 8)63...1 || ISA MODE
              else
                   GPR [31] ← PC + 8
              endif
        T+1: PC ← PC63...28 || temp || 0^2
```

Exceptions:

None

JALR Jump And Link Register JALR



Format:

JALR rs
JALR rd, rs

Description:

The program unconditionally jumps to the address contained in general register *rs*, with a delay of one instruction.

When MIPS16 instructions are enabled, the program unconditionally jumps with a delay of one instruction to the address indicated by the value of clearing the least significant bit of the general register *rs* to 0. Then, the content of the least significant bit of the general register *rs* is set to the ISA mode bit (internal).

The address of the instruction immediately after the delay slot is placed in general register *rd*. The default value of *rd*, if omitted in the assembly language instruction, is 31. When MIPS16 instructions are enabled, the value of bit 0 of *rd* indicates the ISA mode bit before jump.

Register specifiers *rs* and *rd* should not be equal since such an instruction does not have the same effect when re-executed because storing a link address destroys the contents of *rs* if they are equal. However, an attempt to execute this instruction is *not* trapped, and the result of executing such an instruction is undefined.

Since 32-bit length instructions must be word-aligned, a Jump and Link Register (JALR) instruction must specify a target register (*rs*) that contains an address whose two low-order bits are zero when MIPS16 instructions are enabled. If these low-order bits are not zero, an address error exception will occur when the jump target instruction is subsequently fetched.

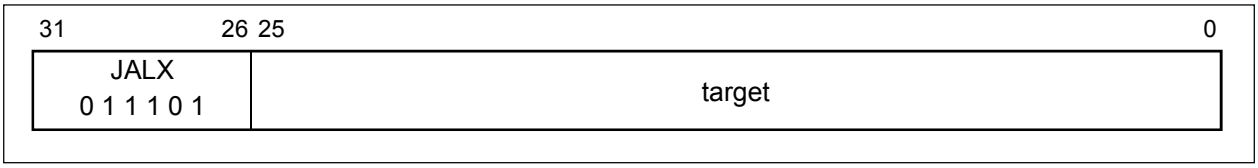
Operation:

```
32, 64   T:    temp ← GPR [rs]
               if MIPS16EN = 1 then
                    GPR [rd] ← (PC + 8)63...1 || ISA MODE
               else
                    GPR [rd] ← PC + 8
               endif
         T+1: if MIPS16EN = 1 then
                    PC ← temp63...1 || 0
                    ISA MODE ← temp0
               else
                    PC ← temp
               endif
```

Exceptions:

None

JALX Jump And Link Exchange JALX



Format:

JALX target

Description:

When MIPS16 instructions are enabled, a 26-bit target address is shifted to left by two bits and combined with the high-order four bits of the address or the delay slot. The program unconditionally jumps to the calculated address with a delay of one instruction. The address of the instruction immediately after a delay slot is placed in the link register (*r31*). The ISA mode bit is inverted with a delay of one instruction. The value of bit 0 of the link register (*r31*) indicates the ISA mode bit (internal) before jump.

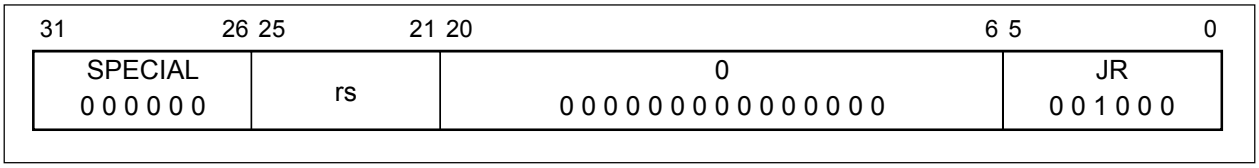
Operation:

```
32      T:    temp ← target
              GPR [31] ← (PC + 8)31...1 || ISA MODE
        T+1:  PC ← PC31...28 || temp || 0^2
              ISA MODE toggle

64      T:    temp ← target
              GPR [31] ← (PC + 8)63...1 || ISA MODE
        T+1:  PC ← PC63...28 || temp || 0^2
              ISA MODE toggle
```

Exceptions:

Reserved instruction exception (when MIPS16 instruction execution disabled)

JR Jump Register JR



Format:

JR rs

Description:

The program unconditionally jumps to the address contained in general register *rs*, with a delay of one instruction.

When MIPS16 instructions are enabled, the program unconditionally jumps with a delay of one instruction to the address indicated by the value of clearing the least significant bit of the general register *rs* to 0. Then, the content of the least significant bit of the general register *rs* is set to the ISA mode bit (internal).

Since 32-bit length instructions must be word-aligned, a Jump Register (JR) instruction must specify a target register (*rs*) that contains an address whose two low-order bits are zero when MIPS16 instructions are enabled. If these low-order bits are not zero, an address error exception will occur when the jump target instruction is subsequently fetched.

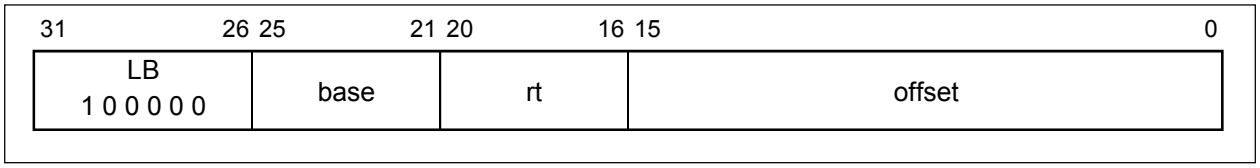
Operation:

```
32, 64  T:    temp ← GPR [rs]
        T+1:  if MIPS16EN = 1 then
                  PC ← temp63...1 || 0
                  ISA MODE ← temp0
              else
                  PC ← temp
              endif
```

Exceptions:

None

LB Load Byte LB



Format:

LB rt, 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 contents of the byte at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

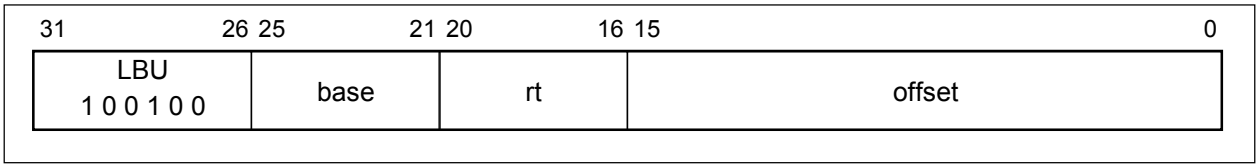
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor BigEndianCPU^3
            GPR [rt] ← (mem7 + 8*byte)^24 || mem7 + 8*byte...8*byte

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor BigEndianCPU^3
            GPR [rt] ← (mem7 + 8*byte)^56 || mem7 + 8*byte...8*byte
```

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LBU Load Byte Unsigned LBU



Format:

LBU rt, 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 contents of the byte at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

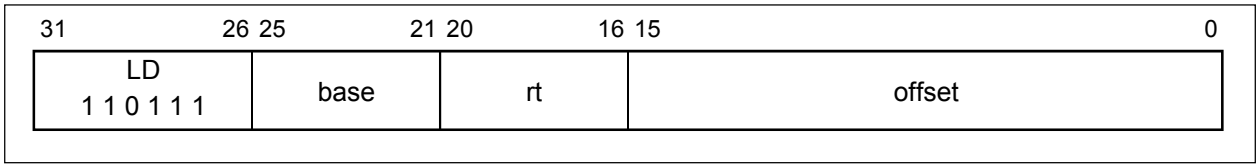
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor BigEndianCPU^3
            GPR [rt] ← 0^24 || mem7 + 8*byte...8*byte

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            mem ← LoadMemory (uncached, BYTE, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor BigEndianCPU^3
            GPR [rt] ← 0^56 || mem7 + 8*byte...8*byte
```

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LD Load Doubleword LD



Format:

LD rt, 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 contents of the 64-bit doubleword at the memory location specified by the effective address are loaded into general register *rt*.

If any of the three least-significant bits of the effective address are non-zero, an address error exception occurs.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

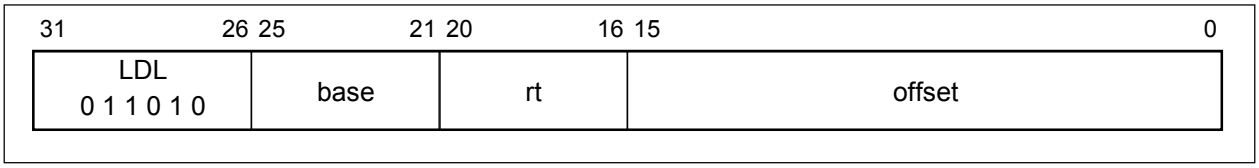
Operation:

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           data ← LoadMemory (uncached, DOUBLEWORD, pAddr, vAddr, DATA)
           GPR [rt] ← data

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           data ← LoadMemory (uncached, DOUBLEWORD, pAddr, vAddr, DATA)
           GPR [rt] ← data
```

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

LDL Load Doubleword Left LDL



Format:

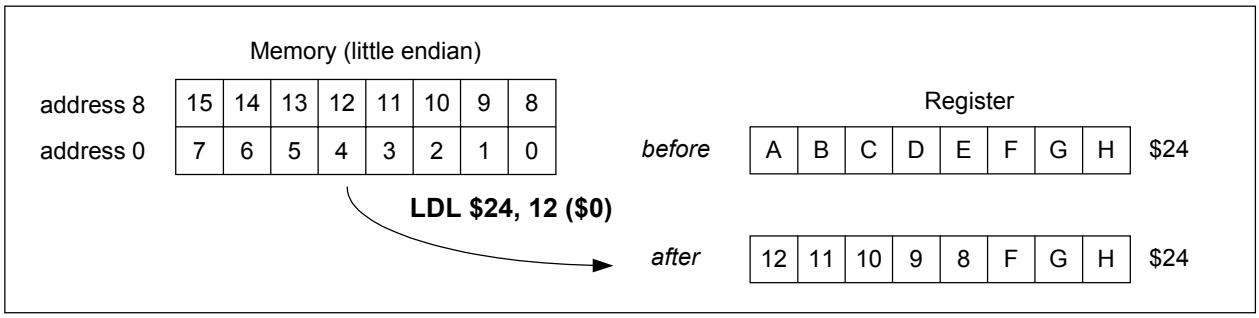
LDL rt, offset (base)

Description:

This instruction can be used in combination with the LDR instruction to load a register with eight consecutive bytes from memory, when the bytes cross a doubleword boundary. LDL loads the left portion of the register with the appropriate part of the high-order doubleword in memory; LDR loads the right portion of the register with the appropriate part of the low-order doubleword.

The LDL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the doubleword in memory that contains the specified starting byte, and places them in the high-order part of general register *rt*. The contents of the remaining part of general register *rt* is retained. From one to eight bytes will be loaded, depending on the starting byte specified.

Conceptually, it starts at the specified byte in memory and loads that byte into the high-order (left-most) byte of the register; then it loads bytes from memory into the register until it reaches the low-order byte of the doubleword in memory. The least-significant (right-most) byte(s) of the register will not be changed.



LDL Load Doubleword Left (Continued) LDL

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LDL (or LDR) instruction which also specifies register *rt*.

No address error exceptions due to alignment are possible.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            mem ← LoadMemory (uncached, byte, pAddr, vAddr, DATA)
            GPR [rt] ← mem7 + 8*byte...0 || GPR [rt]55 – 8*byte...0

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            mem ← LoadMemory (uncached, byte, pAddr, vAddr, DATA)
            GPR [rt] ← mem7 + 8*byte...0 || GPR [rt]55 – 8*byte...0
```

LDL Load Doubleword Left (Continued) LDL

Given a doubleword in a register and a doubleword in memory, the operation of LDL is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	P B C D E F G H	0	0	7	I J K L M N O P	7	0	0
1	O P C D E F G H	1	0	6	J K L M N O P H	6	0	1
2	N O P D E F G H	2	0	5	K L M N O P G H	5	0	2
3	M N O P E F G H	3	0	4	L M N O P F G H	4	0	3
4	L M N O P F G H	4	0	3	M N O P E F G H	3	0	4
5	K L M N O P G H	5	0	2	N O P D E F G H	2	0	5
6	J K L M N O P H	6	0	1	O P C D E F G H	1	0	6
7	I J K L M N O P	7	0	0	P B C D E F G H	0	0	7

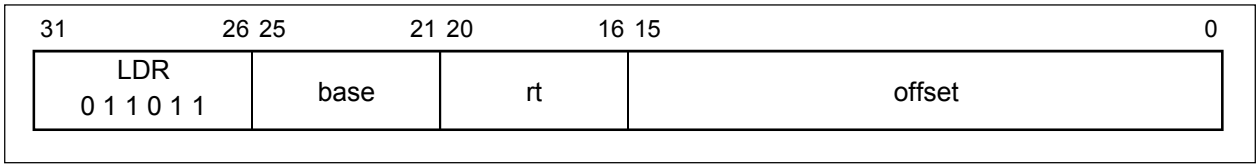
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

LDR Load Doubleword Right LDR



Format:

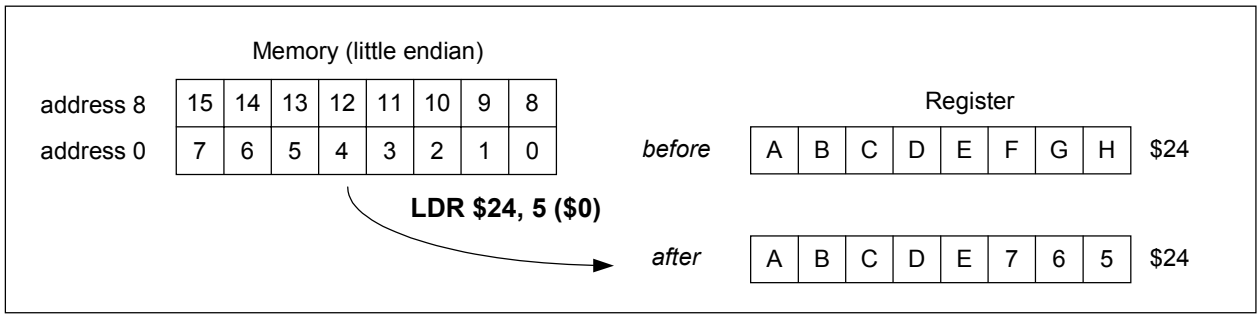
LDR rt, offset (base)

Description:

This instruction can be used in combination with the LDL instruction to load a register with eight consecutive bytes from memory, when the bytes cross a doubleword boundary. LDR loads the right portion of the register with the appropriate part of the low-order doubleword in memory; LDL loads the left portion of the register with the appropriate part of the high-order doubleword.

The LDR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the doubleword in memory that contains the specified starting byte, and places them in the low-order part of general register *rt*. The contents of the remaining part of general register *rt* is retained. From one to eight bytes will be loaded, depending on the starting byte specified.

Conceptually, it starts at the specified byte in memory and loads that byte into the low-order (right-most) byte of the register; then it loads bytes from memory into the register until it reaches the high-order byte of the doubleword in memory. The most significant (left-most) byte(s) of the register will not be changed.



LDR Load Doubleword Right (Continued) LDR

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LDR (or LDL) instruction which also specifies register *rt*.

No address error exceptions due to alignment are possible.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            mem ← LoadMemory (uncached, DOUBLEWORD-byte, pAddr, vAddr, DATA)
            GPR [rt] ← GPR [rt]63...64 – 8*byte || mem63... 8*byte

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            mem ← LoadMemory (uncached, DOUBLEWORD-byte, pAddr, vAddr, DATA)
            GPR [rt] ← GPR [rt]63...64 – 8*byte || mem63... 8*byte
```

LDR Load Doubleword Right (Continued) LDR

Given a doubleword in a register and a doubleword in memory, the operation of LDR is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	I J K L M N O P	7	0	0	A B C D E F G I	0	7	0
1	A I J K L M N O	6	1	0	A B C D E F I J	1	6	0
2	A B I J K L M N	5	2	0	A B C D E I J K	2	5	0
3	A B C I J K L M	4	3	0	A B C D I J K L	3	4	0
4	A B C D I J K L	3	4	0	A B C I J K L M	4	3	0
5	A B C D E I J K	2	5	0	A B I J K L M N	5	2	0
6	A B C D E F I J	1	6	0	A I J K L M N O	6	1	0
7	A B C D E F G I	0	7	0	I J K L M N O P	7	0	0

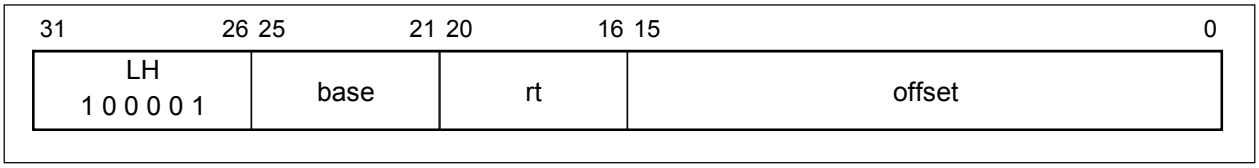
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

LH Load Halfword LH



Format:

LH rt, 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 contents of the halfword at the memory location specified by the effective address are sign-extended and loaded into general register *rt*.

If the least-significant bit of the effective address is non-zero, an address error exception occurs.

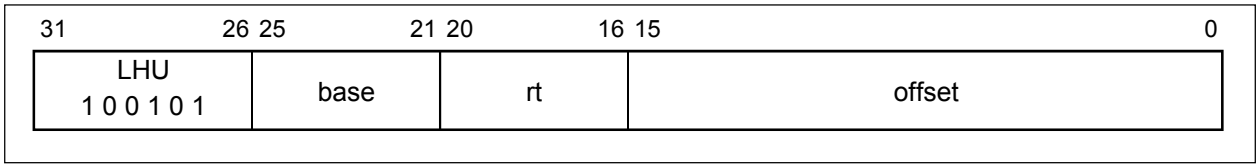
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
            mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
            GPR [rt] ← (mem15 + 8*byte)^16 || mem15 + 8*byte...8*byte

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
            mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
            GPR [rt] ← (mem15 + 8*byte)^48 || mem15 + 8*byte...8*byte
```

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LHU Load Halfword Unsigned LHU



Format:

LHU rt, 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 contents of the halfword at the memory location specified by the effective address are zero-extended and loaded into general register *rt*.

If the least-significant bit of the effective address is non-zero, an address error exception occurs.

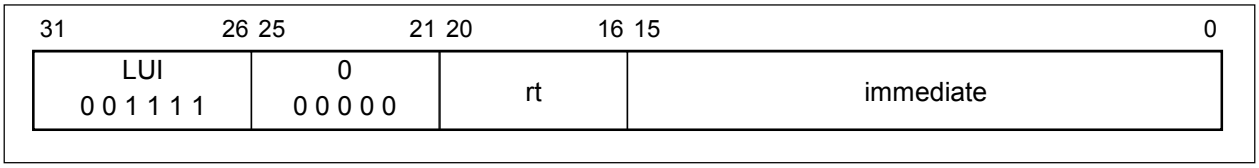
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
            mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
            GPR [rt] ← 0^16 || mem15 + 8*byte...8*byte

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
            mem ← LoadMemory (uncached, HALFWORD, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
            GPR [rt] ← 0^48 || mem15 + 8*byte...8*byte
```

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LUI Load Upper Immediate LUI



Format:

LUI rt, immediate

Description:

The 16-bit *immediate* is shifted left by 16 bits and concatenated to 16 bits of zeros. The result is placed into general register *rt*. In 64-bit mode, the loaded word is sign-extended.

Operation:

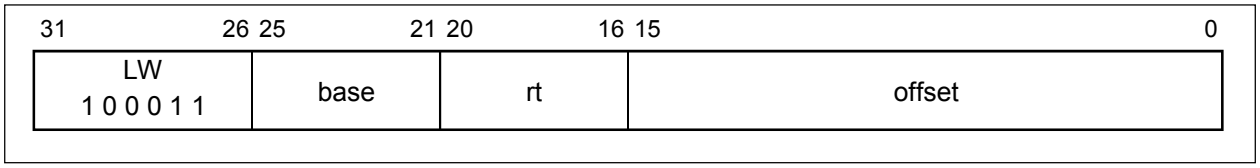
32 T: GPR [rt] ← immediate || 0^{16}

64 T: GPR [rt] ← $(immediate_{15})^{32}$ || immediate || 0^{16}

Exceptions:

None

LW Load Word LW



Format:

LW rt, 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 contents of the word at the memory location specified by the effective address are loaded into general register *rt*. In 64-bit mode, the loaded word is sign-extended.

If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

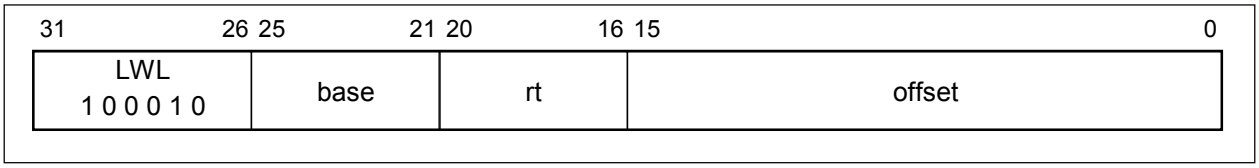
Operation:

```
32    T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
          mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
          byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
          GPR [rt] ← mem31 + 8*byte...8*byte

64    T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
          mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
          byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
          GPR [rt] ← (mem31 + 8*byte)^32 || mem31 + 8*byte...8*byte
```

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LWL Load Word Left LWL



Format:

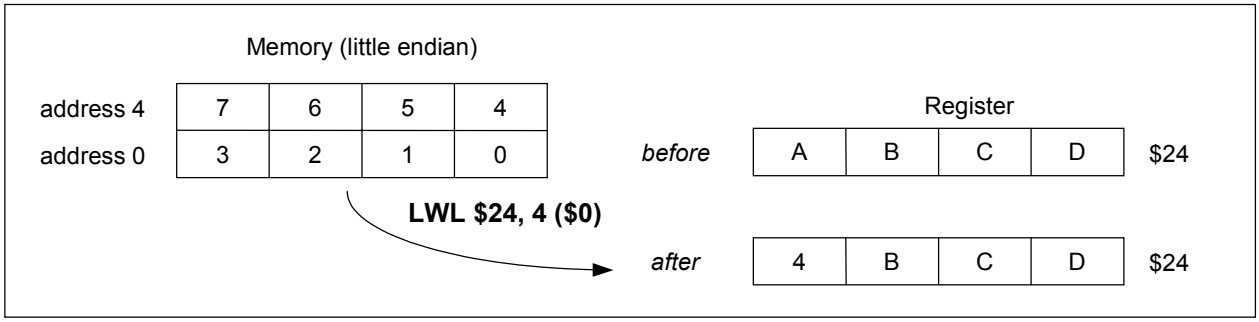
LWL rt, offset (base)

Description:

This instruction can be used in combination with the LWR instruction to load a register with four consecutive bytes from memory, when the bytes cross a word boundary. LWL loads the left portion of the register with the appropriate part of the high-order word in memory; LWR loads the right portion of the register with the appropriate part of the low-order word.

The LWL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the word in memory that contains the specified starting byte, and places them in the high-order part of general register *rt*. The contents of the remaining part of general register *rt* are retained. From one to four bytes will be loaded, depending on the starting byte specified. In 64-bit mode, the loaded word is sign-extended.

Conceptually, it starts at the specified byte in memory and loads that byte into the high-order (left-most) byte of the register; then it loads bytes from memory into the register until it reaches the low-order byte of the word in memory. The least-significant (right-most) byte(s) of the register will not be changed.



LWL Load Word Left (Continued) LWL

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LWL (or LWR) instruction which also specifies register *rt*.

No address error exceptions due to alignment are possible.

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            word ← vAddr2 xor BigEndianCPU
            mem ← LoadMemory (uncached, byte, pAddr, vAddr, DATA)
            temp ← mem32*word + 8*byte + 7...32*word || GPR [rt]23 – 8*byte...0
            GPR [rt] ← temp

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            word ← vAddr2 xor BigEndianCPU
            mem ← LoadMemory (uncached, 0 || byte, pAddr, vAddr, DATA)
            temp ← mem32*word + 8*byte + 7...32*word || GPR [rt]23 – 8*byte...0
            GPR [rt] ← (temp31)^32 || temp
```

LWL Load Word Left (Continued) LWL

Given a doubleword in a register and a doubleword in memory, the operation of LWL is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	S S S S P F G H	0	0	7	S S S S I J K L	3	4	0
1	S S S S O P G H	1	0	6	S S S S J K L H	2	4	1
2	S S S S N O P H	2	0	5	S S S S K L G H	1	4	2
3	S S S S M N O P	3	0	4	S S S S L F G H	0	4	3
4	S S S S L F G H	0	4	3	S S S S M N O P	3	0	4
5	S S S S K L G H	1	4	2	S S S S N O P H	2	0	5
6	S S S S J K L H	2	4	1	S S S S O P G H	1	0	6
7	S S S S I J K L	3	4	0	S S S S P F G H	0	0	7

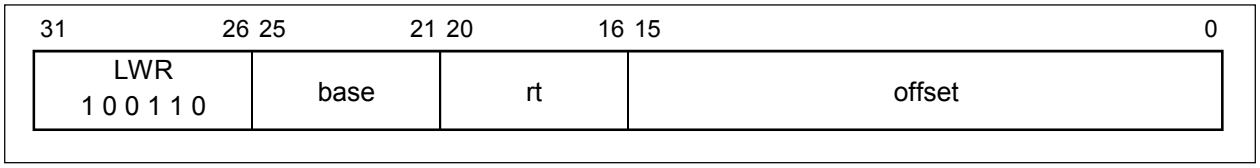
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)
S: sign-extend of $destination_{31}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LWR Load Word Right LWR



Format:

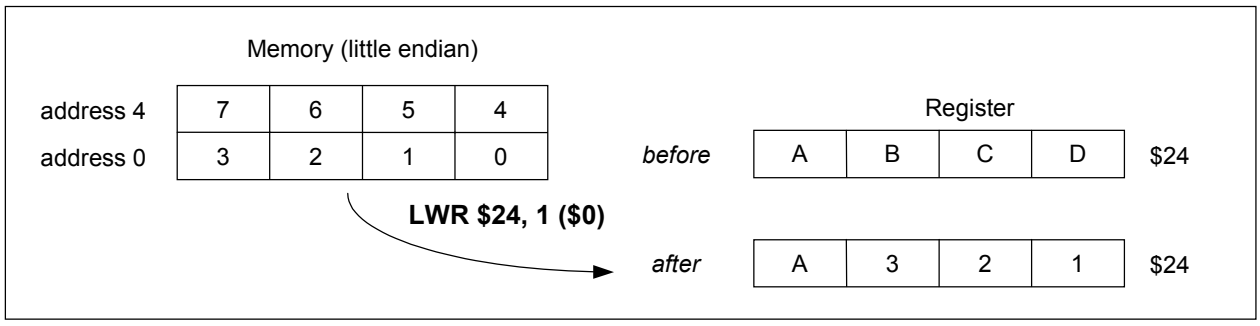
LWR rt, offset (base)

Description:

This instruction can be used in combination with the LWL instruction to load a register with four consecutive bytes from memory, when the bytes cross a word boundary. LWR loads the right portion of the register with the appropriate part of the low-order word in memory; LWL loads the left portion of the register with the appropriate part of the high-order word.

The LWR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that can specify an arbitrary byte. It reads bytes only from the word in memory that contains the specified starting byte, and places them in the low-order part of general register *rt*. The contents of the remaining part of general register *rt* are retained. From one to four bytes will be loaded, depending on the starting byte specified. In 64-bit mode, the loaded word is sign-extended.

Conceptually, it starts at the specified byte in memory and loads that byte into the low-order (right-most) byte of the register; then it loads bytes from memory into the register until it reaches the high-order byte of the word in memory. The most significant (left-most) byte(s) of the register will not be changed.



LWR Load Word Right (Continued) LWR

The contents of general register *rt* are internally bypassed within the processor so that no NOP is needed between an immediately preceding load instruction which specifies register *rt* and a following LWR (or LWL) instruction which also specifies register *rt*.

No address error exceptions due to alignment are possible.

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            word ← vAddr2 xor BigEndianCPU
            mem ← LoadMemory (uncached, 0 || byte, pAddr, vAddr, DATA)
            temp ← GPR [rt]31...32 – 8*byte || mem31 + 32*word…32*word + 8*byte
            GPR [rt] ← temp

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            word ← vAddr2 xor BigEndianCPU
            mem ← LoadMemory (uncached, WORD-byte, pAddr, vAddr, DATA)
            temp ← GPR [rt]31...32 – 8*byte || mem31 + 32*word…32*word + 8*byte
            GPR [rt] ← (temp31)^32 || temp
```

LWR Load Word Right (Continued) LWR

Given a word in a register and a word in memory, the operation of LWR is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	S S S S M N O P	3	0	4	S S S S E F G I	0	7	0
1	S S S S E M N O	2	1	4	S S S S E F I J	1	6	0
2	S S S S E F M N	1	2	4	S S S S E I J K	2	5	0
3	S S S S E F G M	0	3	4	S S S S I J K L	3	4	0
4	S S S S I J K L	3	4	0	S S S S E F G M	0	3	4
5	S S S S E I J K	2	5	0	S S S S E F M N	1	2	4
6	S S S S E F I J	1	6	0	S S S S E M N O	2	1	4
7	S S S S E F G I	0	7	0	S S S S M N O P	3	0	4

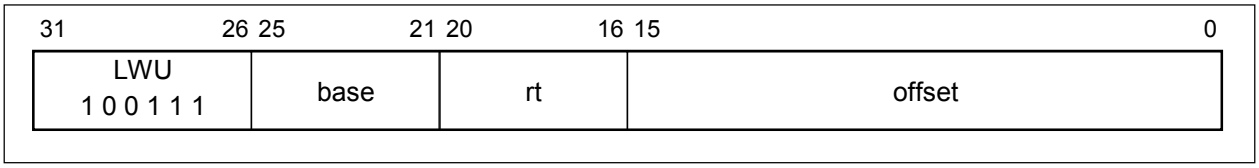
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)
S: sign-extend of $destination_{31}$

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception

LWU Load Word Unsigned LWU



Format:

LWU rt, 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 contents of the word at the memory location specified by the effective address are loaded into general register *rt*. The loaded word is zero-extended.

If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

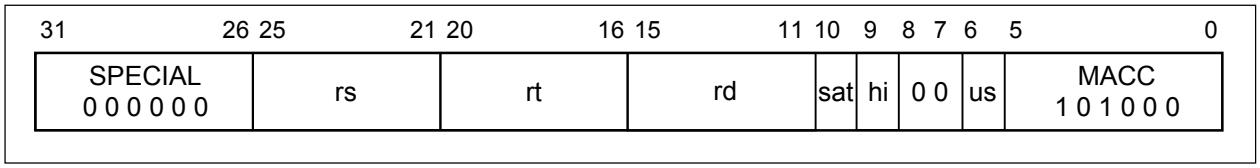
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
            mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
            GPR [rt] ← 0^32 || mem 31 + 8*byte...8*byte

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
            mem ← LoadMemory (uncached, WORD, pAddr, vAddr, DATA)
            byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
            GPR [rt] ← 0^32 || mem 31 + 8*byte...8*byte
```

Exceptions:

TLB refill exception
TLB invalid exception
Bus error exception
Address error exception
Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

MACC Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) MACC



Format:

MACC	rd, rs, rt
MACCU	rd, rs, rt
MACCHI	rd, rs, rt
MACCHIU	rd, rs, rt
MACCS	rd, rs, rt
MACCUS	rd, rs, rt
MACCHIS	rd, rs, rt
MACCHIUS	rd, rs, rt

Description:

The mnemonics of the MACC instruction differ as shown in the table below by the setting of the *sat*, *hi*, or *us* bits.

Mnemonic	sat	hi	us
MACC	0	0	0
MACCU	0	0	1
MACCHI	0	1	0
MACCHIU	0	1	1
MACCS	1	0	0
MACCUS	1	0	1
MACCHIS	1	1	0
MACCHIUS	1	1	1

The number of valid bits in the operands differs depending on whether saturation processing is executed (*sat* = 1) or not (*sat* = 0).

- **When saturation processing is executed (*sat* = 1): MACCS, MACCUS, MACCHIS, and MACCHIUS instructions**

  The contents of general register *rs* are multiplied by the contents of general register *rt*. If *us* = 1, the contents of both operands are handled as 16-bit unsigned data. If *us* = 0, the contents are handled as 16-bit signed integers. Sign/zero extension by software is required for bits 16 to 31 in the operands.

MACC Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) MACC

The product of this multiply operation is added to the 64-bit value (of which only the low-order 32 bits are valid) formed by concatenating special registers *HI* and *LO*. If *us* = 1, this add operation handles the values being added as 32-bit unsigned data. If *us* = 0, the values are handled as 32-bit signed integers. Sign/zero extension by software is required for bits 32 to 63 of the value formed by concatenating special registers *HI* and *LO*.

After saturation processing of 32 bits has been performed (refer to the table below), the sum from this add operation is loaded to special registers *HI* and *LO*. When *hi* = 1, data that is the same as the data loaded to special register *HI* is also loaded to general register *rd*. When *hi* = 0, data that is the same as the data loaded to special register *LO* is also loaded to general register *rd*. Overflow exceptions do not occur.

- **When saturation processing is not executed (*sat* = 0): MACC, MACCU, MACCHI, and MACCHIU instructions**

The contents of general register *rs* are multiplied by the contents of general register *rt*. If *us* = 1, the contents of both operands are handled as 32-bit unsigned data. If *us* = 0, the contents are handled as 32-bit signed integers. Sign/zero extension by software is required for bits 32 to 63 in the operands.

The product of this multiply operation is added to the 64-bit value formed by concatenating special registers *HI* and *LO*. If *us* = 1, this add operation handles the values being added as 64-bit unsigned data. If *us* = 0, the values are handled as 64-bit signed integers.

The low-order word of the sum from this add operation is loaded to special register *LO*, and the high-order word to special register *HI*. When *hi* = 1, data that is the same as the data loaded to special register *HI* is also loaded to general register *rd*. When *hi* = 0, data that is the same as the data loaded to special register *LO* is also loaded to general register *rd*. Overflow exceptions do not occur.

MACC Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) MACC

The correspondence of *us* and *sat* settings and values stored during saturation processing is shown below, along with the hazard cycles required between execution of the instruction for manipulating the *HI* and *LO* registers and execution of the MACC instruction.

Values Stored During Saturation Processing

us	sat	Overflow	Underflow
0	0	Store calculation result as is	Store calculation result as is
1	0	Store calculation result as is	Store calculation result as is
0	1	0x0000 0000 7FFF FFFF	0xFFFF FFFF 8000 0000
1	1	0xFFFF FFFF FFFF FFFF	None

Hazard Cycle Counts

Instruction	Cycle Count
MULT, MULTU	**Note1**
DMULT, DMULTU	3
DIV, DIVU	36
DDIV, DDIVU	68
MFHI, MFLO	**Note2**
MTHI, MTLO	0
MACC	0
DMACC	0

Notes 1. VR4121, VR4122 … 1
VR4131 … 0
VR4181A … 1

2. VR4121, VR4122 … 2
VR4131 … 0
VR4181A … 2

Operation:

```
32, sat = 0, hi = 0, us = 0 (MACC instruction)
        T:   temp1 ← GPR[rs] * GPR[rt]
             temp2 ← temp1 + (HI || LO)
             LO ← temp2[63..32]
             HI ← temp2[31..0]
             GPR[rd] ← LO
32, sat = 0, hi = 0, us = 1 (MACCU instruction)
        T:   temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
             temp2 ← temp1 + ((0 || HI) || (0 || LO))
             LO ← temp2[63..32]
             HI ← temp2[31..0]
             GPR[rd] ← LO
```

MACC Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) MACC

```
32, sat = 0, hi = 1, us = 0 (MACCHI instruction)
        T:    temp1 ← GPR[rs] * GPR[rt]
              temp2 ← temp1 + (HI || LO)
              LO ← temp2 63..32
              HI ← temp2 31..0
              GPR[rd] ← HI
32, sat = 0, hi = 1, us = 1 (MACCHIU instruction)
        T:    temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
              temp2 ← temp1 + ((0 || HI) || (0 || LO))
              LO ← temp2 63..32
              HI ← temp2 31..0
              GPR[rd] ← HI
32, sat = 1, hi = 0, us = 0 (MACCS instruction)
        T:    temp1 ← GPR[rs] * GPR[rt]
              temp2 ← saturation(temp1 + (HI || LO))
              LO ← temp2 63..32
              HI ← temp2 31..0
              GPR[rd] ← LO
32, sat = 1, hi = 0, us = 1 (MACCUS instruction)
        T:    temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
              temp2 ← saturation(temp1 + ((0 || HI) || (0 || LO)))
              LO ← temp2 63..32
              HI ← temp2 31..0
              GPR[rd] ← LO
32, sat = 1, hi = 1, us = 0 (MACCHIS instruction)
        T:    temp1 ← GPR[rs] * GPR[rt]
              temp2 ← saturation(temp1 + (HI || LO))
              LO ← temp2 63..32
              HI ← temp2 31..0
              GPR[rd] ← HI
32, sat = 1, hi = 1, us = 1 (MACCHIUS instruction)
        T:    temp1 ← (0 || GPR[rs]) * (0 || GPR[rt])
              temp2 ← saturation(temp1 + ((0 || HI) || (0 || LO)))
              LO ← temp2 63..32
              HI ← temp2 31..0
              GPR[rd] ← HI
```

MACC Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) MACC

64, sat = 0, hi = 0, us = 0 (MACC instruction)

T: temp1 ← ((GPR[rs]$_{31}$)^{32}|| GPR[rs]) * ((GPR[rt]$_{31}$)^{32}|| GPR[rt])
temp2 ← temp1 + (HI$_{31..0}$ || LO$_{31..0}$)
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← LO

64, sat = 0, hi = 0, us = 1 (MACCU instruction)

T: temp1 ← (0^{32}|| GPR[rs]) * (0^{32}|| GPR[rt])
temp2 ← temp1 + (HI$_{31..0}$ || LO$_{31..0}$)
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← LO

64, sat = 0, hi = 1, us = 0 (MACCHI instruction)

T: temp1 ← ((GPR[rs]$_{31}$)^{32}|| GPR[rs]) * ((GPR[rt]$_{31}$)^{32}|| GPR[rt])
temp2 ←temp1 + (HI$_{31..0}$ || LO$_{31..0}$)
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← HI

64, sat = 0, hi = 1, us = 1 (MACCHIU instruction)

T: temp1 ← (0^{32}|| GPR[rs]) * (0^{32}|| GPR[rt])
temp2 ← temp1 + (HI$_{31..0}$ || LO$_{31..0}$)
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← HI

64, sat = 1, hi = 0, us = 0 (MACCS instruction)

T: temp1 ← ((GPR[rs]$_{31}$)^{32}|| GPR[rs]) * ((GPR[rt]$_{31}$)^{32}|| GPR[rt])
temp2 ← saturation(temp1 + (HI$_{31..0}$ || LO$_{31..0}$))
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← LO

64, sat = 1, hi = 0, us = 1 (MACCUS instruction)

T: temp1 ← (0^{32}|| GPR[rs]) * (0^{32}|| GPR[rt])
temp2 ←saturation(temp1 + (HI$_{31..0}$ || LO$_{31..0}$))
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← LO

MACC Multiply and Add Accumulate (for VR4121, VR4122, VR4131, and VR4181A) (Continued) MACC

64, sat = 1, hi = 1, us = 0 (MACCHIS instruction)

T: temp1 ← ((GPR[rs]$_{31}$)^{32}|| GPR[rs]) * ((GPR[rt]$_{31}$)^{32}|| GPR[rt])
temp2 ← saturation(temp1 + (HI$_{31..0}$ || LO$_{31..0}$))
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← HI

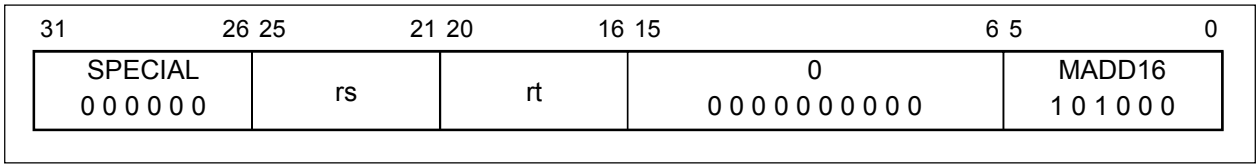
64, sat = 1, hi = 1, us = 1 (MACCHIUS instruction)

T: temp1 ← (0^{32}|| GPR[rs]) * (0^{32}|| GPR[rt])
temp2 ← saturation(temp1 + (HI$_{31..0}$ || LO$_{31..0}$)
LO ← ((temp2$_{63}$)^{32}|| temp2$_{63..32}$)
HI ← ((temp2$_{31}$)^{32}|| temp2$_{31..0}$)
GPR[rd] ← HI

Exceptions:

None

MADD16 Multiply and Add 16-bit integer (for VR4181 only) MADD16



Format:

MADD16 rs, rt

Description:

The contents of general registers *rs* and *rt* are multiplied, treating both operands as 16-bit 2's complement values. Bits 62 to 15 of the operand must be valid sign-extended values. If not, the result is unpredictable.

This multiplied result and the 64-bit data joined special register *HI* to *LO* are added to form the result. When the operation completes, the low-order word of the result is loaded into special register *LO*, and the high-order word of the result is loaded into special register *HI*.

No integer overflow exception occurs under any circumstances.

Hazard cycles required between MADD16 and other instructions are as follows.

Instruction sequence	No. of cycles
MULT/MULTU → MADD16	1 Cycle
DMULT/DMULTU → MADD16	4 Cycles
DIV/DIVU → MADD16	36 Cycles
DDIV/DDIVU → MADD16	68 Cycles
MFHI/MFLO → MADD16	2 Cycles
DMADD16 → MADD16	0 Cycles
MADD16 → MADD16	0 Cycles

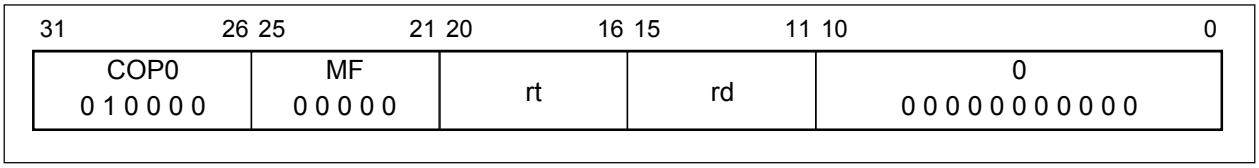
Operation:

```
32, 64   T:   temp1 ← GPR [rs] * GPR [rt]
              temp2 ← temp1 + (HI31...0 || LO31...0)
              LO  ← (temp2 31)^32 || temp2 31...0
              HI  ← (temp2 63)^32 || temp2 63...32
```

Exceptions:

None

MFC0 Move from System Control Coprocessor MFC0



Format:

MFC0 rt, rd

Description:

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

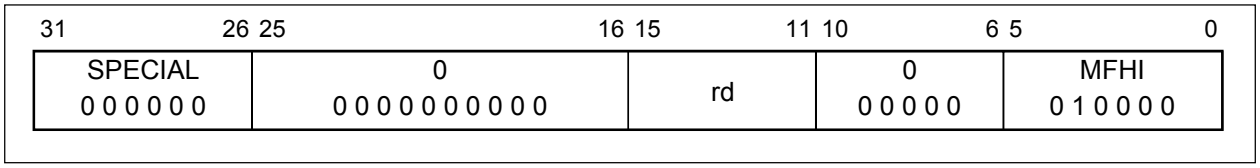
Operation:

```
32      T:    data ← CPR [0, rd]
        T+1:  GPR [rt] ← data

64      T:    data ← CPR [0, rd]
        T+1:  GPR [rt] ← (data31)^32 || data31...0
```

Exceptions:

Coprocessor unusable exception (User and Supervisor mode if CP0 not enabled)

MFHI — Move from HI — MFHI



31 26	25 16	15 11	10 6	5 0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	rd	0 0 0 0 0 0	MFHI 0 1 0 0 0 0

Format:

MFHI rd

Description:

The contents of special register *HI* are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions which follow a MFHI instruction may not be any of the instructions which modify the *HI* register: MACC, DMACC, MADD16, DMADD16, MULT, MULTU, DIV, DIVU, MTHI, DMULT, DMULTU, DDIV, DDIVU.

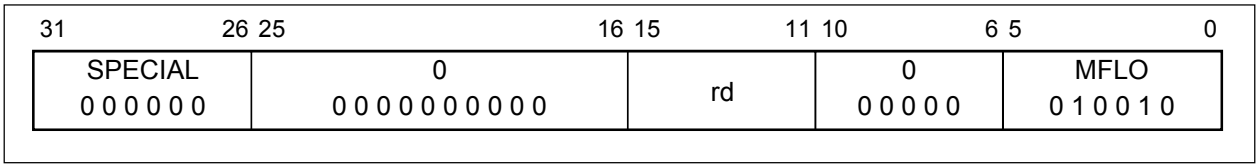
Operation:

```
32, 64   T:    GPR [rd] ← HI
```

Exceptions:

None

MFLO Move from LO MFLO



Format:

MFLO rd

Description:

The contents of special register *LO* are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions which follow a MFLO instruction may not be any of the instructions which modify the *LO* register: MACC, DMACC, MADD16, DMADD16, MULT, MULTU, DIV, DIVU, MTLO, DMULT, DMULTU, DDIV, DDIVU.

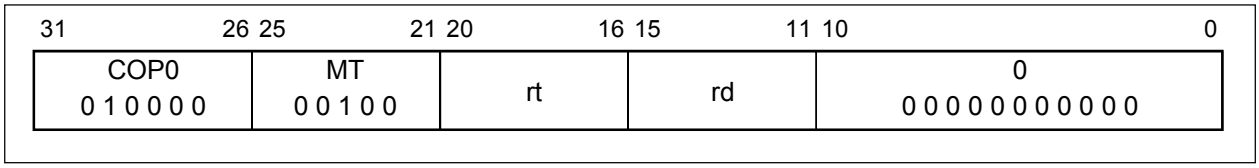
Operation:

```
32, 64   T:    GPR [rd] ← LO
```

Exceptions:

None

MTC0 Move to Coprocessor0 MTC0



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.

When using a register used by the MTC0 by means of instructions before and after it, refer to **CHAPTER 11 COPROCESSOR 0 HAZARDS** and place the instructions in the appropriate location.

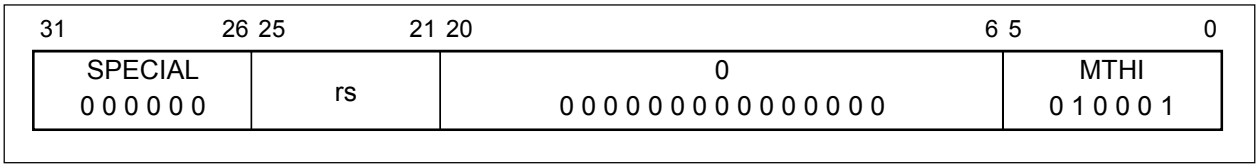
Operation:

```
32, 64  T:    data ← GPR [rt]
        T+1:  CPR [0, rd] ← data
```

Exceptions:

Coprocessor unusable exception (User and Supervisor mode if CP0 not enabled)

MTHI — Move to HI — MTHI



Format:

MTHI rs

Description:

The contents of general register *rs* are loaded into special register *HI*.

Restrictions:

The operation results written to the *HI*/*LO* register pair via a DDIV, DDIVU, DIV, DIVU, DMULT, DMULTU, MULT, or MULTU instruction should be read by the MFHI or MFLO instruction before another result is written to either of the registers. If the MTHI instruction is executed prior to the MFLO or MFHI instruction following the execution of any one of the arithmetic instructions, the contents of the *LO* register are undefined as shown in the example below.

```
MULT  r2, r4  # start operation that will eventually write to HI, LO
 …            # code not containing MFHI or MFLO
MTHI  r6
 …            # code not containing MFLO
MFLO  r3      # this MFLO would get an undefined value
```

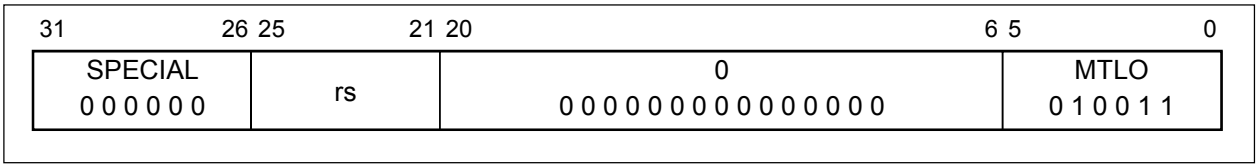
Operation:

```
32, 64  T–2: HI  ← undefined
        T–1: HI  ← undefined
        T:   HI  ← GPR [rs]
```

Exceptions:

None

MTLO Move to LO MTLO



Format:

MTLO rs

Description:

The contents of general register *rs* are loaded into special register *LO.*

Restrictions:

The operation results written to the *HI*/*LO* register pair via a DDIV, DDIVU, DIV, DIVU, DMULT, DMULTU, MULT, or MULTU instruction should be read by the MFHI or MFLO instruction before another result is written to either of the registers. If the MTLO instruction is executed prior to the MFLO or MFHI instruction following the execution of any one of the arithmetic instructions, the contents of the *HI* register are undefined as shown in the example below.

```
MULT  r2, r4  # start operation that will eventually write to HI, LO
 …            # code not containing MFHI or MFLO
MTLO  r6
 …            # code not containing MFHI
MFHI  r3      # this MFHI would get an undefined value
```

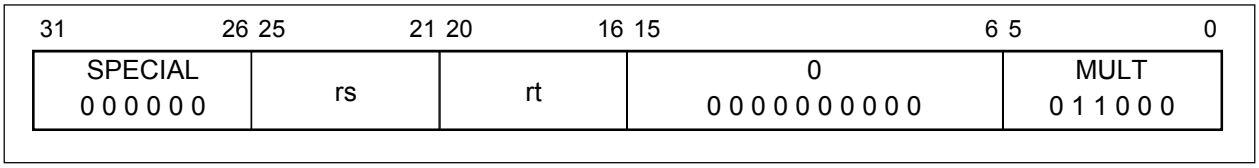
Operation:

```
32, 64  T–2: LO ← undefined
        T–1: LO ← undefined
        T:   LO ← GPR [rs]
```

Exceptions:

None

MULT — Multiply — MULT



Format:

MULT rs, rt

Description:

The contents of general registers *rs* and *rt* are multiplied, treating both operands as signed 32-bit integer. No integer overflow exception occurs under any circumstances. In 64-bit mode, the operands must be valid 32-bit, sign-extended values.

When the operation completes, the low-order doubleword of the result is loaded into special register *LO*, and the high-order doubleword of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two other instructions.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

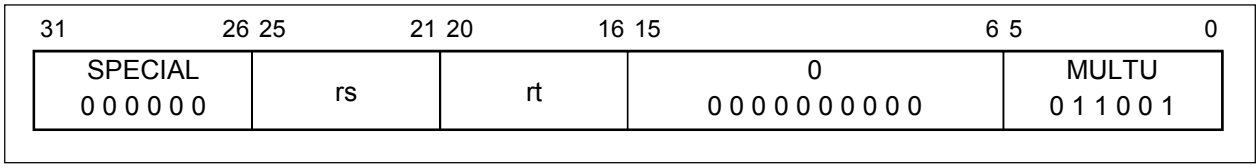
Operation:

```
32      T−2: LO ← undefined
             HI ← undefined
        T−1: LO ← undefined
             HI ← undefined
        T:   t  ← GPR [rs] * GPR [rt]
             LO ← t 31...0
             HI ← t 63...32

64      T−2: LO ← undefined
             HI ← undefined
        T−1: LO ← undefined
             HI ← undefined
        T:   t  ← GPR [rs] 31...0 * GPR [rt] 31...0
             LO ← (t 31)^32 || t 31...0
             HI ← (t 63)^32 || t 63...32
```

Exceptions:

None

MULTU Multiply Unsigned MULTU



Format:

MULTU rs, rt

Description:

The contents of general registers *rs* and *rt* are multiplied, treating both operands as unsigned values. No overflow exception occurs under any circumstances. In 64-bit mode, the operands must be valid 32-bit, sign-extended values.

When the operation completes, the low-order doubleword of the result is loaded into special register *LO*, and the high-order doubleword of the result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two instructions.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

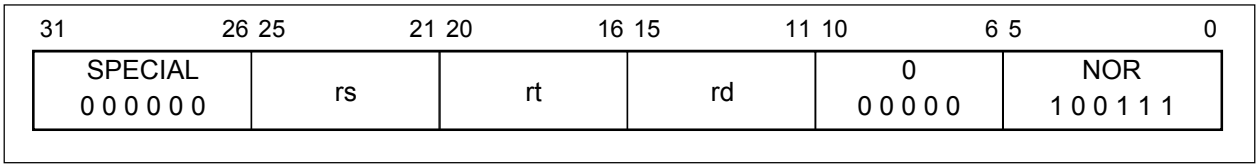
Operation:

```
32      T–2: LO ← undefined
             HI ← undefined
        T–1: LO ← undefined
             HI ← undefined
        T:   t  ← (0 || GPR [rs]) * (0 || GPR [rt])
             LO ← t 31...0
             HI ← t 63...32

64      T–2: LO ← undefined
             HI ← undefined
        T–1: LO ← undefined
             HI ← undefined
        T:   t  ← (0 || GPR [rs] 31...0 ) * (0 || GPR [rt] 31...0)
             LO ← (t 31)^32 || t 31...0
             HI ← (t 63)^32 || t 63...32
```

Exceptions:

None

NOR NOR NOR



Format:

NOR rd, rs, rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical NOR operation. The result is placed into general register *rd*.

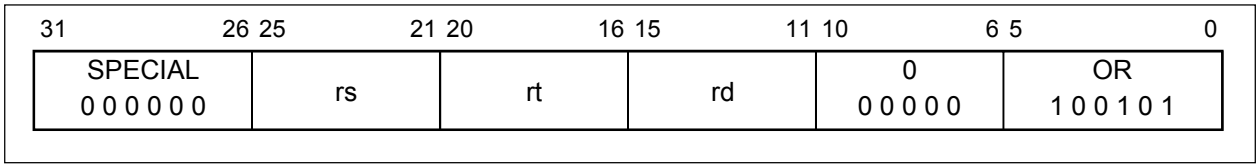
Operation:

```
32, 64   T:     GPR [rd] ← GPR [rs] nor GPR [rt]
```

Exceptions:

None

OR OR OR



Format:

OR rd, rs, rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical OR operation. The result is placed into general register *rd*.

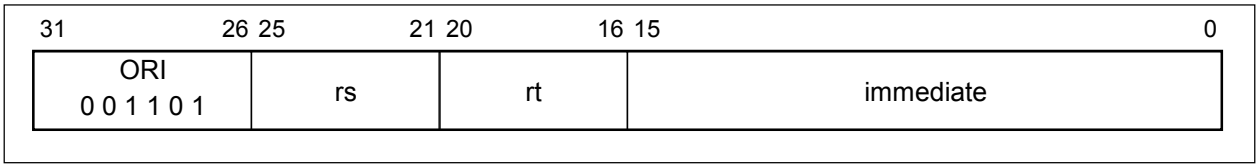
Operation:

```
32, 64   T:    GPR [rd] ← GPR [rs] or GPR [rt]
```

Exceptions:

None

ORI OR Immediate ORI



Format:

ORI rt, rs, immediate

Description:

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical OR operation. The result is placed into general register *rt*.

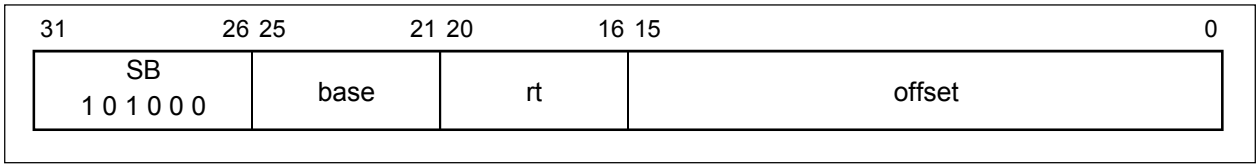
Operation:

32	T:	GPR [rt] ← GPR [rs] $_{31...16}$ || (immediate or GPR [rs] $_{15...0}$)
64	T:	GPR [rt] ← GPR [rs] $_{63...16}$ || (immediate or GPR [rs] $_{15...0}$)

Exceptions:

None

SB Store Byte SB



Format:

SB rt, 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 least-significant byte of register *rt* is stored at the effective address.

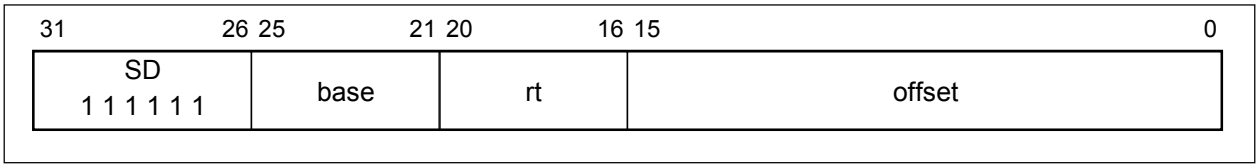
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← GPR [rt]63 – 8*byte...0 || 0^8*byte
            StoreMemory (uncached, BYTE, data, pAddr, vAddr, DATA)

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← GPR [rt]63 – 8*byte...0 || 0^8*byte
            StoreMemory (uncached, BYTE, data, pAddr, vAddr, DATA)
```

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception

SD Store Doubleword SD



Format:

SD rt, 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 contents of general register *rt* are stored at the memory location specified by the effective address.

If either of the three least-significant bits of the effective address are non-zero, an address error exception occurs.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

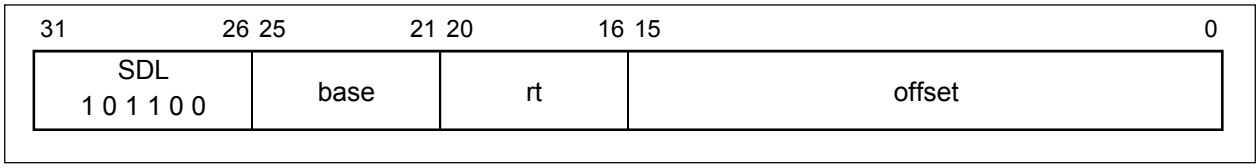
Operation:

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           data ← GPR [rt]
           StoreMemory (uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           data ← GPR [rt]
           StoreMemory (uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)
```

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception
Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

SDL Store Doubleword Left SDL



Format:

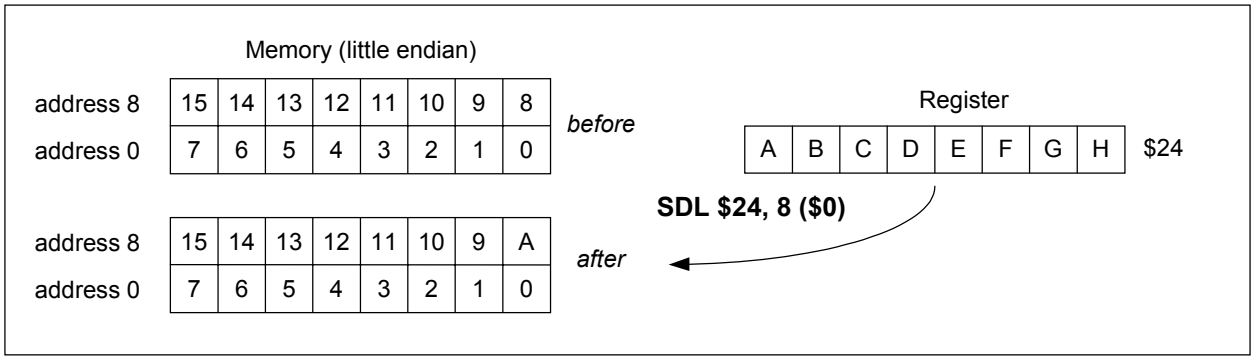
SDL rt, offset (base)

Description:

This instruction can be used with the SDR instruction to store the contents of a register into eight consecutive bytes of memory, when the bytes cross a doubleword boundary. SDL stores the left portion of the register into the appropriate part of the high-order doubleword in memory; SDR stores the right portion of the register into the appropriate part of the low-order doubleword.

The SDL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the doubleword in memory that contains the specified starting byte, with the high-order part of general register *rt*. From one to eight bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the most-significant (leftmost) byte of the register and copies it to the specified byte in memory; then it copies bytes from register to memory until it reaches the low-order byte of the doubleword in memory.



SDL Store Doubleword Left (Continued) SDL

No address error exceptions due to alignment are possible.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← 0^56 – 8*byte || GPR [rt]63...56 – 8*byte
            StoreMemory (uncached, byte, data, pAddr, vAddr, DATA)

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← 0^56 – 8*byte || GPR [rt]63...56 – 8*byte
            StoreMemory (uncached, byte, data, pAddr, vAddr, DATA)
```

SDL Store Doubleword Left (Continued) SDL

Given a doubleword in a register and a doubleword in memory, the operation of SDL is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	I J K L M N O A	0	0	7	A B C D E F G H	7	0	0
1	I J K L M N A B	1	0	6	I A B C D E F G	6	0	1
2	I J K L M A B C	2	0	5	I J A B C D E F	5	0	2
3	I J K L A B C D	3	0	4	I J K A B C D E	4	0	3
4	I J K A B C D E	4	0	3	I J K L A B C D	3	0	4
5	I J A B C D E F	5	0	2	I J K L M A B C	2	0	5
6	I A B C D E F G	6	0	1	I J K L M N A B	1	0	6
7	A B C D E F G H	7	0	0	I J K L M N O A	0	0	7

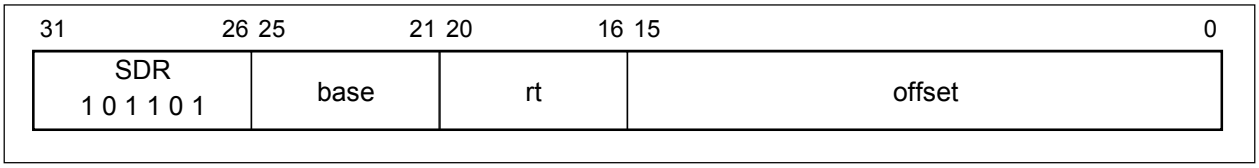
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception
Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

SDR Store Doubleword Right SDR



Format:

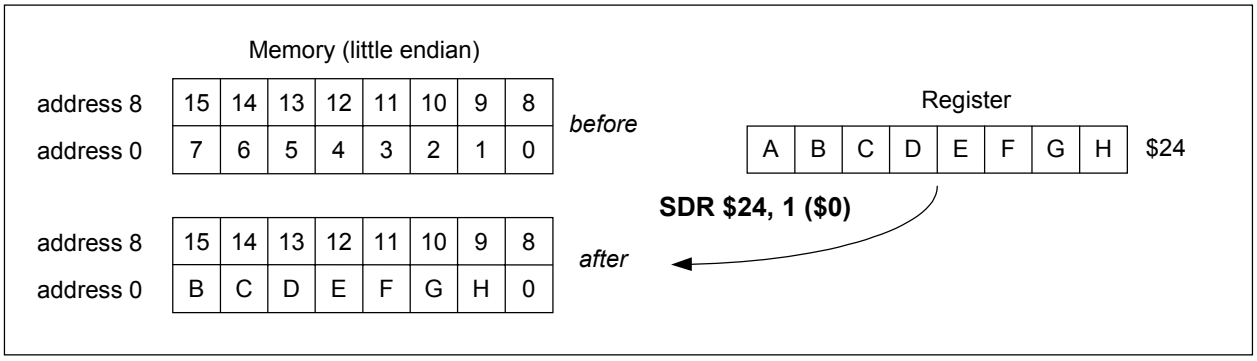
SDR rt, offset (base)

Description:

This instruction can be used with the SDL instruction to store the contents of a register into eight consecutive bytes of memory, when the bytes cross a doubleword boundary. SDR stores the right portion of the register into the appropriate part of the low-order doubleword in memory; SDL stores the left portion of the register into the appropriate part of the high-order doubleword.

The SDR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the doubleword in memory that contains the specified starting byte, with the low-order part of general register *rt*. From one to eight bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the least-significant (rightmost) byte of the register and copies it to the specified byte in memory; then it copies bytes from register to memory until it reaches the high-order byte of the doubleword in memory.



SDR Store Doubleword Right (Continued) SDR

No address error exceptions due to alignment are possible.

This operation is defined for the VR4100 Series operating in 64-bit mode or in 32-bit Kernel mode. Execution of this instruction in 32-bit User or Supervisor mode causes a reserved instruction exception.

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← GPR [rt]63 – 8*byte || 0^8*byte
            StoreMemory (uncached, DOUBLEWORD-byte, data, pAddr, vAddr, DATA)

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...3 || 0^3
            endif
            byte ← vAddr2...0 xor BigEndianCPU^3
            data ← GPR [rt]63 – 8*byte || 0^8*byte
            StoreMemory (uncached, DOUBLEWORD-byte, data, pAddr, vAddr, DATA)
```

SDR Store Doubleword Right (Continued) SDR

Given a doubleword in a register and a doubleword in memory, the operation of SDR is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	A B C D E F G H	7	0	0	I J K L M N O A	0	7	0
1	B C D E F G H P	6	1	0	I J K L M N A B	1	6	0
2	C D E F G H O P	5	2	0	I J K L M A B C	2	5	0
3	D E F G H N O P	4	3	0	I J K L A B C D	3	4	0
4	E F G H M N O P	3	4	0	I J K A B C D E	4	3	0
5	F G H L M N O P	2	5	0	I J A B C D E F	5	2	0
6	G H K L M N O P	1	6	0	I A B C D E F G	6	1	0
7	H J K L M N O P	0	7	0	A B C D E F G H	7	0	0

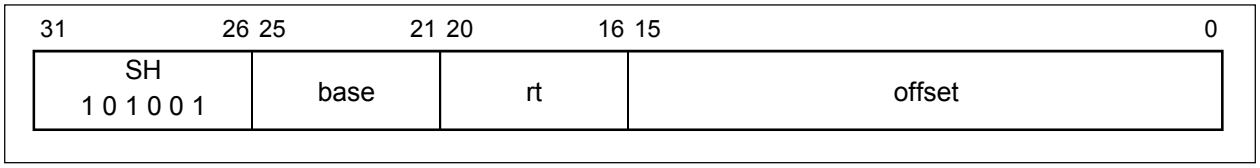
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception
Reserved instruction exception (VR4100 Series in 32-bit User mode, VR4100 Series in 32-bit Supervisor mode)

SH Store Halfword SH



Format:

SH rt, offset (base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form an effective address. The least-significant halfword of register *rt* is stored at the effective address. If the least-significant bit of the effective address is non-zero, an address error exception occurs.

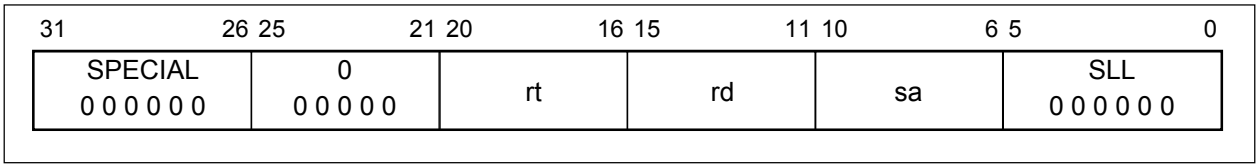
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
            byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
            data ← GPR [rt]63 – 8*byte...0 || 0^8*byte
            StoreMemory (uncached, HALFWORD, data, pAddr, vAddr, DATA)

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
            byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
            data ← GPR [rt]63 – 8*byte...0 || 0^8*byte
            StoreMemory (uncached, HALFWORD, data, pAddr, vAddr, DATA)
```

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception

SLL Shift Left Logical SLL



Format:

SLL rd, rt, sa

Description:

The contents of general register *rt* are shifted left by *sa* bits, inserting zeros into the low-order bits. The result is placed in register *rd.*

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register. It is sign extended for all shift amounts, including zero; SLL with zero shift amount truncates a 64-bit value to 32 bits and then sign extends this 32-bit value. SLL, unlike nearly all other word operations, does not require an operand to be a properly sign-extended word value to produce a valid sign-extended word result.

Operation:

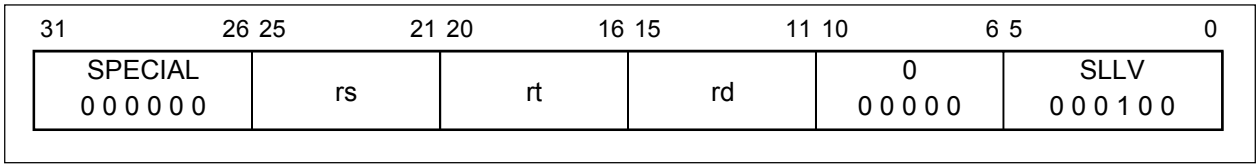
```
32    T:   GPR [rd] ← GPR [rt] 31 – sa...0 || 0^sa

64    T:   s ← 0 || sa
           temp ← GPR [rt] 31 – s...0) || 0^s
           GPR [rd] ← (temp31)^32 || temp
```

Exceptions:

None

Caution SLL with a shift amount of zero may be treated as a NOP by some assemblers, at some optimization levels. If using SLL with a zero shift to truncate 64-bit values, check the assembler you are using.

SLLV Shift Left Logical Variable SLLV



Format:

SLLV rd, rt, rs

Description:

The contents of general register *rt* are shifted left the number of bits specified by the low-order five bits contained in general register *rs*, inserting zeros into the low-order bits. The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register. It is sign extended for all shift amounts, including zero; SLLV with zero shift amount truncates a 64-bit value to 32 bits and then sign extends this 32-bit value. SLLV, unlike nearly all other word operations, does not require an operand to be a properly sign-extended word value to produce a valid sign-extended word result.

Operation:

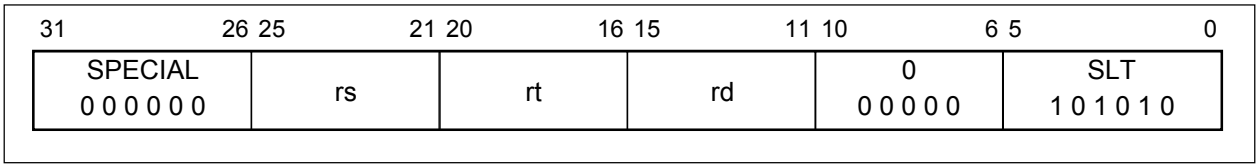
```
32    T:   s ← GPR [rs] 4...0
           GPR [rd] ← GPR [rt] 31 – s...0 || 0^s

64    T:   s ← 0 || GPR [rs] 4...0
           temp ← GPR [rt] 31 – s...0) || 0^s
           GPR [rd] ← (temp31)^32 || temp
```

Exceptions:

None

Caution SLLV with a shift amount of zero may be treated as a NOP by some assemblers, at some optimization levels. If using SLLV with a zero shift to truncate 64-bit values, check the assembler you are using.

SLT Set on Less Than SLT



Format:

SLT rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to one; otherwise the result is set to zero. The result is placed into general register *rd*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

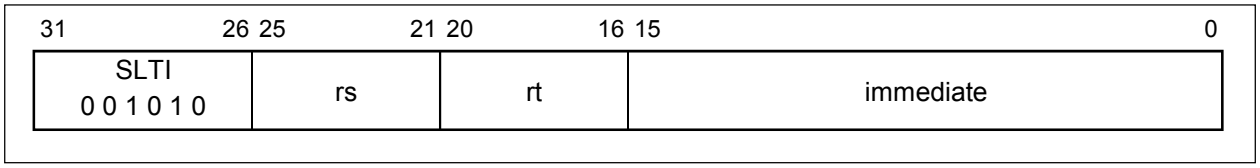
Operation:

```
32      T:  if GPR [rs] < GPR [rt] then
                GPR [rd] ← 0^31 || 1
            else
                GPR [rd] ← 0^32
            endif

64      T:  if GPR [rs] < GPR [rt] then
                GPR [rd] ← 0^63 || 1
            else
                GPR [rd] ← 0^64
            endif
```

Exceptions:

None

SLTI Set on Less Than Immediate SLTI



Format:

SLTI rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and subtracted from the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the sign-extended immediate, the result is set to 1; otherwise the result is set to 0. The result is placed into general register *rt*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

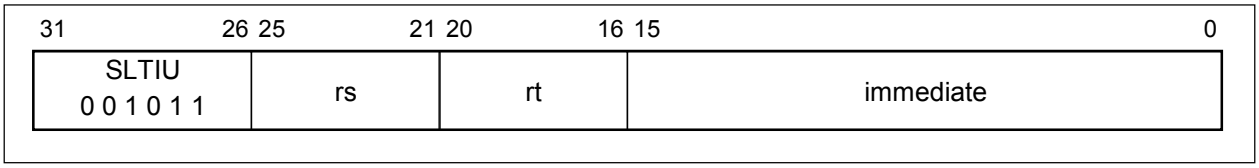
Operation:

```
32      T:  if GPR [rs] < (immediate15)^16 || immediate15...0 then
                GPR [rt] ← 0^31 || 1
            else
                GPR [rt] ← 0^32
            endif

64      T:  if GPR [rs] < (immediate15)^48 || immediate15...0 then
                GPR [rt] ← 0^63 || 1
            else
                GPR [rt] ← 0^64
            endif
```

Exceptions:

None

SLTIU Set on Less Than Immediate Unsigned SLTIU



Format:

SLTIU rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and subtracted from the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the sign-extended immediate, the result is set to 1; otherwise the result is set to 0. The result is placed into general register *rt*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

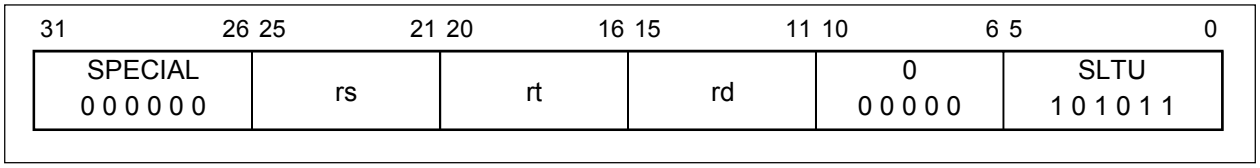
Operation:

```
32      T:  if (0 || GPR [rs]) < (0 || (immediate15)^16 || immediate15...0) then
                GPR [rt] ← 0^31 || 1
            else
                GPR [rt] ← 0^32
            endif

64      T:  if (0 || GPR [rs]) < (0 || (immediate15)^48 || immediate15...0) then
                GPR [rt] ← 0^63 || 1
            else
                GPR [rt] ← 0^64
            endif
```

Exceptions:

None

SLTU Set on Less Than Unsigned SLTU



Format:

SLTU rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs.* Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the contents of general register *rt*, the result is set to 1; otherwise the result is set to 0. The result is placed into general register *rd*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

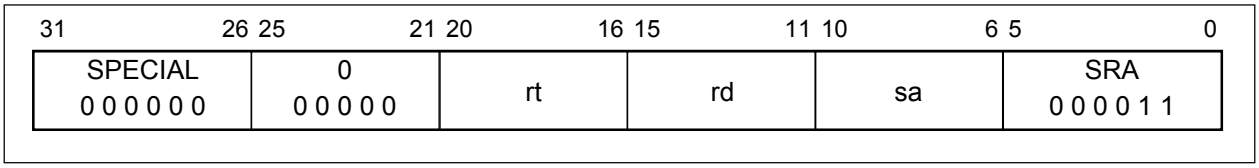
Operation:

```
32      T:  if (0 || GPR [rs]) < (0 || GPR [rt]) then
                GPR [rd] ← 0^31 || 1
            else
                GPR [rd] ← 0^32
            endif

64      T:  if (0 || GPR [rs]) < (0 || GPR [rt]) then
                GPR [rd] ← 0^63 || 1
            else
                GPR [rd] ← 0^64
            endif
```

Exceptions:

None

SRA Shift Right Arithmetic SRA



Format:

SRA rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, sign-extending the high-order bits. The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register.

Restrictions:

If the value of general register *rt* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

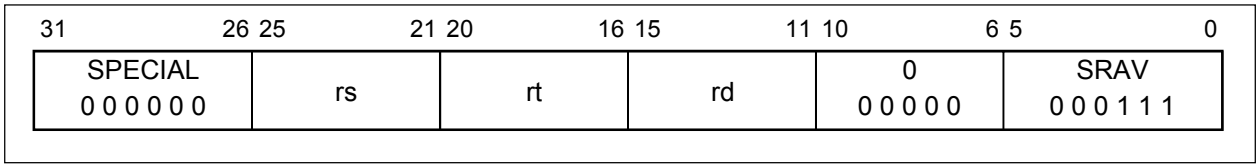
Operation:

```
32    T:   GPR [rd] ← (GPR [rt] 31)^sa || GPR [rt] 31...sa

64    T:   s ← 0 || sa
           temp ← (GPR [rt] 31)^s || GPR [rt] 31...s
           GPR [rd] ← (temp31)^32 || temp
```

Exceptions:

None

SRAV Shift Right Arithmetic Variable SRAV



Format:

SRAV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of general register *rs*, sign-extending the high-order bits. The result is placed in register *rd*.
In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register.

Restrictions:

If the value of general register *rt* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

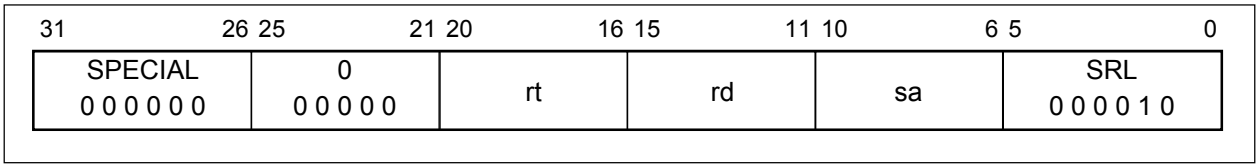
Operation:

```
32      T:  s ← GPR [rs] 4...0
            GPR [rd] ← (GPR [rt] 31)^s || GPR [rt] 31...s

64      T:  s ← GPR [rs] 4...0
            temp ← (GPR [rt] 31)^s || GPR [rt] 31...s
            GPR [rd] ← (temp31)^32 || temp
```

Exceptions:

None

SRL Shift Right Logical SRL



Format:

SRL rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, inserting zeros into the high-order bits. The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register.

Restrictions:

If the value of general register *rt* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

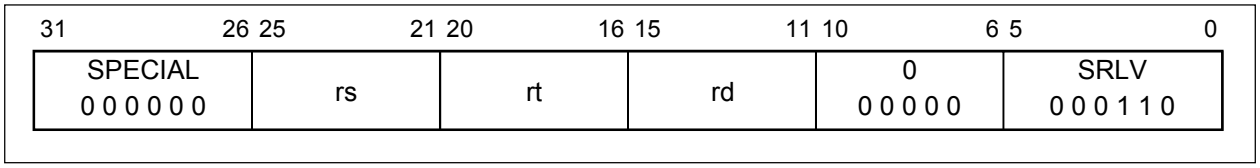
Operation:

```
32      T:   GPR [rd] ← 0^sa || GPR [rt] 31...sa

64      T:   s ← 0 || sa
             temp ← 0^s || GPR [rt] 31...s
             GPR [rd] ← (temp31)^32 || temp
```

Exceptions:

None

SRLV Shift Right Logical Variable SRLV



Format:

SRLV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of general register *rs,* inserting zeros into the high-order bits. The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register.

Restrictions:

If the value of general register *rt* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

Operation:

```
32    T:   s ← GPR [rs] 4...0
           GPR [rd] ← 0^s || GPR [rt] 31...s

64    T:   s ← GPR [rs] 4...0
           temp ← 0^s || GPR [rt] 31...s
           GPR [rd] ← (temp31)^32 || temp
```

Exceptions:

None

STANDBY Standby STANDBY

31 26	25 24	6 5	0
COP0	CO	0	STANDBY
0 1 0 0 0 0	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 1

Format:

STANDBY

Description:

STANDBY instruction starts mode transition from Fullspeed mode to Standby mode.

When the STANDBY instruction finishes the WB stage, the VR4100 Series wait by the SysAD bus is idle state, and then fix the internal clocks to high level, thus freezing the pipeline. In the VR4131 and VR4181A, IE bit of the Status register in the CP0 is also set to 1.

The PLL, Timer/Interrupt clocks and the internal bus clocks (TClock and MasterOut) will continue to run.

Once the VR4100 Series is in Standby mode, any interrupt, including the internally generated timer interrupt, NMI, Soft Reset, and Cold Reset will cause the VR4100 Series to exit Standby mode and to enter Fullspeed mode.

Operation:

```
32, 64  T:
        T+1: Standby operation ( )
```

Exceptions:

Coprocessor unusable exception

Remark Refer to **Hardware User's Manual** of each product for details about the operation of the peripheral units at mode transition.

Program examples to enter Standby mode are shown below.

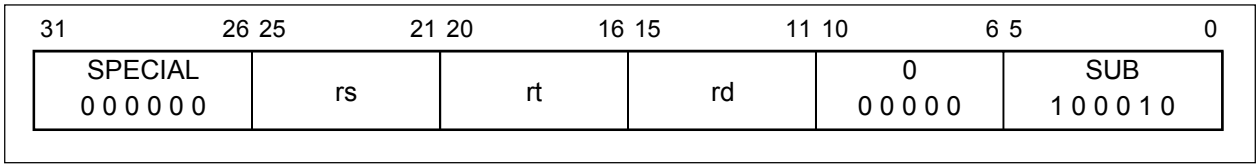
• For VR4121, VR4122, and VR4181

```
# Insert process to mask interrupts in the Interrupt Control Unit (ICU)
 …
# Insert process for entering Standby mode
 …
# Insert process to enable interrupts in the ICU
STANDBY
```

• For VR4131 and VR4181A

```
MFC0  t5, psr
ORI   t5, t5, 1
XORI  t5, t5, 1
MTC0  t5, psr
# Insert process for entering Standby mode
STANDBY
```

SUB Subtract SUB



Format:

SUB rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd*.

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register.

An integer overflow exception takes place if the carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

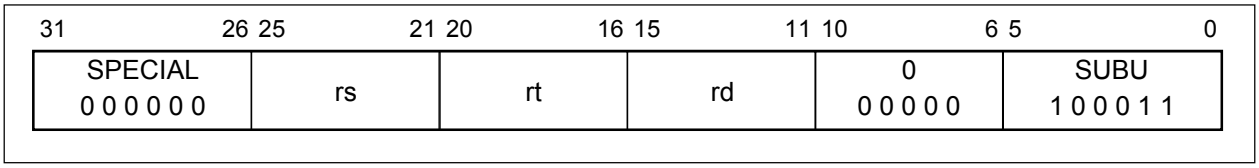
Operation:

```
32      T:   GPR [rd] ← GPR [rs] – GPR [rt]

64      T:   temp ← GPR [rs] – GPR [rt]
             GPR [rd] ← (temp31)^32 || temp31...0
```

Exceptions:

Integer overflow exception

SUBU Subtract Unsigned SUBU



Format:

SUBU rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd*.

In 64-bit mode, the 32-bit result is sign-extended when placed in the destination register.

The only difference between this instruction and the SUB instruction is that SUBU never traps on overflow. No integer overflow exception occurs under any circumstances.

Restrictions:

If the value of either general register *rt* or general register *rs* is not a sign-extended 32-bit value (bits 63 to 31 have the same value), the result of this operation will be undefined.

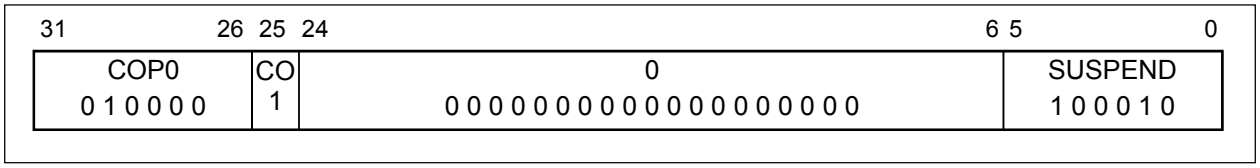
Operation:

```
32      T:   GPR [rd] ← GPR [rs] – GPR [rt]

64      T:   temp ← GPR [rs] – GPR [rt]
             GPR [rd] ← (temp31)^32 || temp31...0
```

Exceptions:

None

SUSPEND Suspend SUSPEND



Format:

SUSPEND

Description:

SUSPEND instruction starts mode transition from Fullspeed mode to Suspend mode.

When the SUSPEND instruction finishes the WB stage, the VR4100 Series wait by the SysAD bus is idle state, and then fix the internal clocks including the TClock to high level, thus freezing the pipeline. In the VR4131 and VR4181A, IE bit of the Status register in the CP0 is also set to 1.

The PLL, Timer/Interrupt clocks and MasterOut, will continue to run.

Once the VR4100 Series is in Suspend mode, any interrupt, including the internally generated timer interrupt, NMI, Soft Reset and Cold Reset will cause the VR4100 Series to exit Suspend mode and to enter Fullspeed mode.

Operation:

```
32, 64   T:
         T+1: Suspend Operation ( )
```

Exceptions:

Coprocessor unusable exception

Remark Refer to **Hardware User's Manual** of each product for details about the operation of the peripheral units at mode transition.

Program examples to enter Suspend mode are shown below.

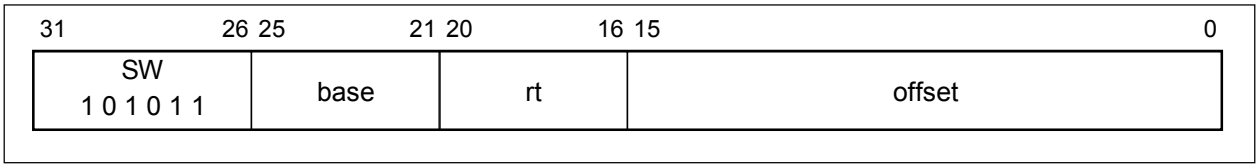
• For VR4121, VR4122, and VR4181

```
# Insert process to mask interrupts in the Interrupt Control Unit (ICU)
 …
# Insert process for entering Suspend mode
 …
# Insert process to enable interrupts in the ICU
SUSPEND
```

• For VR4131 and VR4181A

```
MFC0  t5, psr
ORI   t5, t5, 1
XORI  t5, t5, 1
MTC0  t5, psr
# Insert process for entering Suspend mode
SUSPEND
```

SW Store Word SW



Format:

SW rt, 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 contents of general register *rt* are stored at the memory location specified by the effective address.

If either of the two least-significant bits of the effective address are non-zero, an address error exception occurs.

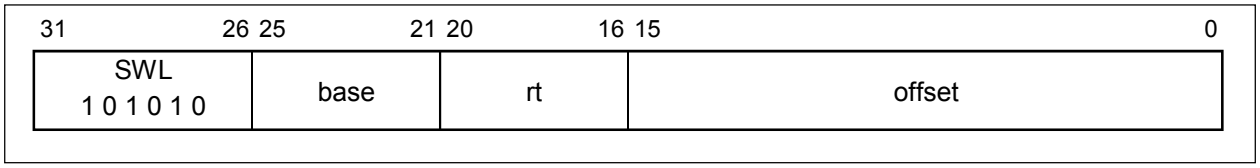
Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
            byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
            data ← GPR [rt]63 – 8*byte...0 || 0^8*byte
            StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
            byte ← vAddr2...0 xor (BigEndianCPU || 0^2)
            data ← GPR [rt]63 – 8*byte...0 || 0^8*byte
            StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)
```

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception

SWL Store Word Left SWL



Format:

SWL rt, offset (base)

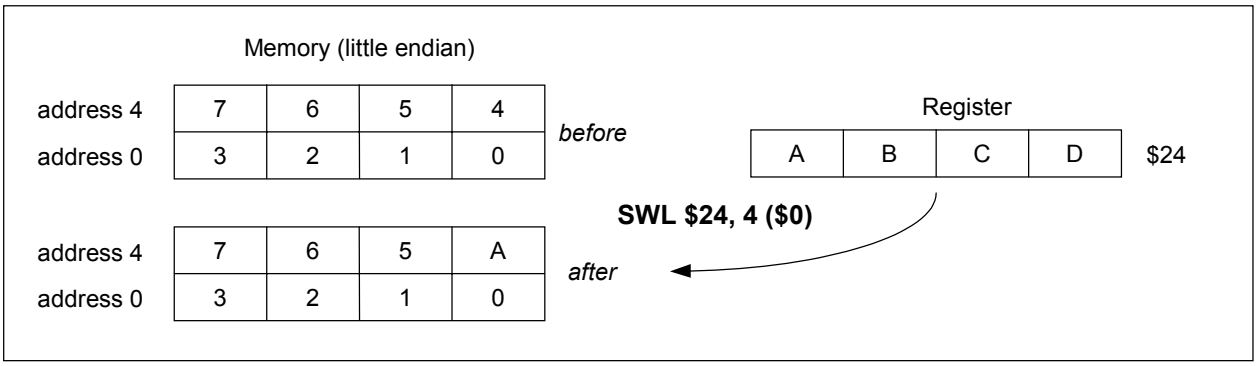
Description:

This instruction can be used with the SWR instruction to store the contents of a register into four consecutive bytes of memory, when the bytes cross a word boundary. SWL stores the left portion of the register into the appropriate part of the high-order word in memory; SWR stores the right portion of the register into the appropriate part of the low-order word.

The SWL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the word in memory that contains the specified starting byte, with the high-order part of general register *rt*. From one to four bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the most-significant (leftmost) byte of the register and copies it to the specified byte in memory; then it copies bytes from register to memory until it reaches the low-order byte of the word in memory.

No address error exceptions due to alignment are possible.



SWL Store Word Left (Continued) SWL

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || 0^24 – 8*byte || GPR [rt]31...24 – 8*byte
            else
                data ← 0^24 – 8*byte || GPR [rt]31...24 – 8*byte || 0^32
            endif
            StoreMemory (uncached, byte, data, pAddr, vAddr, DATA)

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 0 then
                pAddr ← pAddrPSIZE – 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || 0^24 – 8*byte || GPR [rt]31...24 – 8*byte
            else
                data ← 0^24 – 8*byte || GPR [rt]31...24 – 8*byte || 0^32
            endif
            StoreMemory (uncached, byte, data, pAddr, vAddr, DATA)
```

SWL Store Word Left (Continued) SWL

Given a doubleword in a register and a doubleword in memory, the operation of SWL is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	I J K L M N O E	0	0	7	E F G H M N O P	3	4	0
1	I J K L M N E F	1	0	6	I E F G M N O P	2	4	1
2	I J K L M E F G	2	0	5	I J E F M N O P	1	4	2
3	I J K L E F G H	3	0	4	I J K E M N O P	0	4	3
4	I J K E M N O P	0	4	3	I J K L E F G H	3	0	4
5	I J E F M N O P	1	4	2	I J K L M E F G	2	0	5
6	I E F G M N O P	2	4	1	I J K L M N E F	1	0	6
7	E F G H M N O P	3	4	0	I J K L M N O E	0	0	7

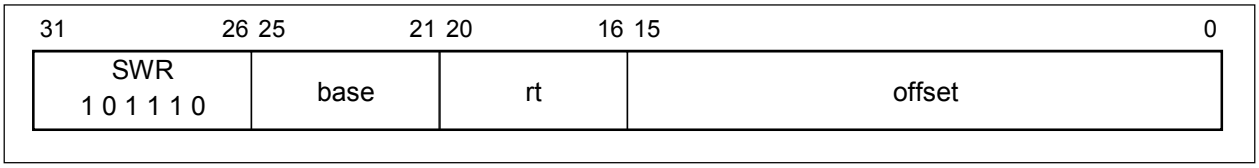
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception

SWR Store Word Right SWR



Format:

SWR rt, offset (base)

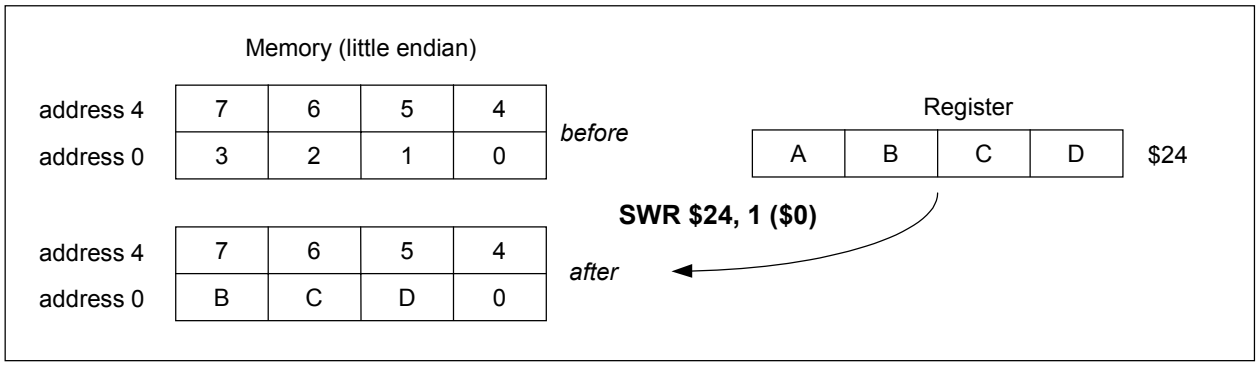
Description:

This instruction can be used with the SWL instruction to store the contents of a register into four consecutive bytes of memory, when the bytes cross a word boundary. SWR stores the right portion of the register into the appropriate part of the low-order word in memory; SWL stores the left portion of the register into the appropriate part of the high-order word.

The SWR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address that may specify an arbitrary byte. It alters only the word in memory that contains the specified starting byte, with low-order part of general register *rt*. From one to four bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the least-significant (rightmost) byte of the register and copies it to the specified byte in memory; then copies bytes from register to memory until it reaches the high-order byte of the word in memory.

No address error exceptions due to alignment are possible.



SWR Store Word Right (Continued) SWR

Operation:

```
32      T:  vAddr ← ((offset15)^16 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE – 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || GPR [rt]31 – 8*byte...0 || 0^8*byte
            else
                data ← GPR [rt]31 – 8*byte || 0^8*byte || 0^32
            endif
            StoreMemory (uncached, WORD-byte, data, pAddr, vAddr, DATA)

64      T:  vAddr ← ((offset15)^48 || offset15...0) + GPR [base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
            if BigEndianMem = 1 then
                pAddr ← pAddrPSIZE – 1...2 || 0^2
            endif
            byte ← vAddr1...0 xor BigEndianCPU^2
            if (vAddr2 xor BigEndianCPU) = 0 then
                data ← 0^32 || GPR [rt]31 – 8*byte...0 || 0^8*byte
            else
                data ← GPR [rt]31 – 8*byte || 0^8*byte || 0^32
            endif
            StoreMemory (uncached, WORD-byte, data, pAddr, vAddr, DATA)
```

SWR Store Word Right (Continued) SWR

Given a doubleword in a register and a doubleword in memory, the operation of SWR is as follows:

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1 Note			
	destination	type	offset		destination	type	offset	
			LEM	BEM Note			LEM	BEM
0	I J K L E F G H	3	0	4	H J K L M N O P	0	7	0
1	I J K L F G H P	2	1	4	G H K L M N O P	1	6	0
2	I J K L G H O P	1	2	4	F G H L M N O P	2	5	0
3	I J K L H N O P	0	3	4	E F G H M N O P	3	4	0
4	E F G H M N O P	3	4	0	I J K L H N O P	0	3	4
5	F G H L M N O P	2	5	0	I J K L G H O P	1	2	4
6	G H K L M N O P	1	6	0	I J K L F G H P	2	1	4
7	H J K L M N O P	0	7	0	I J K L E F G H	3	0	4

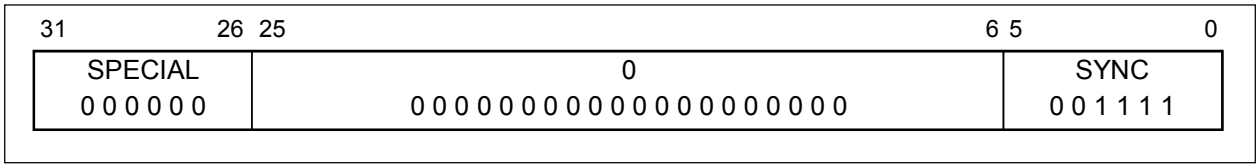
Note For VR4131 only

Remark *type*: access type (see **Figure 2-2**) sent to memory
offset: $pAddr_{2..0}$ sent to memory
LEM: Little-endian memory (BigEndianMem = 0)
BEM: Big-endian memory (BigEndianMem = 1)

Exceptions:

TLB refill exception
TLB invalid exception
TLB modified exception
Bus error exception
Address error exception

SYNC Synchronize SYNC



Format:

SYNC

Description:

The SYNC instruction is executed as a NOP on the VR4100 Series. This operation is compatible with code compiled for the VR4000.

This instruction is defined for the purpose of maintaining software compatibility with the VR4000 and VR4400.

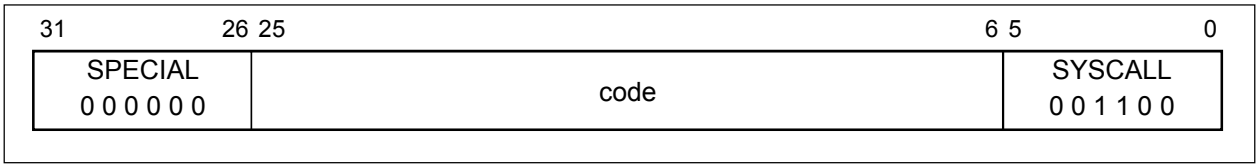
Operation:

```
32, 64 T:    SyncOperation ( )
```

Exceptions:

None

SYSCALL System Call SYSCALL



Format:

SYSCALL

Description:

A system call exception occurs by executing this instruction, immediately and unconditionally transferring control to the exception handler.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

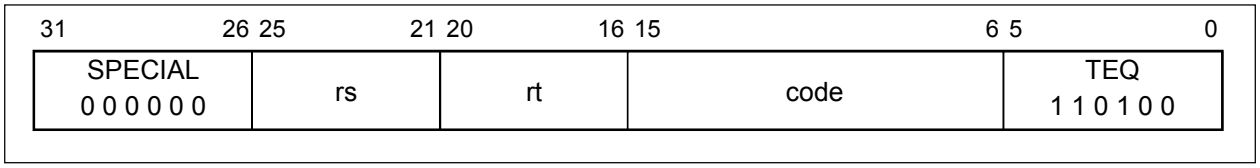
Operation:

```
32, 64   T:    SystemCallException
```

Exceptions:

System call exception

TEQ Trap if Equal TEQ



Format:

TEQ rs, rt

Description:

The contents of general register *rt* are compared to general register *rs*. If the contents of general register *rs* are equal to the contents of general register *rt*, a trap exception occurs.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

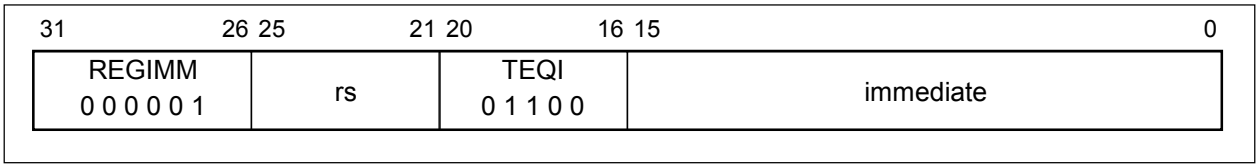
Operation:

```
32, 64   T:    if GPR [rs] = GPR [rt] then
                   TrapException
               endif
```

Exceptions:

Trap exception

TEQI Trap if Equal Immediate TEQI



Format:

TEQI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. If the contents of general register *rs* are equal to the sign-extended *immediate*, a trap exception occurs.

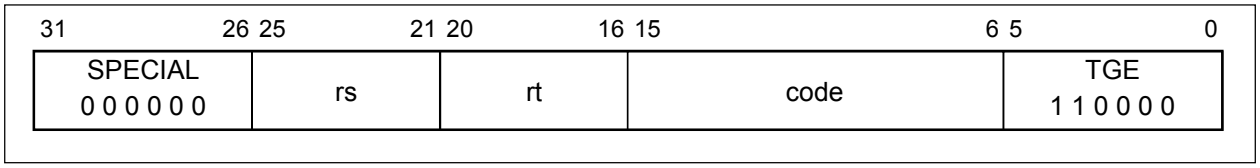
Operation:

```
32      T:   if GPR [rs] = (immediate15)^16 || immediate15...0 then
                 TrapException
             endif

64      T:   if GPR [rs] = (immediate15)^48 || immediate15...0 then
                 TrapException
             endif
```

Exceptions:

Trap exception

TGE Trap if Greater Than or Equal TGE



Format:

TGE rs, rt

Description:

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are greater than or equal to the contents of general register *rt*, a trap exception occurs.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

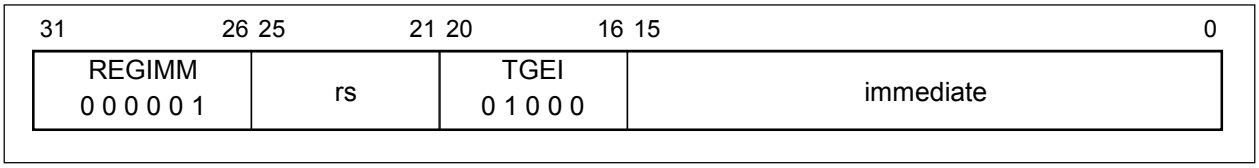
Operation:

```
32, 64   T:   if GPR [rs] ≥ GPR [rt] then
                  TrapException
              endif
```

Exceptions:

Trap exception

TGEI Trap if Greater Than or Equal Immediate TGEI



Format:

TGEI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are greater than or equal to the sign-extended *immediate*, a trap exception occurs.

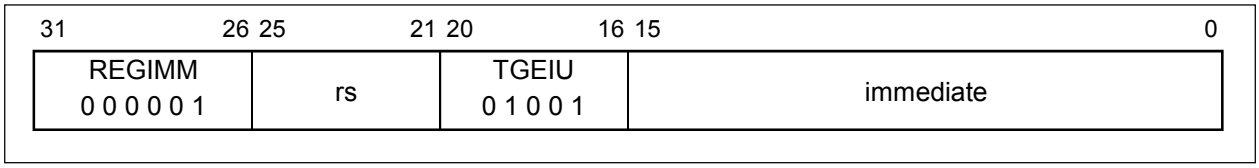
Operation:

```
32    T:   if GPR [rs] ≥ (immediate15)16 || immediate15...0 then
             TrapException
           endif

64    T:   if GPR [rs] ≥ (immediate15)48 || immediate15...0 then
             TrapException
           endif
```

Exceptions:

Trap exception

TGEIU Trap if Greater Than or Equal Immediate Unsigned TGEIU



Format:

TGEIU rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are greater than or equal to the sign-extended *immediate*, a trap exception occurs.

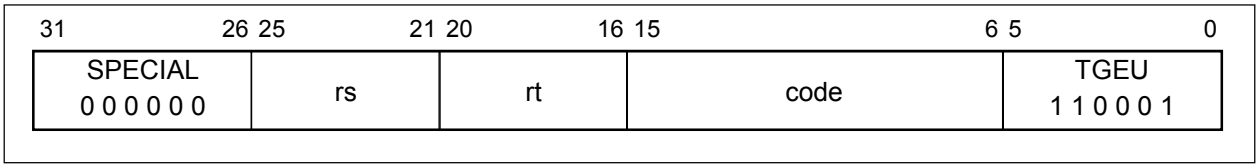
Operation:

```
32    T:   if (0 || GPR [rs]) ≥ (0 || (immediate15)^16 || immediate15...0) then
               TrapException
           endif

64    T:   if (0 || GPR [rs]) ≥ (0 || (immediate15)^48 || immediate15...0) then
               TrapException
           endif
```

Exceptions:

Trap exception

TGEU Trap if Greater Than or Equal Unsigned TGEU



Format:

TGEU rs, rt

Description:

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are greater than or equal to the contents of general register *rt*, a trap exception occurs.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

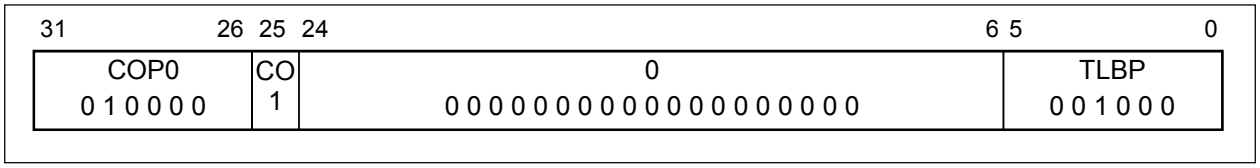
Operation:

```
32, 64   T:   if (0 || GPR [rs]) ≥ (0 || GPR [rt]) then
                  TrapException
              endif
```

Exceptions:

Trap exception

TLBP Probe TLB for Matching Entry TLBP



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.

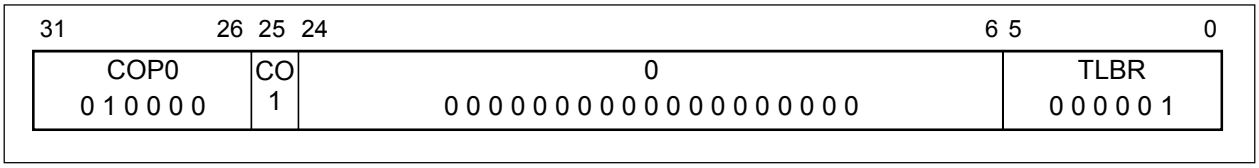
Operation:

```
32      T:  Index ← 1 || 0^25 || Undefined^6
            for i in 0…TLBEntries – 1
                if (TLB [i]95...77 = EntryHi31...13) and (TLB [i]76 or (TLB [i]71...64 = EntryHi7...0)) then
                    Index ← 0^26 || i5...0
                endif
            endfor

64      T:  Index ← 1 || 0^25 || Undefined^6
            for i in 0…TLBEntries – 1
                if (TLB [i]167...141 and not (0^15 || TLB [i]216...205)) =
                (EntryHi39...13 and not (0^15 || TLB [i]216...205)) and
                (TLB [i]140 or (TLB [i]135...126 = EntryHi7...0)) then
                    Index ← 0^26 || i5...0
                endif
            endfor
```

Exceptions:

Coprocessor unusable exception

TLBR Read Indexed TLB Entry TLBR



Format:

TLBR

Description:

The EntryHi and EntryLo registers are loaded with the contents of the TLB entry pointed at by the contents of the Index register. The *G* bit (which controls ASID matching) read from the TLB is written into both of the EntryLo0 and EntryLo1 registers.

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

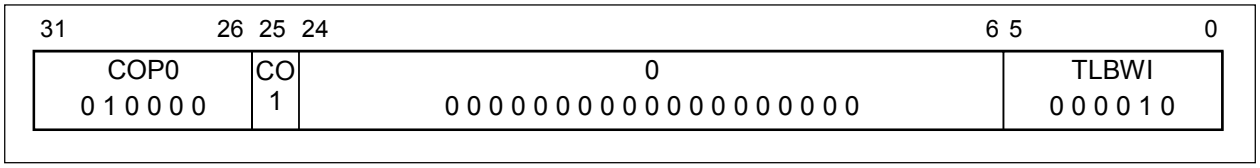
Operation:

```
32      T:  PageMask ← TLB [Index5...0]127...96
            EntryHi ← TLB [Index5...0]95...64 and not TLB [Index5...0]127...96
            EntryLo1 ← TLB [Index5...0]63...33 || TLB [Index5...0]76
            EntryLo0 ← TLB [Index5...0]31...1 || TLB [Index5...0]76

64      T:  PageMask ← TLB [Index5...0]255...192
            EntryHi ← TLB [Index5...0]191...128 and not TLB [Index5...0]255...192
            EntryLo1 ← TLB [Index5...0]127...65 || TLB [Index5...0]140
            EntryLo0 ← TLB [Index5...0]63...1 || TLB [Index5...0]140
```

Exceptions:

Coprocessor unusable exception

TLBWI Write Indexed TLB Entry TLBWI



Format:

TLBWI

Description:

The TLB entry pointed at by the contents of the Index register is loaded with the contents of the EntryHi and EntryLo registers. The *G* bit of the TLB 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 the Index register are greater than the number of TLB entries in the processor.

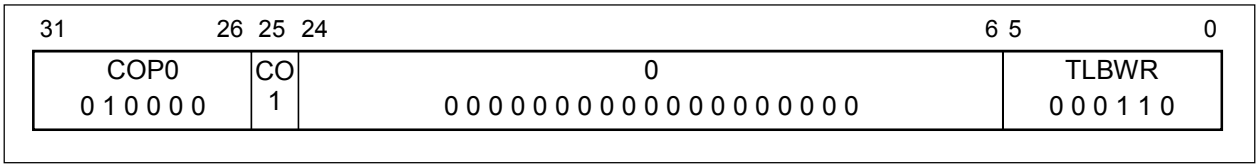
Operation:

```
32, 64   T:     TLB [Index5...0] ← PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0
```

Exceptions:

Coprocessor unusable exception

TLBWR Write Random TLB Entry TLBWR



Format:

TLBWR

Description:

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

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

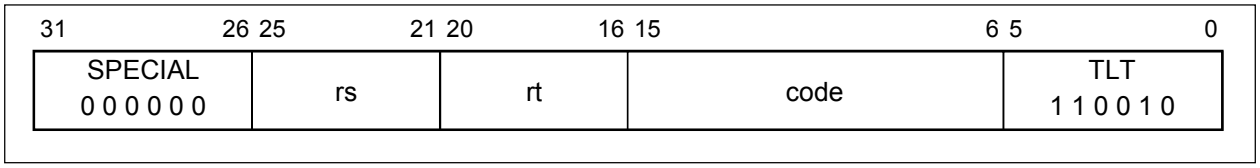
Operation:

```
32, 64   T:     TLB [Random5...0] ← PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0
```

Exceptions:

Coprocessor unusable exception

TLT Trap if Less Than TLT



Format:

TLT rs, rt

Description:

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the contents of general register *rt*, a trap exception occurs.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

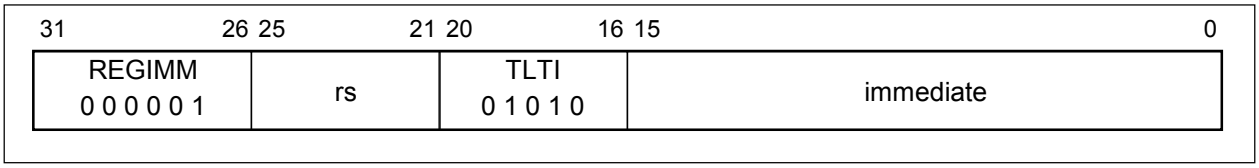
Operation:

```
32, 64   T:    if GPR [rs] < GPR [rt] then
                   TrapException
               endif
```

Exceptions:

Trap exception

TLTI Trap if Less Than Immediate TLTI



Format:

TLTI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the sign-extended *immediate*, a trap exception occurs.

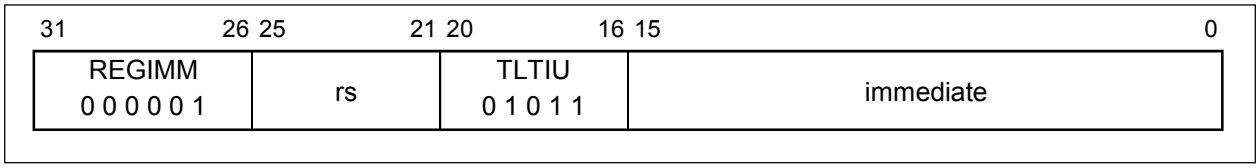
Operation:

```
32    T:   if GPR [rs] < (immediate15)^16 || immediate15...0 then
               TrapException
           endif

64    T:   if GPR [rs] < (immediate15)^48 || immediate15...0 then
               TrapException
           endif
```

Exceptions:

Trap exception

TLTIU Trap if Less Than Immediate Unsigned TLTIU



Format:

TLTIU rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the sign-extended *immediate*, a trap exception occurs.

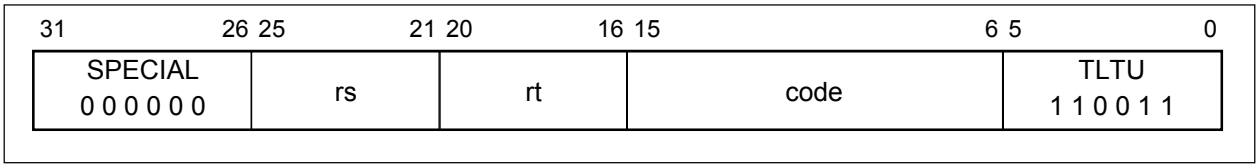
Operation:

```
32    T:   if (0 || GPR [rs]) < (0 || (immediate15)^16 || immediate15...0) then
               TrapException
           endif

64    T:   if (0 || GPR [rs]) < (0 || (immediate15)^48 || immediate15...0) then
               TrapException
           endif
```

Exceptions:

Trap exception

TLTU Trap if Less Than Unsigned TLTU



Format:

TLTU rs, rt

Description:

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the contents of general register *rt*, a trap exception occurs.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

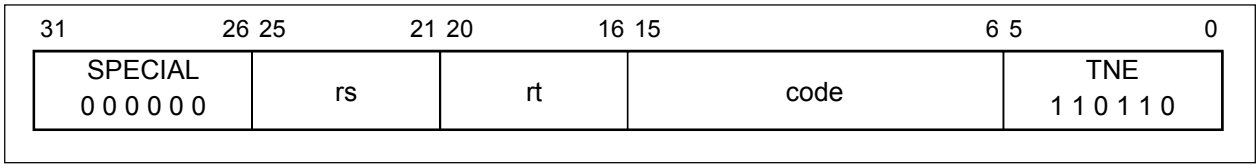
Operation:

```
32, 64   T:    if (0 || GPR [rs]) < (0 || GPR [rt]) then
                   TrapException
               endif
```

Exceptions:

Trap exception

TNE Trap if Not Equal TNE



Format:

TNE rs, rt

Description:

The contents of general register *rt* are compared to the contents of general register *rs*. If the contents of general register *rs* are not equal to the contents of general register *rt*, a trap exception occurs.

The *code* field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

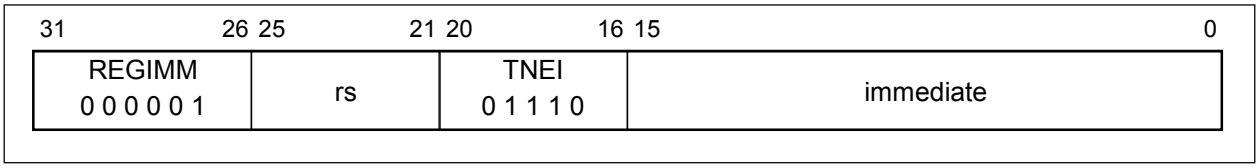
Operation:

```
32, 64   T:    if GPR [rs] ≠ GPR [rt] then
                   TrapException
               endif
```

Exceptions:

Trap exception

TNEI Trap if Not Equal Immediate TNEI



Format:

TNEI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. If the contents of general register *rs* are not equal to the sign-extended *immediate*, a trap exception occurs.

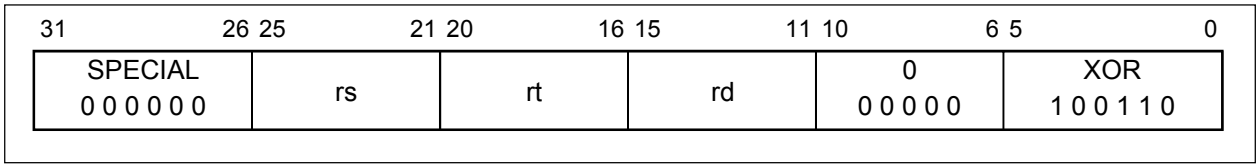
Operation:

```
32    T:   if GPR [rs] ≠ (immediate15)16 || immediate15...0 then
             TrapException
           endif

64    T:   if GPR [rs] ≠ (immediate15)48 || immediate15...0 then
             TrapException
           endif
```

Exceptions:

Trap exception

XOR Exclusive OR XOR



Format:

XOR rd, rs, rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical exclusive OR operation. The result is placed into general register *rd*.

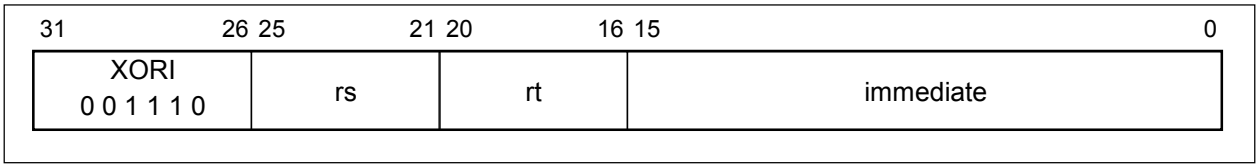
Operation:

```
32, 64   T:    GPR [rd] ← GPR [rs] xor GPR [rt]
```

Exceptions:

None

XORI Exclusive OR Immediate XORI



Format:

XORI rt, rs, immediate

Description:

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical exclusive OR operation. The result is placed into general register *rt.*

Operation:

32 T: GPR [rt] ← GPR [rs] xor (0^{16} || immediate)

64 T: GPR [rt] ← GPR [rs] xor (0^{48} || immediate)

Exceptions:

None

9.4 CPU Instruction Opcode Bit Encoding

The remainder of this chapter presents the opcode bit encoding for the CPU instruction set (ISA and extensions), as implemented by the VR4100 Series. Figure 9-1 lists the VR4100 Series Opcode Bit Encoding.

Figure 9-1. CPU Instruction Opcode Bit Encoding (1/3)

Opcode

31...29 \ 28...26	0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	π	π	*	BEQL	BNEL	BLEZL	BGTZL
3	DADDIε	DADDIUε	LDLε	LDRε	*	JALXθ	*	*
4	LB	LH	LWL	LW	LBU	LHU	LWR	LWUε
5	SB	SH	SWL	SW	SDLε	SDRε	SWR	CACHEδ
6	*	π	π	*	*	π	π	LDε
7	*	π	π	*	*	π	π	SDε

SPECIAL function

5...3 \ 2...0	0	1	2	3	4	5	6	7
0	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1	JR	JALR	*	*	SYSCALL	BREAK	*	SYNC
2	MFHI	MTHI	MFLO	MTLO	DSLLVε	*	DSRLVε	DSRAVε
3	MULT	MULTU	DIV	DIVU	DMULTε	DMULTUε	DDIVε	DDIVUε
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	**Note 1**	**Note 2**	SLT	SLTU	DADDε	DADDUε	DSUBε	DSUBUε
6	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7	DSLLε	*	DSRLε	DSRAε	DSLL32ε	*	DSRL32ε	DSRA32ε

REGIMM rt

20...19 \ 18...16	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1	TGEI	TGEIU	TLTI	TLTIU	TEQI	*	TNEI	*
2	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3	*	*	*	*	*	*	*	*

Notes 1. VR4121, VR4122, VR4131, VR4181A … MACC
VR4181 … MADD16

2. VR4121, VR4122, VR4131, VR4181A … DMACC
VR4181 … DMADD16

Figure 9-1. CPU Instruction Opcode Bit Encoding (2/3)

23...21 **COP0 rs**

25…24	0	1	2	3	4	5	6	7
0	MF	DMFε	γ	γ	MT	DMTε	γ	γ
1	BC	γ	γ	γ	γ	γ	γ	γ
2	CO							
3								

18...16 **COP0 rt**

20...19	0	1	2	3	4	5	6	7
0	BCF	BCT	BCFL	BCTL	γ	γ	γ	γ
1	γ	γ	γ	γ	γ	γ	γ	γ
2	γ	γ	γ	γ	γ	γ	γ	γ
3	γ	γ	γ	γ	γ	γ	γ	γ

2...0 **CP0 Function**

5...3	0	1	2	3	4	5	6	7
0	ϕ	TLBR	TLBWI	ϕ	ϕ	ϕ	TLBWR	ϕ
1	TLBP	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
2	ξ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
3	ERET χ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
4	ϕ	STANDBY	SUSPEND	HIBERNATE	ϕ	ϕ	ϕ	ϕ
5	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
6	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
7	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

Figure 9-1. CPU Instruction Opcode Bit Encoding (3/3)

Key:

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

γ Operation codes marked with a gamma cause a reserved instruction exception. They are reserved for future versions of the architecture.

δ Operation codes marked with a delta are valid only for processors conforming to MIPS III instruction set or later with CP0 enabled, and cause a reserved instruction exception on other processors.

φ Operation codes marked with a phi are invalid but do not cause reserved instruction exceptions in VR4100 Series implementations.

ξ Operation codes marked with a xi cause a reserved instruction exception on VR4100 Series processors.

χ Operation codes marked with a chi are valid on processors conforming to MIPS III instruction set or later only.

ε Operation codes marked with an epsilon are valid when the processor operating in 64-bit mode or in 32-bit Kernel mode. These instructions will cause a reserved instruction exception if the processor operates in 32-bit User or Supervisor mode.

π Operation codes marked with a pi are invalid and cause coprocessor unusable exception on VR4100 Series processors.

θ Operation codes marked with a theta are valid when MIPS16 instruction execution is enabled, and cause a reserved instruction exception when MIPS16 instruction execution is disabled.

CHAPTER 10 MIPS16 INSTRUCTION SET FORMAT

This chapter describes the format of each MIPS16 instruction, and the format of the MIPS instructions that are made by converting MIPS16 instructions in alphabetical order. For details of MIPS16 instruction conversion and opcode, refer to **CHAPTER 3 MIPS16 INSTRUCTION SET**.

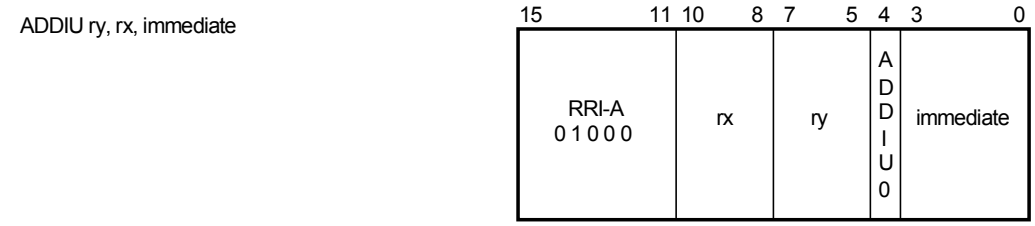
Caution For some instructions, their format or syntax may become ineffective after they are converted to a 32-bit instruction. For details of formats and syntax of 32-bit instructions, refer to CHAPTER 2 CPU INSTRUCTION SET SUMMARY and CHAPTER 9 CPU INSTRUCTION SET DETAILS.

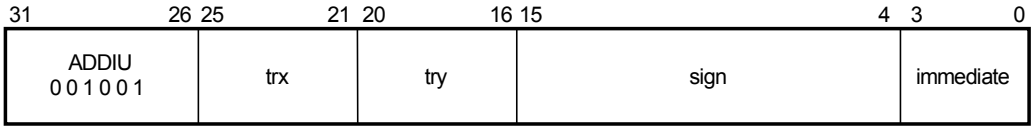
ADDIU

Add Immediate Unsigned

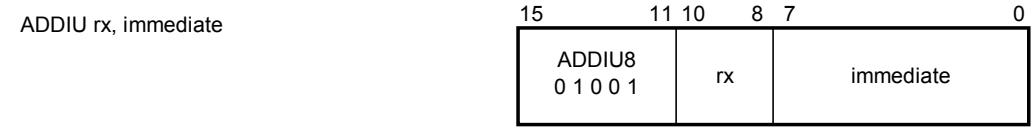
(1/2)

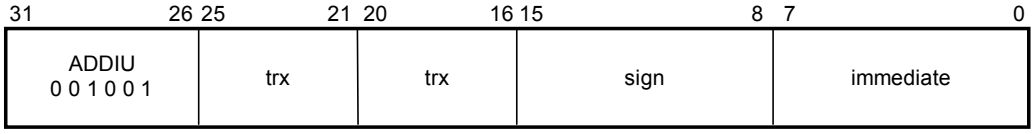
ADDIU ry, rx, immediate



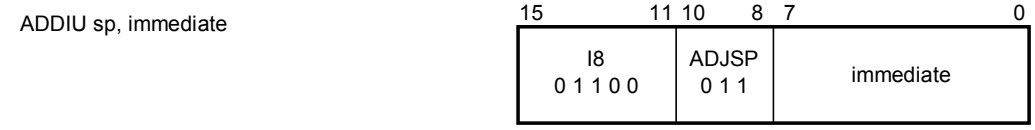


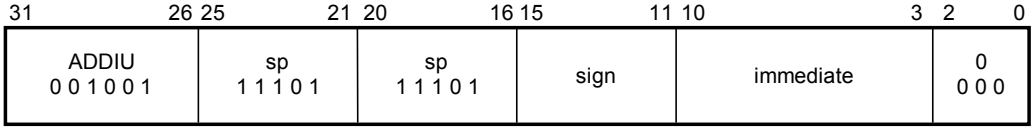
ADDIU rx, immediate





ADDIU sp, immediate

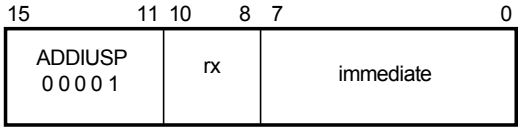


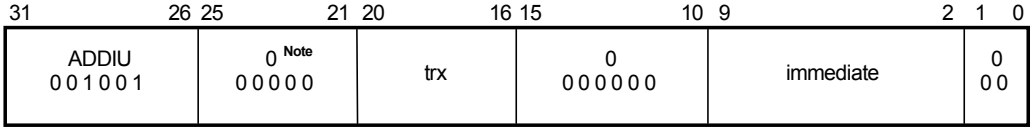


ADDIU

Add Immediate Unsigned

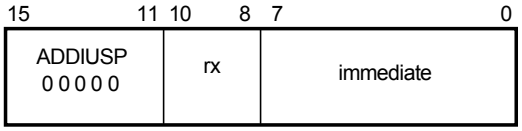
ADDIU rx, pc, immediate

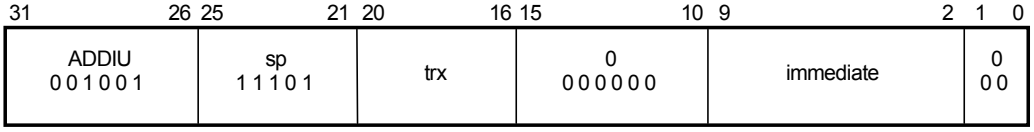




Note Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 3 for a complete definition of the semantics of the MIPS16 PC-relative instructions.

ADDIU rx, sp, immediate

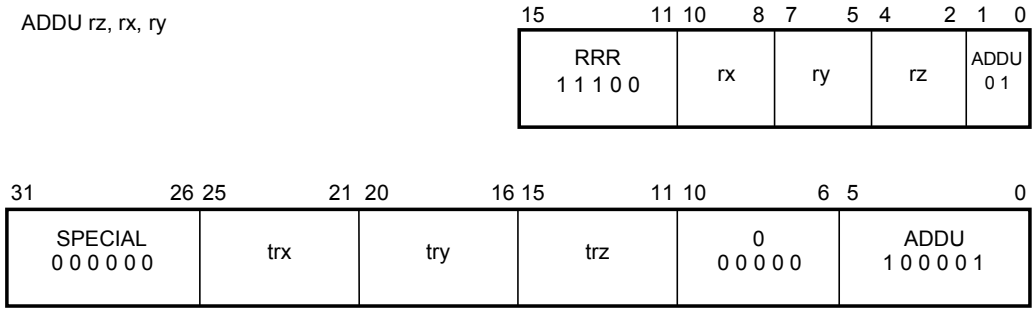




ADDU

Add Unsigned

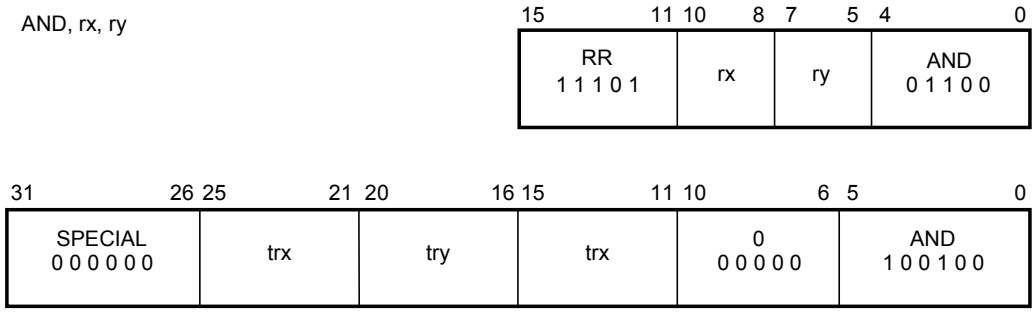
ADDU rz, rx, ry



AND

AND

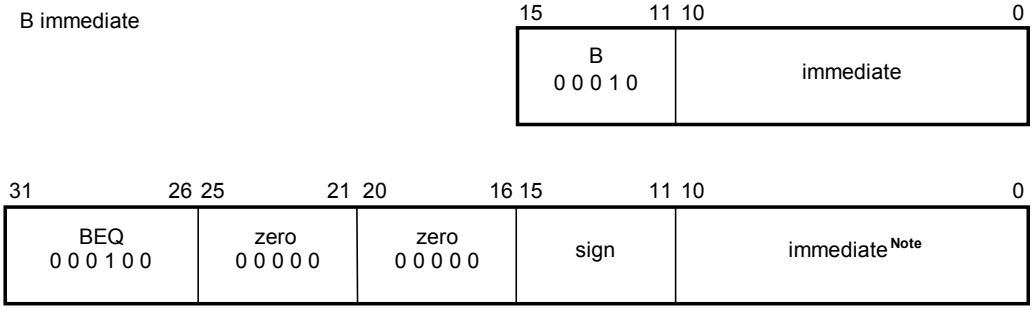
AND, rx, ry



B

Branch Unconditional

B immediate

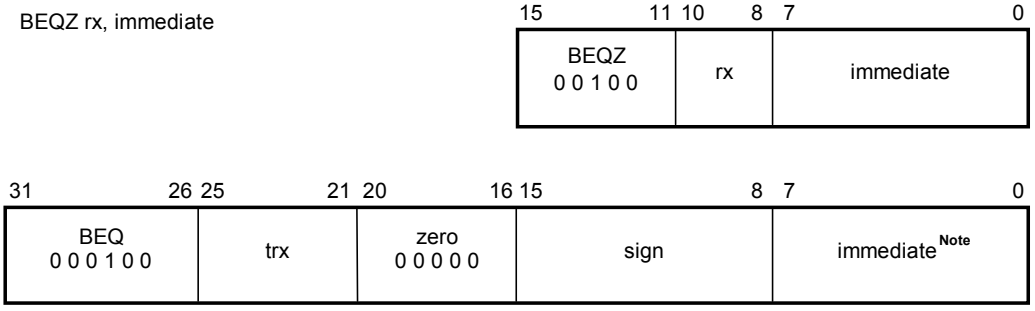


Note In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 2 and Chapter 9 for a complete definition of the semantics of the branch instructions.

BEQZ

Branch on Equal to Zero

BEQZ rx, immediate

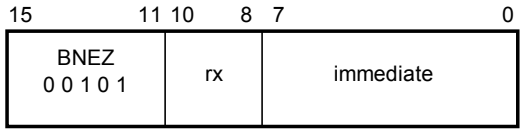


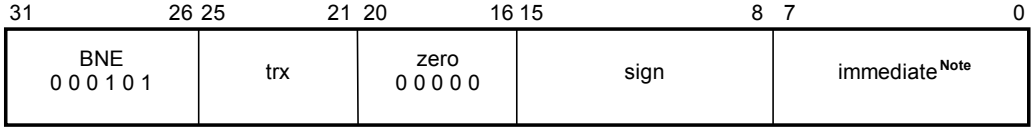
Note In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 2 and Chapter 9 for a complete definition of the semantics of the branch instructions.

BNEZ

Branch on Not Equal to Zero

BNEZ rx, immediate



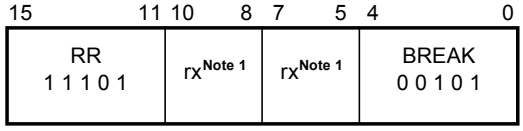


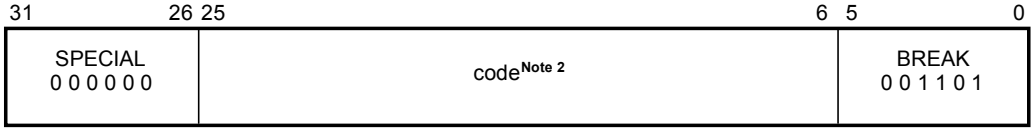
Note In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 2 and Chapter 9 for a complete definition of the semantics of the branch instructions.

BREAK

Breakpoint

BREAK immediate





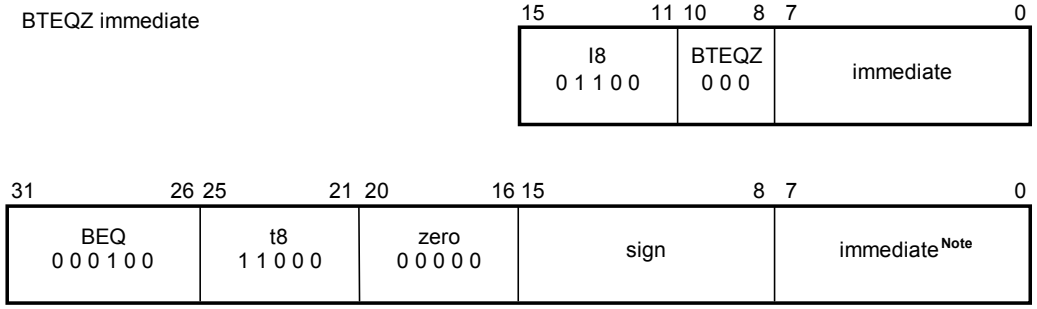
Notes 1. The two register fields in the MIPS16 break instruction may be used as a 6-bit code (immediate) field for software parameters. The 6-bit code can be retrieved by the exception handler.

2. The 32-bit break instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. The code field is entirely ignored by the pipeline, and it is not visible in any way to the software executing on the processor.

BTEQZ

Branch on T Equal to Zero

BTEQZ immediate

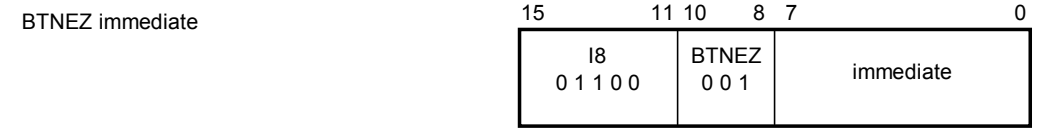


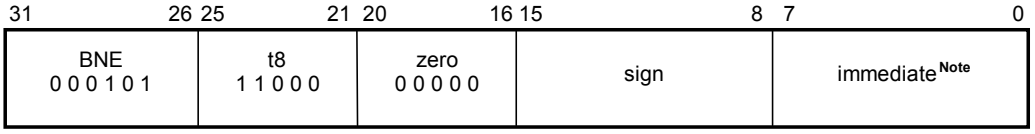
Note In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 2 and Chapter 9 for a complete definition of the semantics of the branch instructions.

BTNEZ

Branch on T Not Equal to Zero

BTNEZ immediate



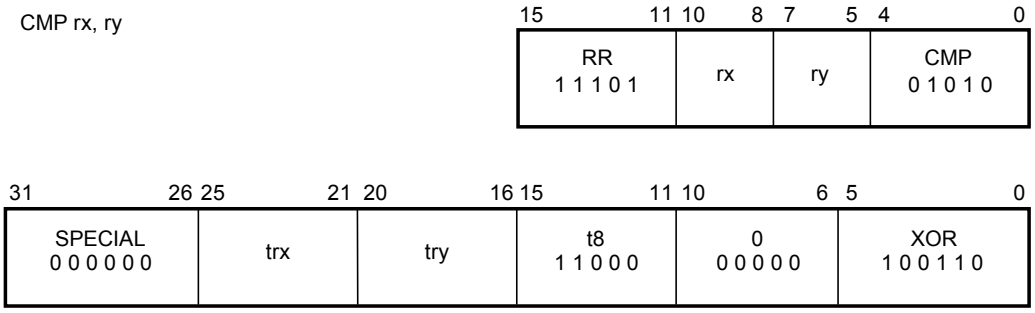


Note In MIPS16 mode, the branch offset is interpreted as halfword aligned. This is unlike 32-bit MIPS mode which interprets the offset value as word aligned. The 32-bit branch instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 2 and Chapter 9 for a complete definition of the semantics of the branch instructions.

CMP

Compare

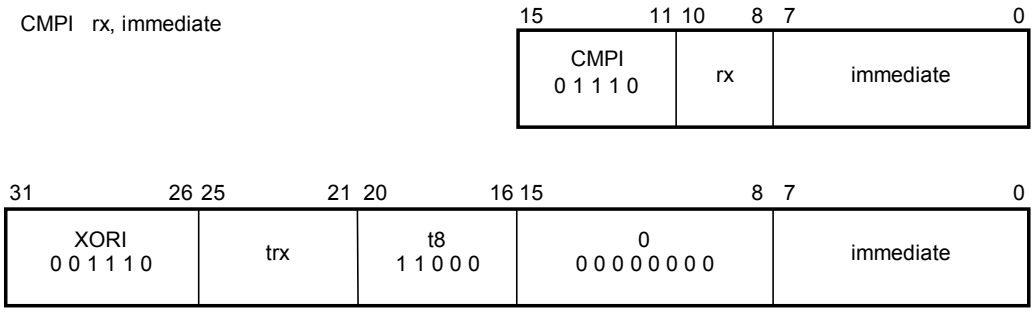
CMP rx, ry



CMPI

Compare Immediate

CMPI rx, immediate

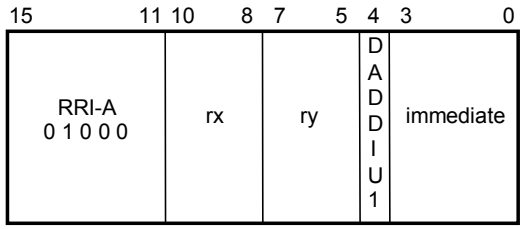


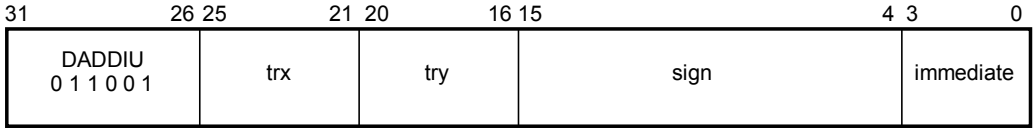
DADDIU

Doubleword Add Immediate Unsigned

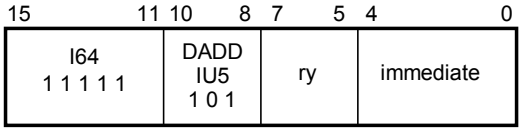
(1/2)

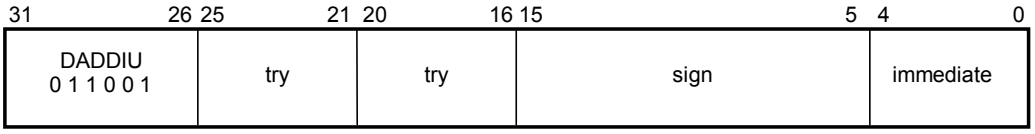
DADDIU ry, rx, immediate



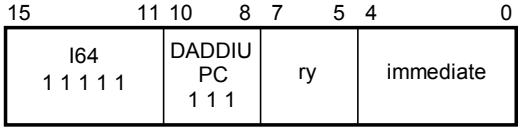


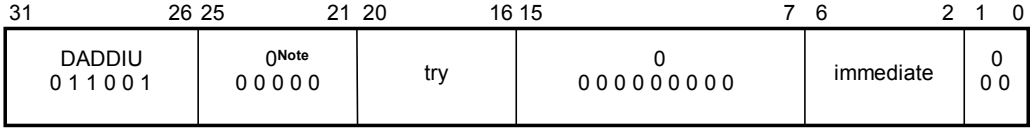
DADDIU ry, immediate





DADDIU ry, pc, immediate





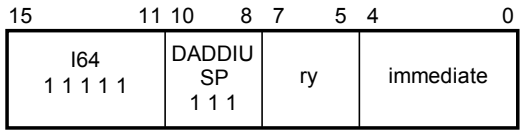
Note Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 3 for a complete definition of the semantics of the MIPS16 PC-relative instructions.

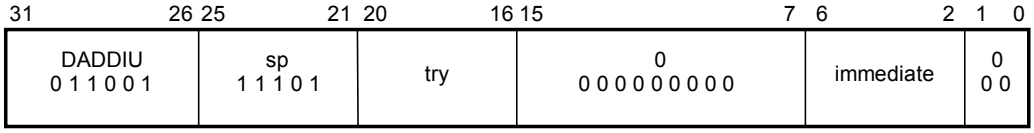
DADDIU

Doubleword Add Immediate Unsigned

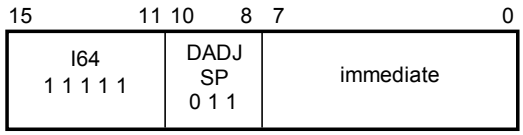
(2/2)

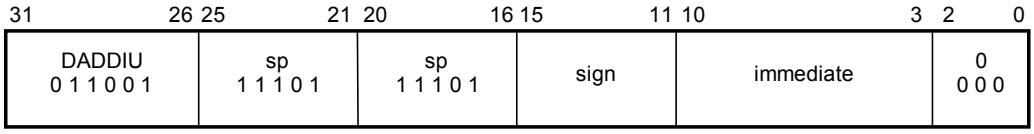
DADDIU ry, sp, immediate





DADDIU sp, immediate

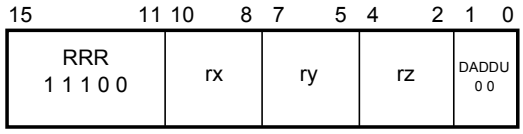


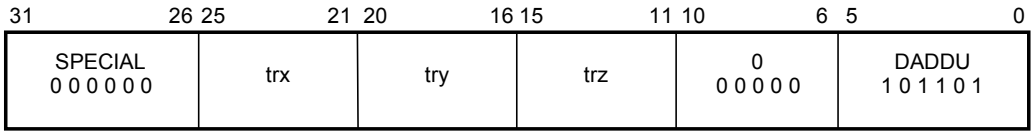


DADDU

Doubleword Add Unsigned

DADDU rz, rx, ry

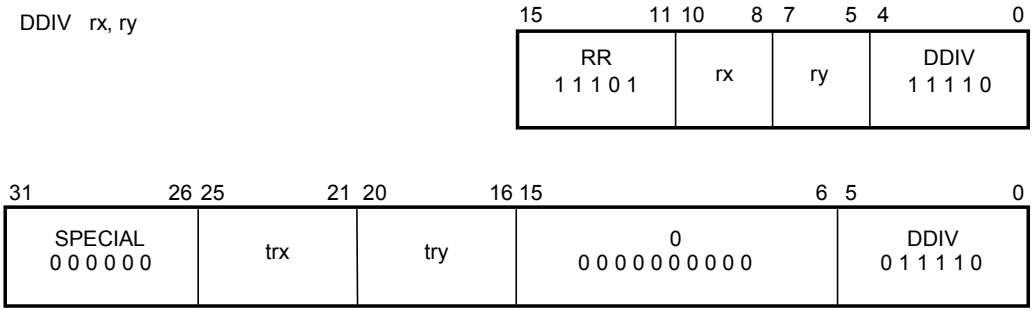




DDIV

Doubleword Divide

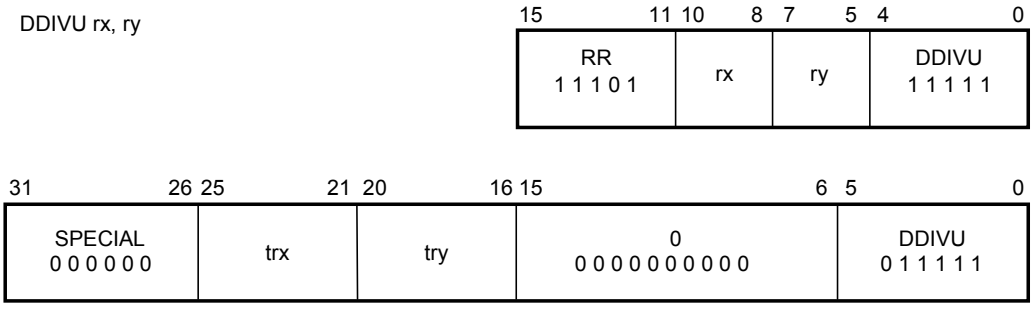
DDIV rx, ry



DDIVU

Doubleword Divide Unsigned

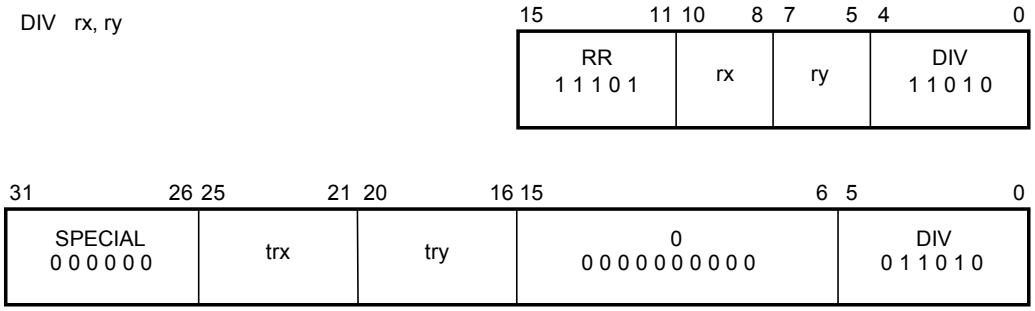
DDIVU rx, ry



DIV

Divide

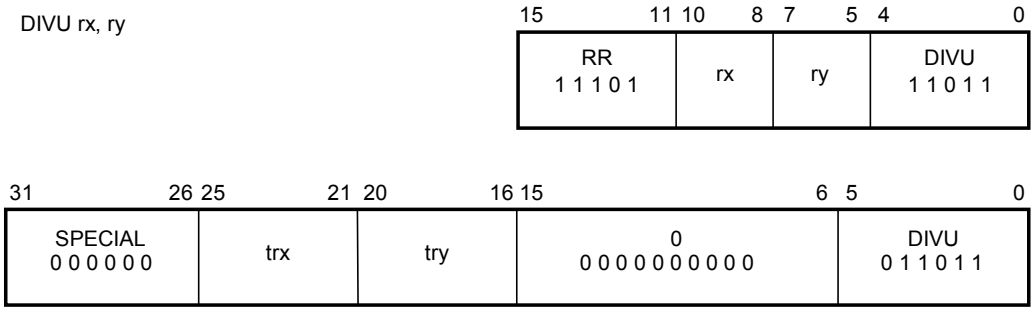
DIV rx, ry



DIVU

Divide Unsigned

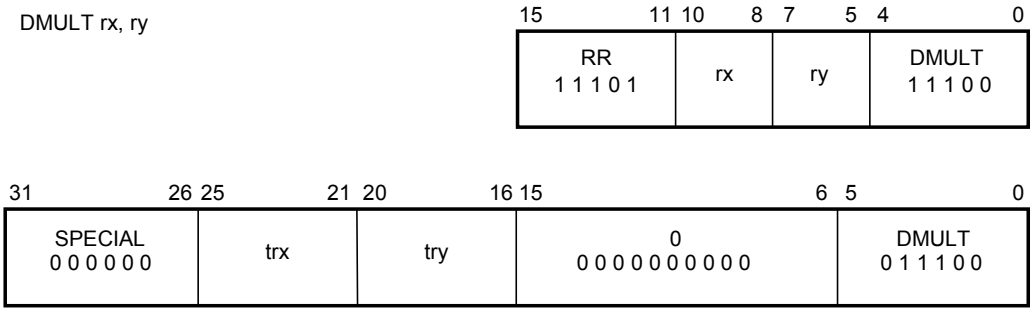
DIVU rx, ry



DMULT

Doubleword Multiply

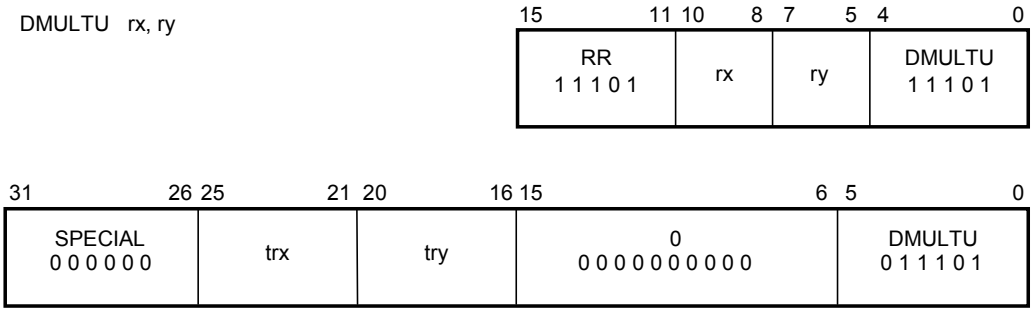
DMULT rx, ry



DMULTU

Doubleword Multiply Unsigned

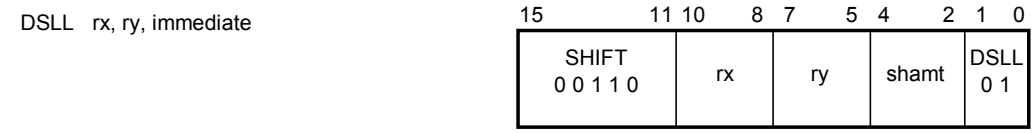
DMULTU rx, ry

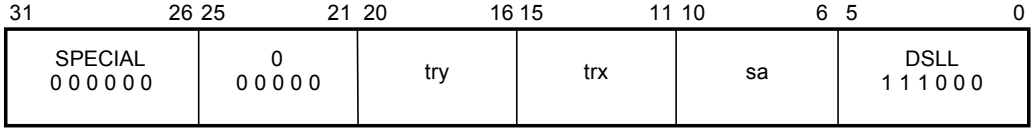


DSLL

Doubleword Shift Left Logical

DSLL rx, ry, immediate

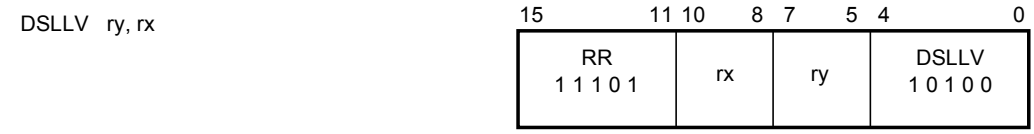


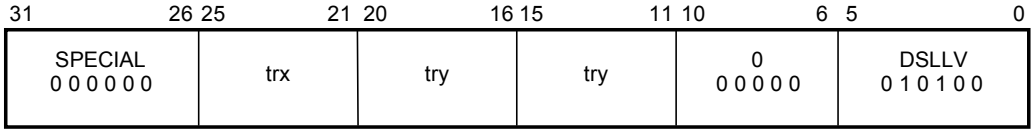


DSLLV

Doubleword Shift Left Logical Variable

DSLLV ry, rx

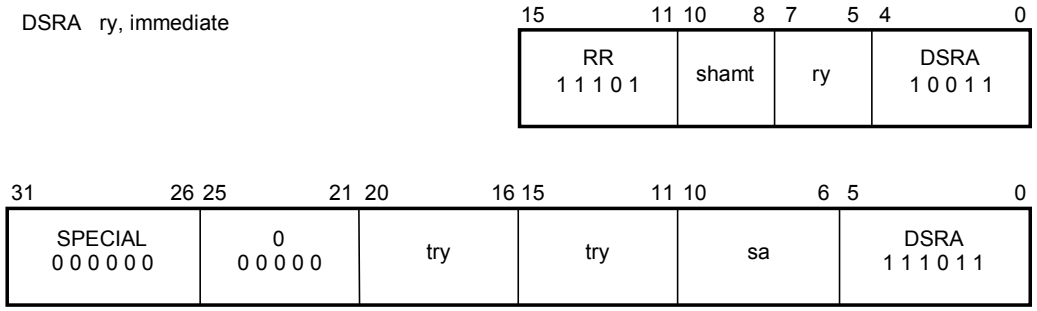




DSRA

Doubleword Shift Right Arithmetic

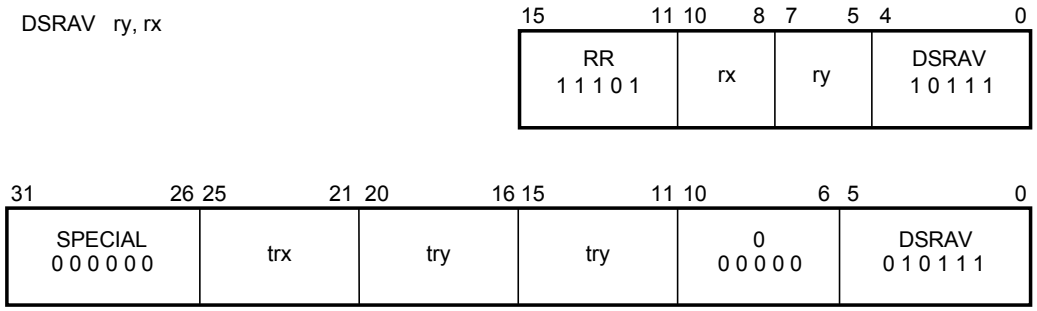
DSRA ry, immediate



DSRAV

Doubleword Shift Right Arithmetic Variable

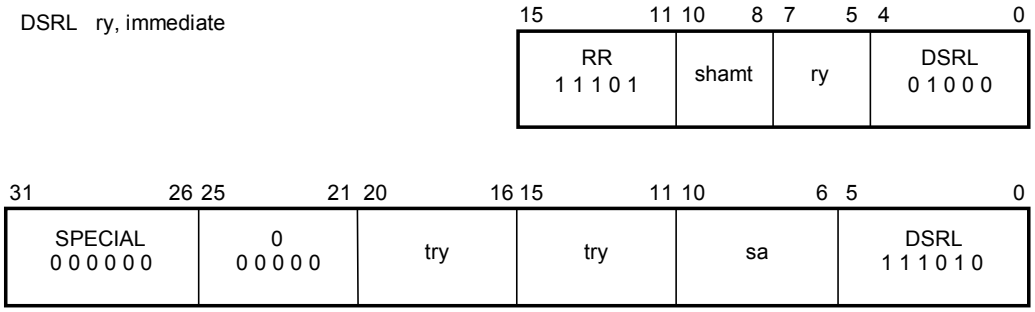
DSRAV ry, rx



DSRL

Doubleword Shift Right Logical

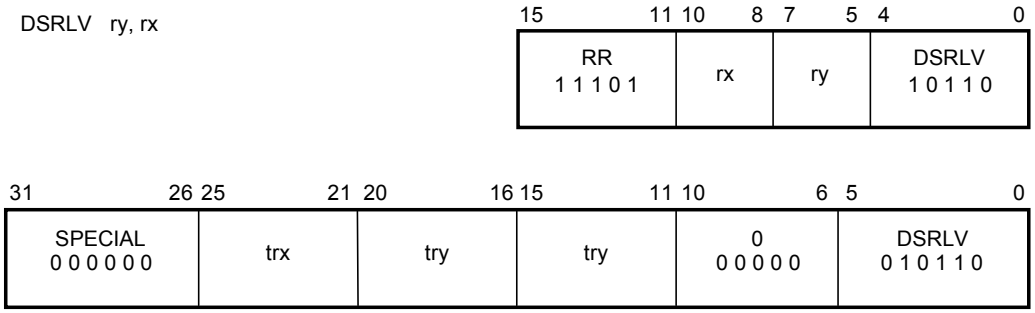
DSRL ry, immediate



DSRLV

Doubleword Shift Right Logical Variable

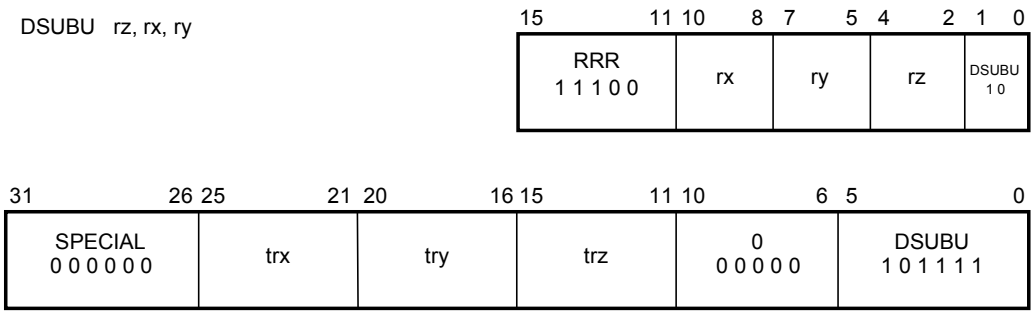
DSRLV ry, rx



DSUBU

Doubleword Subtract Unsigned

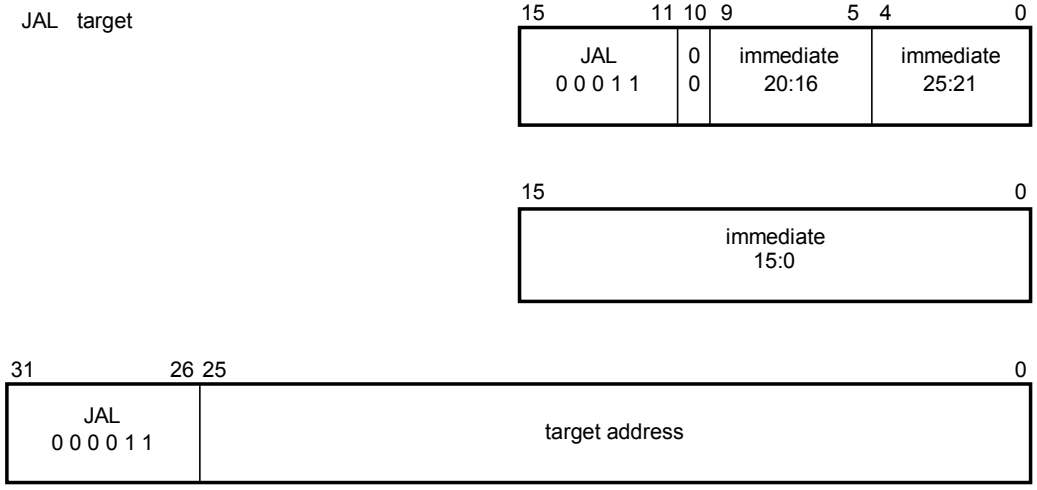
DSUBU rz, rx, ry



JAL

Jump and Link

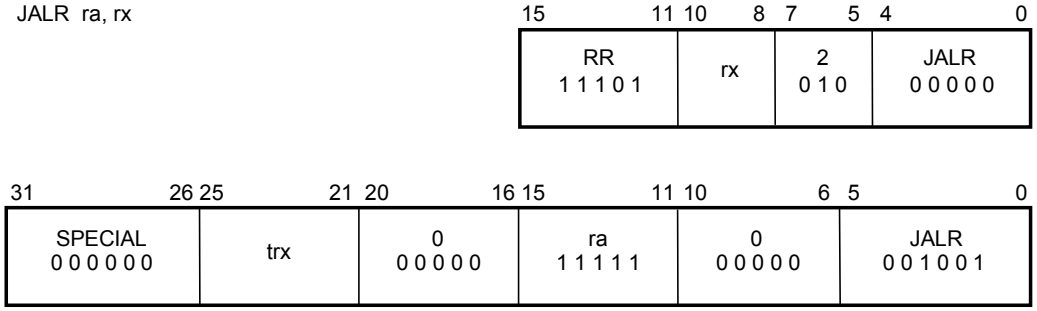
JAL target



JALR

Jump and Link Register

JALR ra, rx



JALX

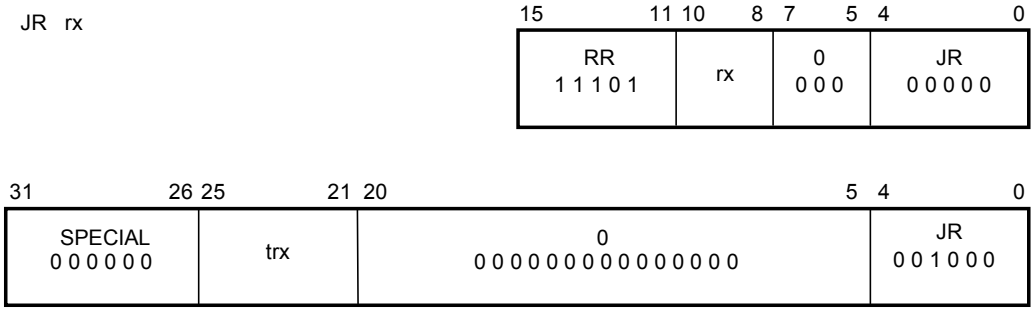
Jump and Link Exchange

JALX target

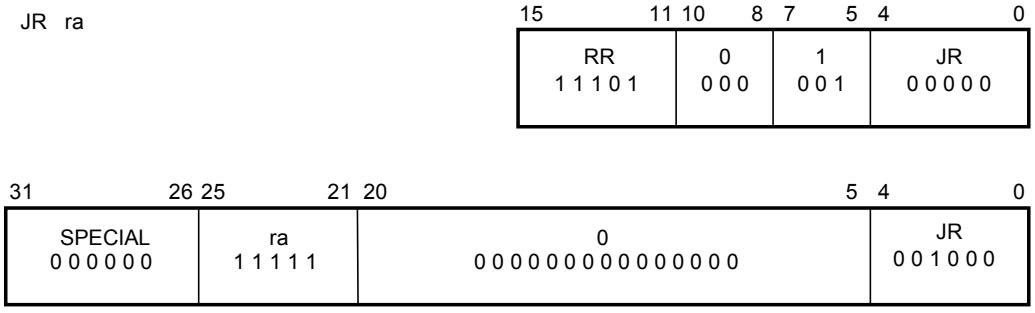
JR

Jump Register

JR rx



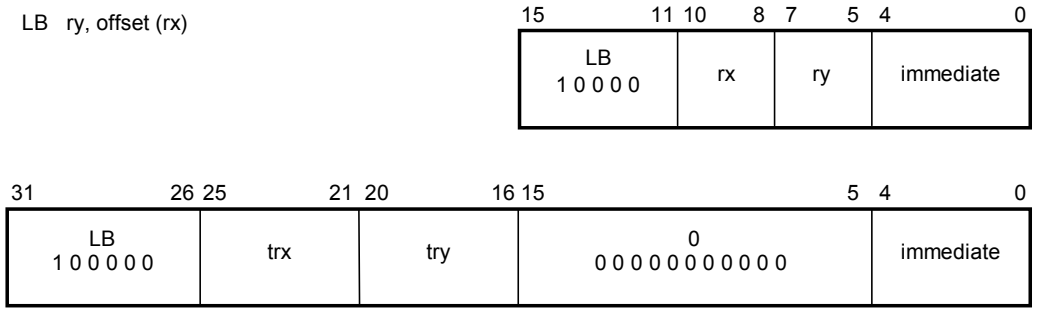
JR ra



LB

Load Byte

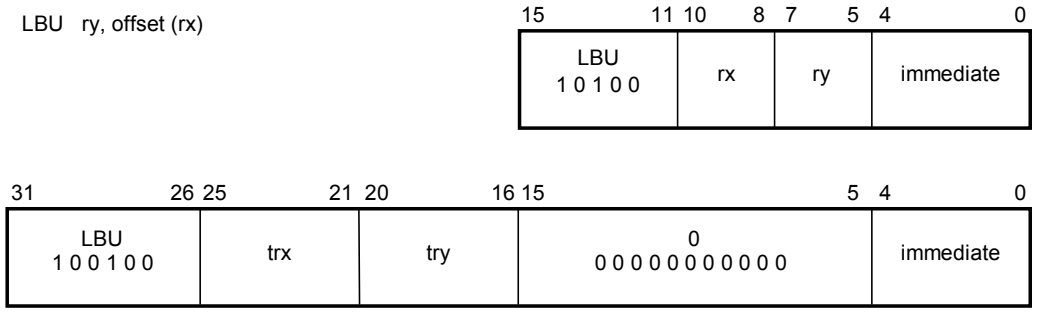
LB ry, offset (rx)



LBU

Load Byte Unsigned

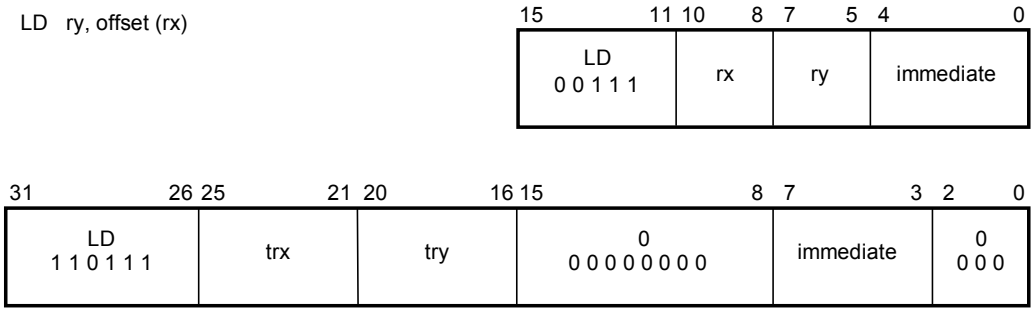
LBU ry, offset (rx)



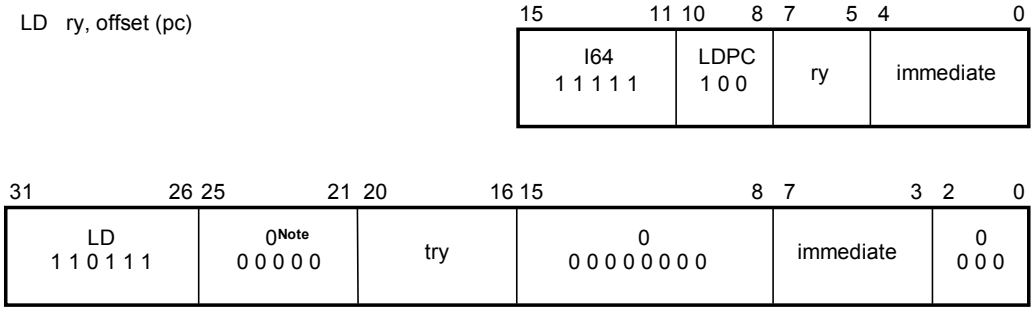
LD

Load Doubleword

LD ry, offset (rx)

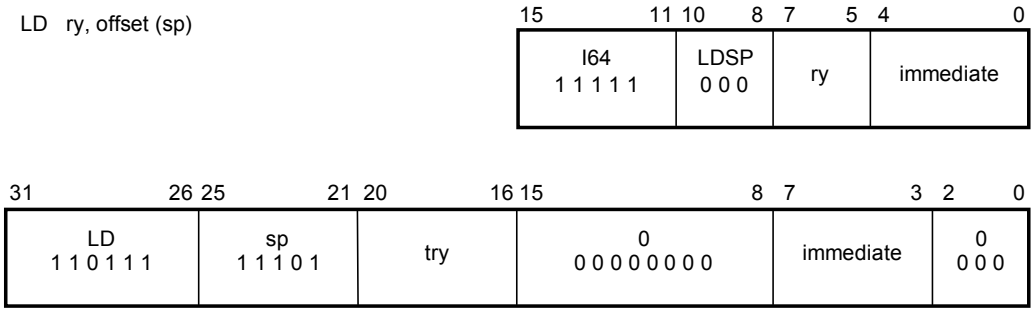


LD ry, offset (pc)



Note Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 3 for a complete definition of the semantics of the MIPS16 PC-relative instructions.

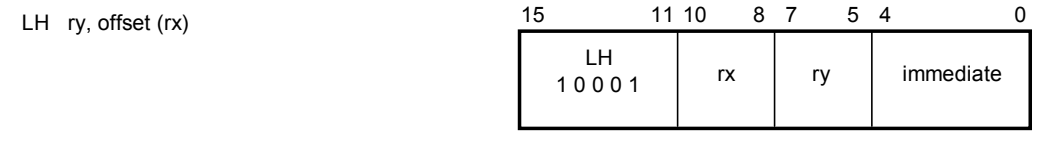
LD ry, offset (sp)

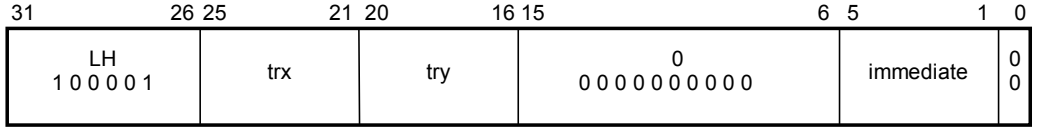


LH

Load Halfword

LH ry, offset (rx)

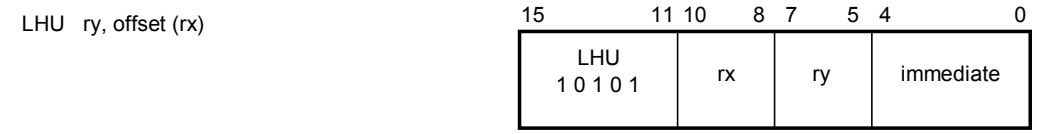


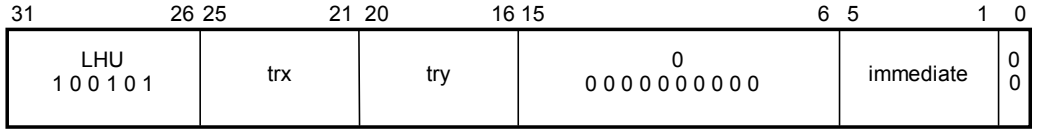


LHU

Load Halfword Unsigned

LHU ry, offset (rx)

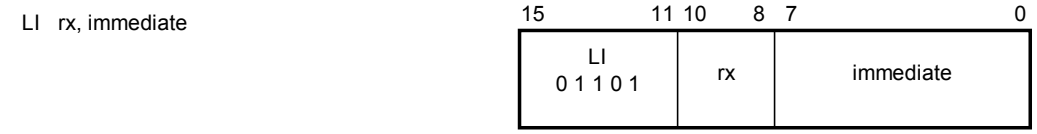


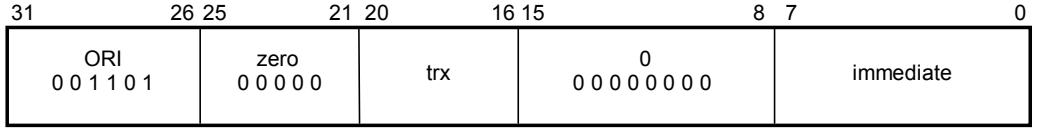


LI

Load Immediate

LI rx, immediate

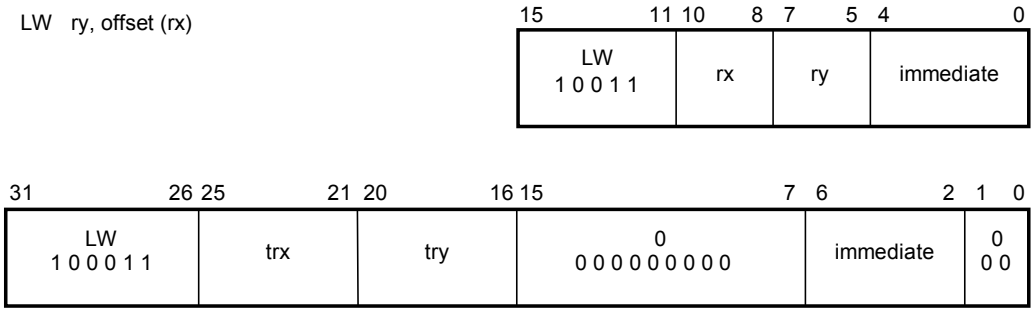




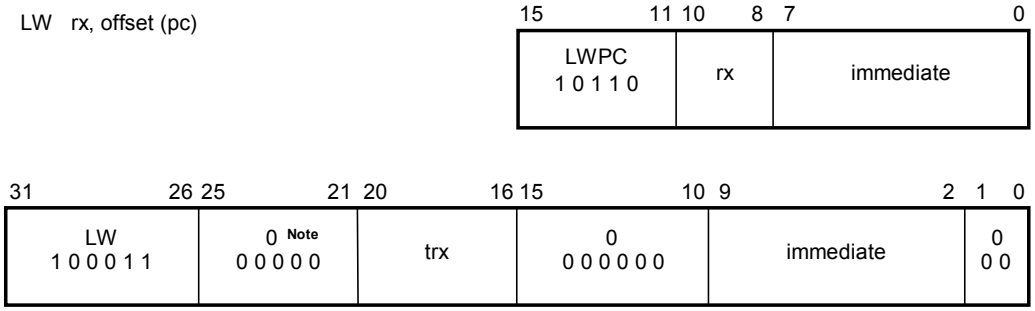
LW

Load Word

LW ry, offset (rx)

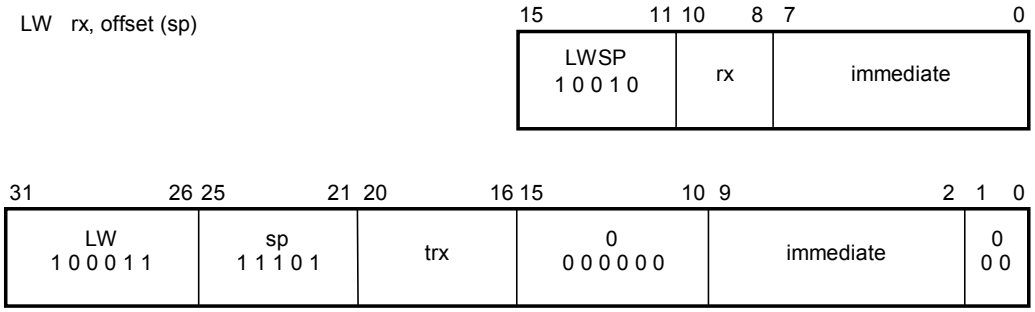


LW rx, offset (pc)



Note Zeros are shown in the field of bits 21 to 25 as placeholders. The 32-bit PC-relative instruction format shown above is provided here only to make the description complete; it is not a valid 32-bit MIPS instruction. See Chapter 3 for a complete definition of the semantics of the MIPS16 PC-relative instructions.

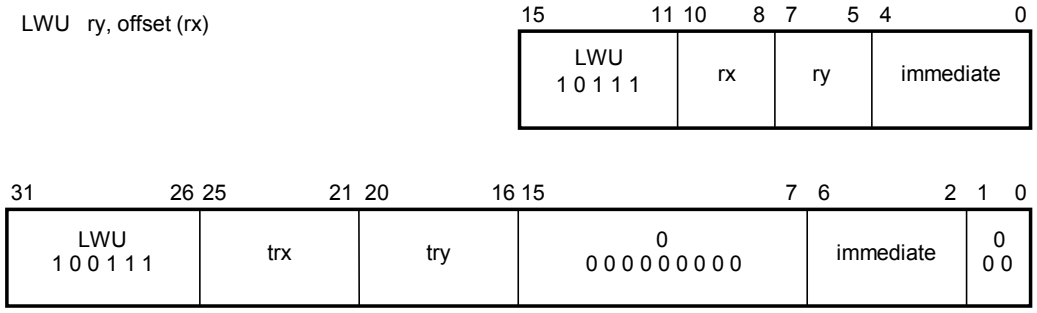
LW rx, offset (sp)



LWU

Load Word Unsigned

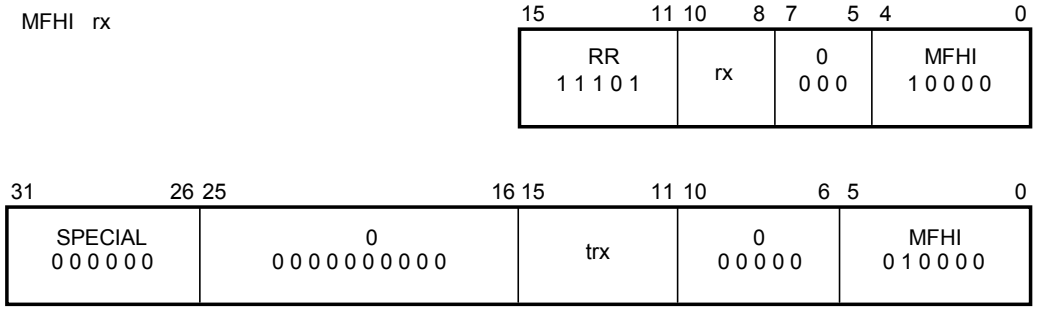
LWU ry, offset (rx)



MFHI

Move from HI Register

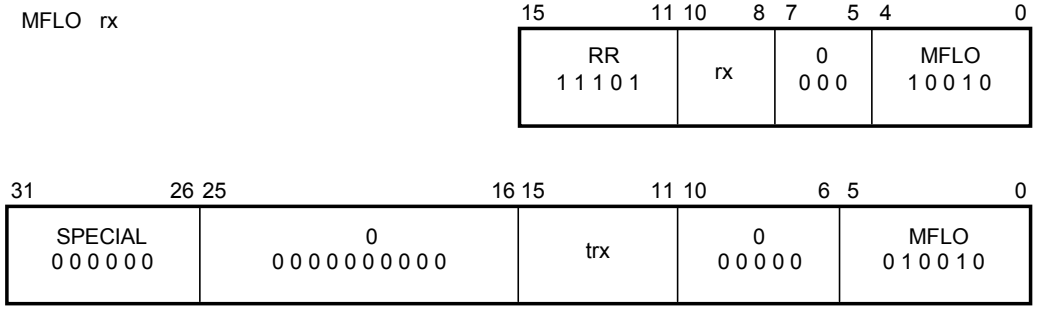
MFHI rx



MFLO

Move from LO Register

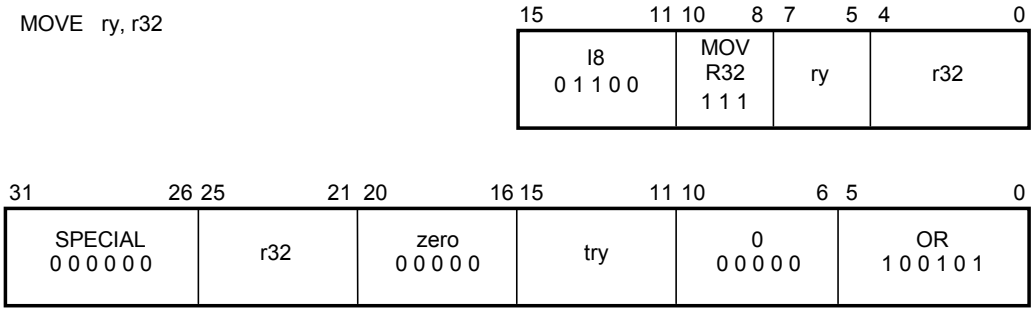
MFLO rx



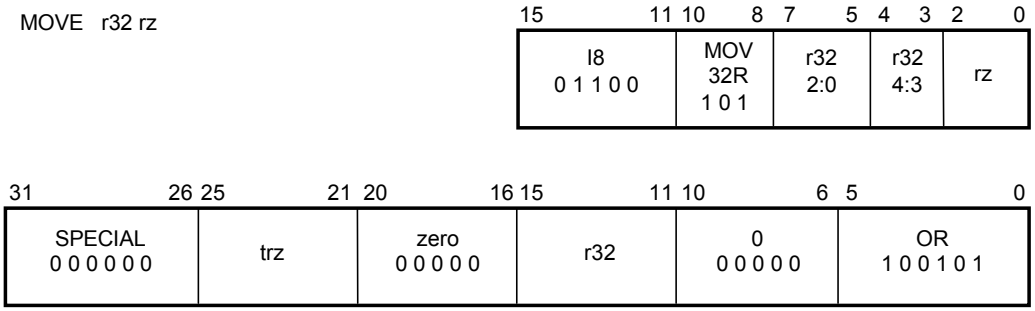
MOVE

Move

MOVE ry, r32



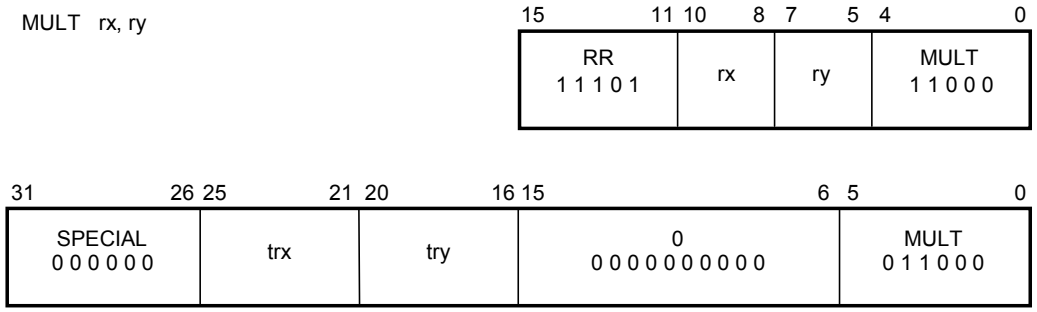
MOVE r32 rz



MULT

Multiply

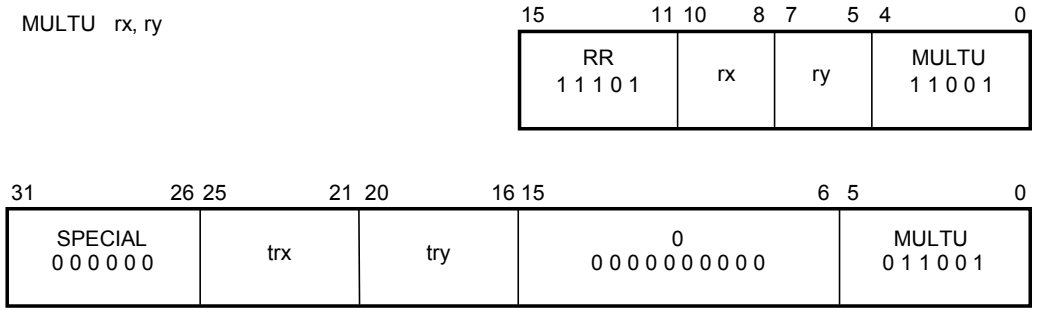
MULT rx, ry



MULTU

Multiply Unsigned

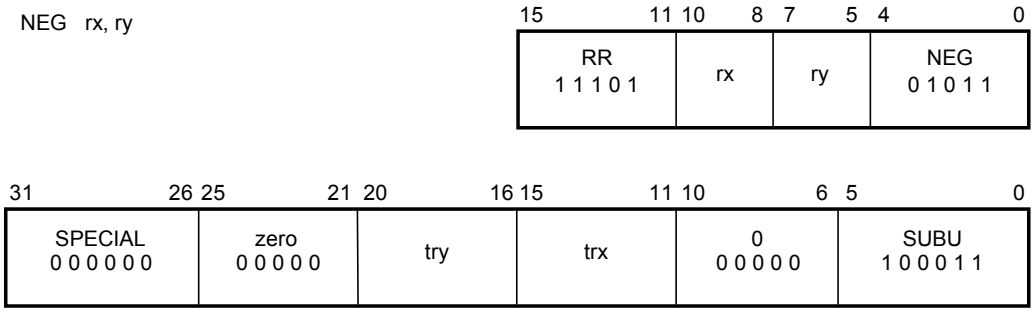
MULTU rx, ry



NEG

Negate

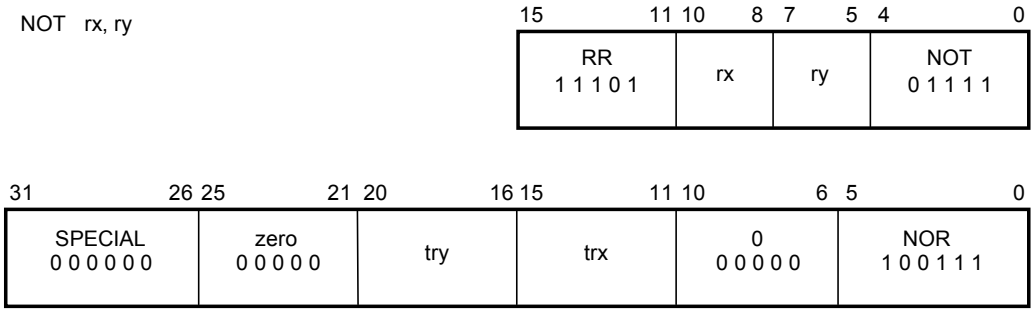
NEG rx, ry



NOT

NOT

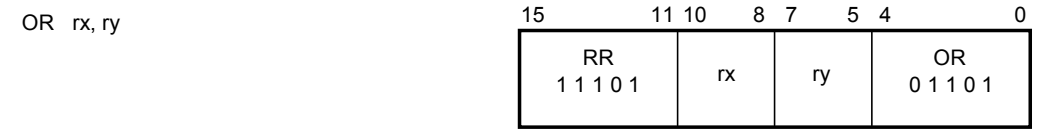
NOT rx, ry

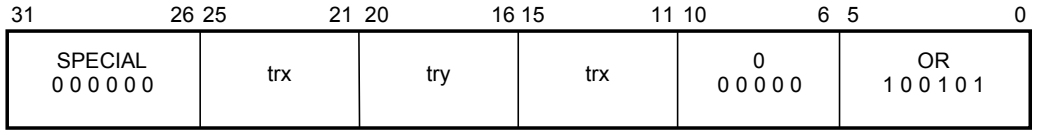


OR

OR

OR rx, ry

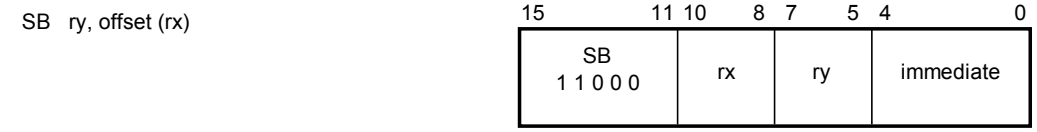


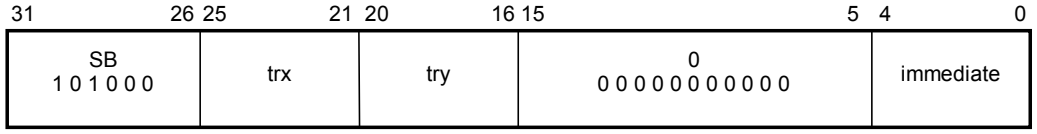


SB

Store Byte

SB ry, offset (rx)

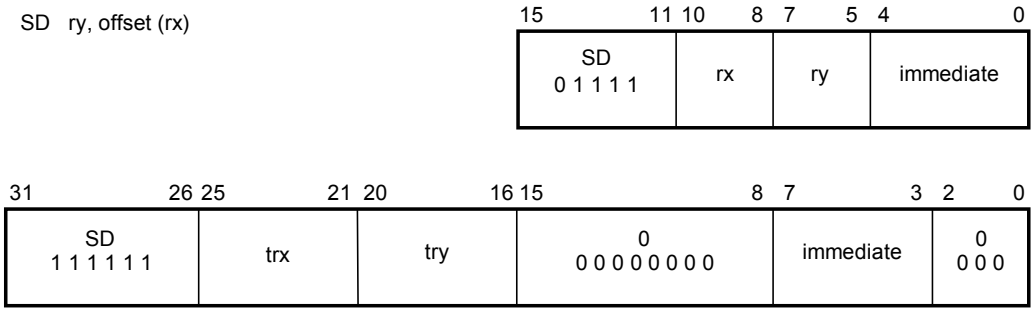




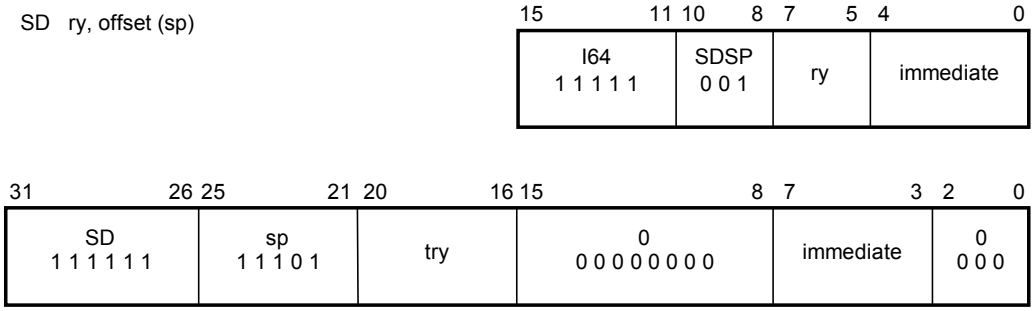
SD

Store Doubleword

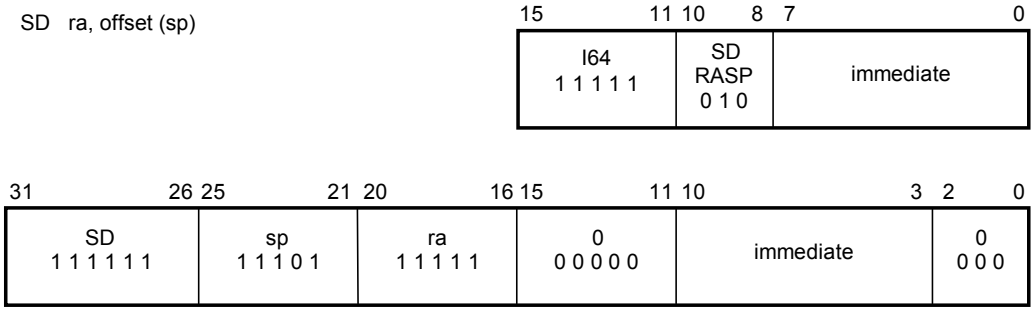
SD ry, offset (rx)



SD ry, offset (sp)



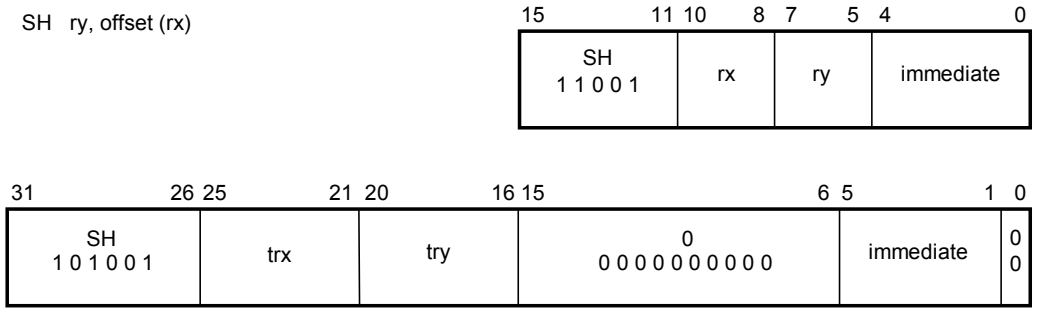
SD ra, offset (sp)



SH

Store Halfword

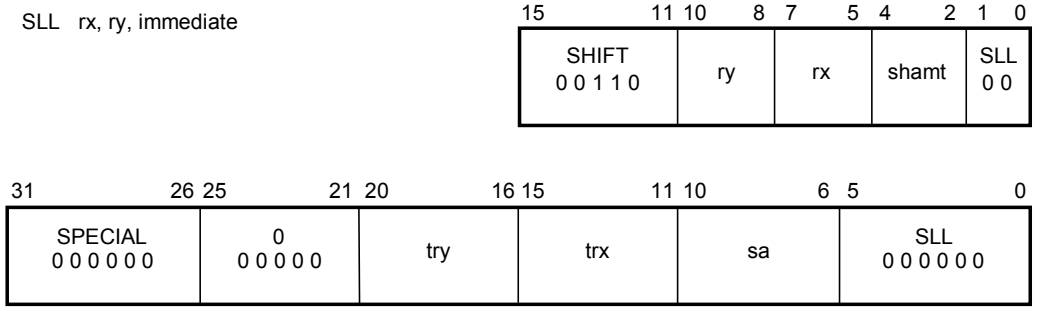
SH ry, offset (rx)



SLL

Shift Left Logical

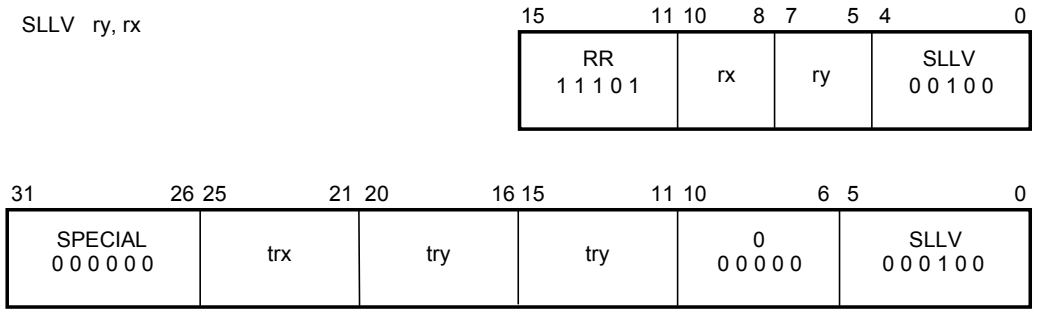
SLL rx, ry, immediate



SLLV

Shift Left Logical Variable

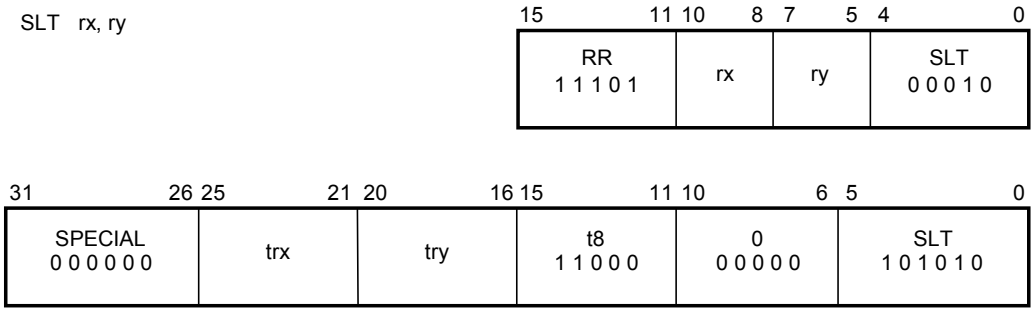
SLLV ry, rx



SLT

Set on Less Than

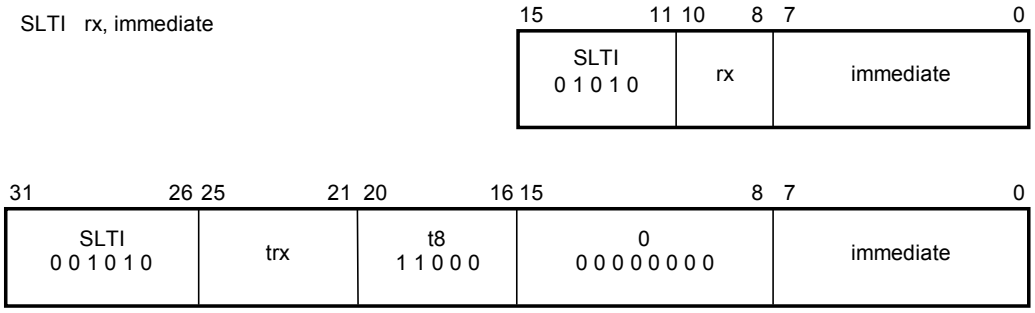
SLT rx, ry



SLTI

Set on Less Than Immediate

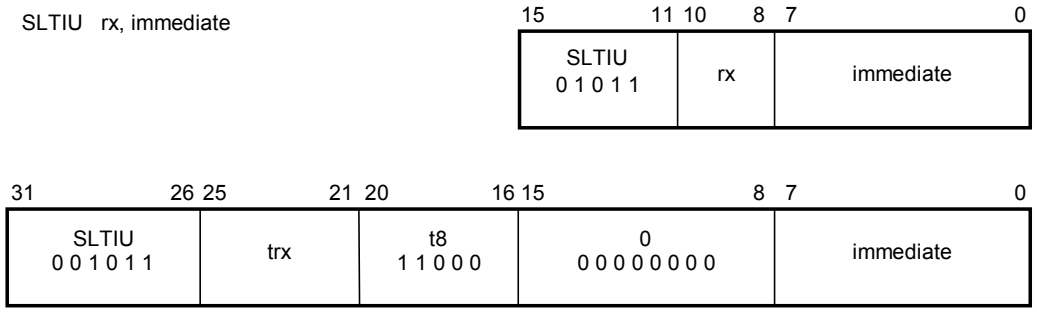
SLTI rx, immediate



SLTIU

Set on Less Than Immediate Unsigned

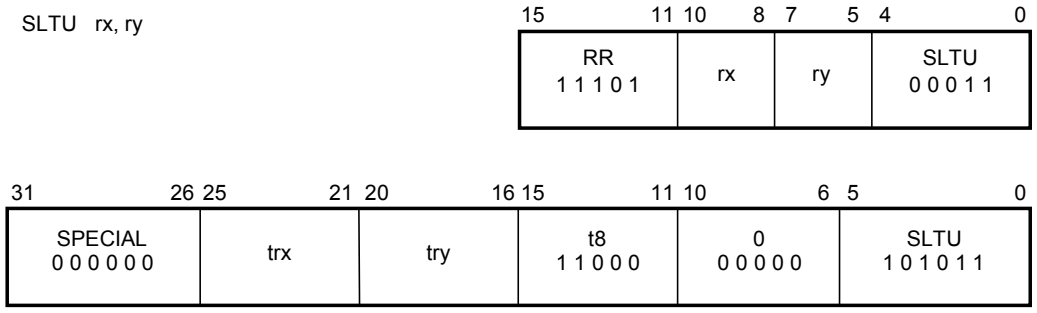
SLTIU rx, immediate



SLTU

Set on Less Than Unsigned

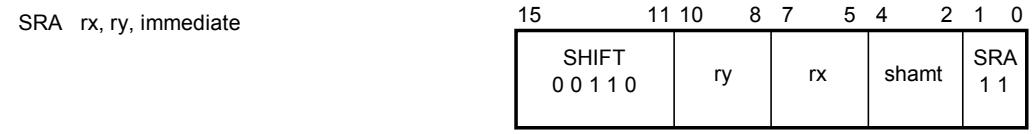
SLTU rx, ry

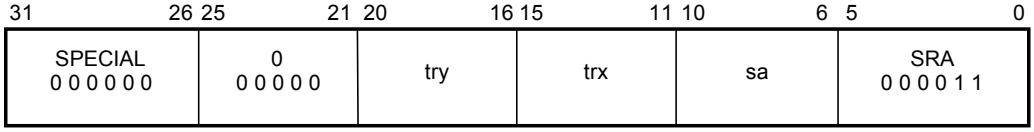


SRA

Shift Right Arithmetic

SRA rx, ry, immediate

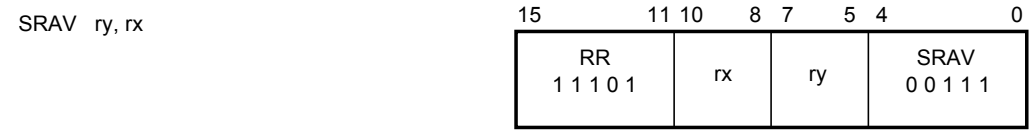


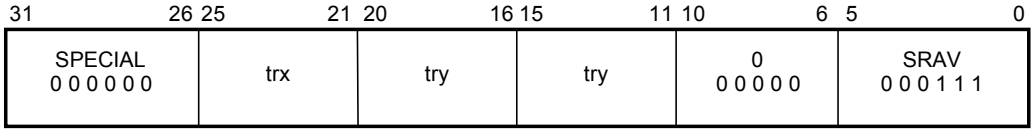


SRAV

Shift Right Arithmetic Variable

SRAV ry, rx

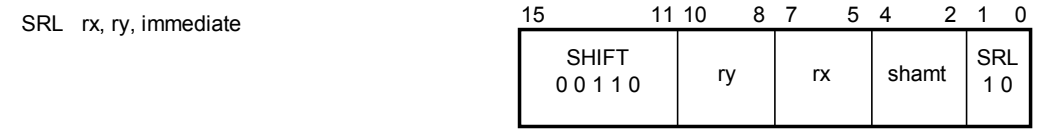


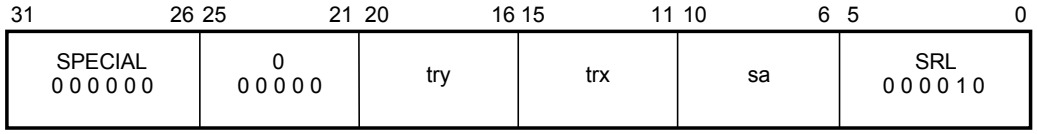


SRL

Shift Right Logical

SRL rx, ry, immediate

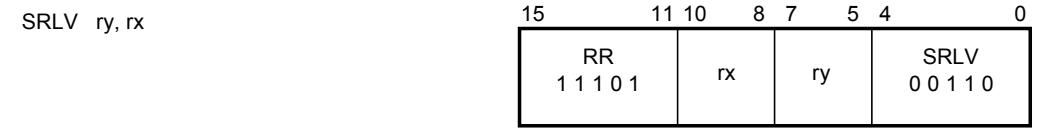


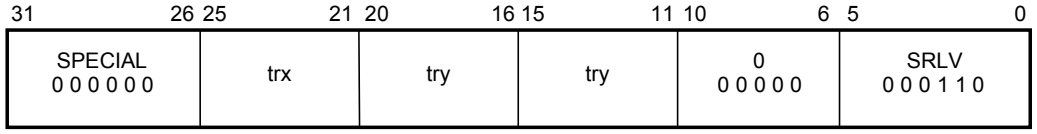


SRLV

Shift Right Logical Variable

SRLV ry, rx

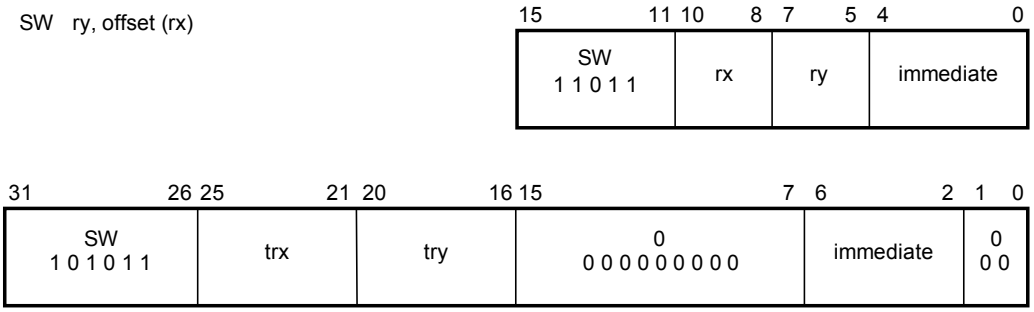




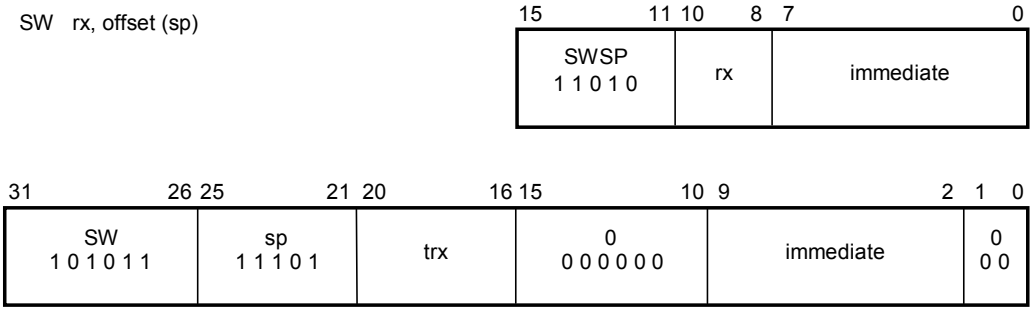
SW

Store Word

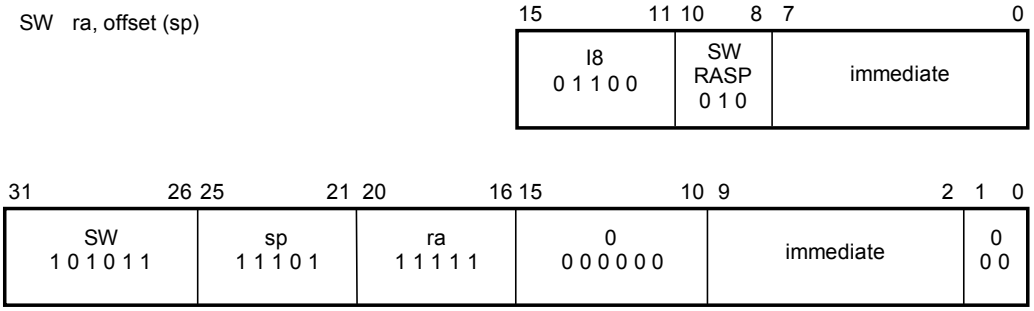
SW ry, offset (rx)



SW rx, offset (sp)



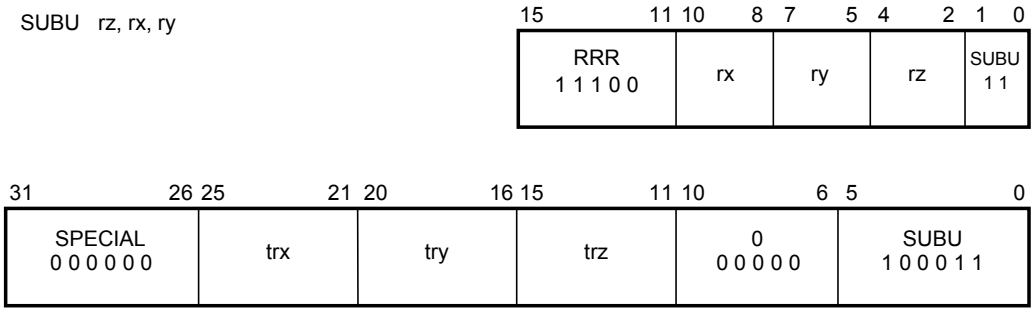
SW ra, offset (sp)



SUBU

Subtract Unsigned

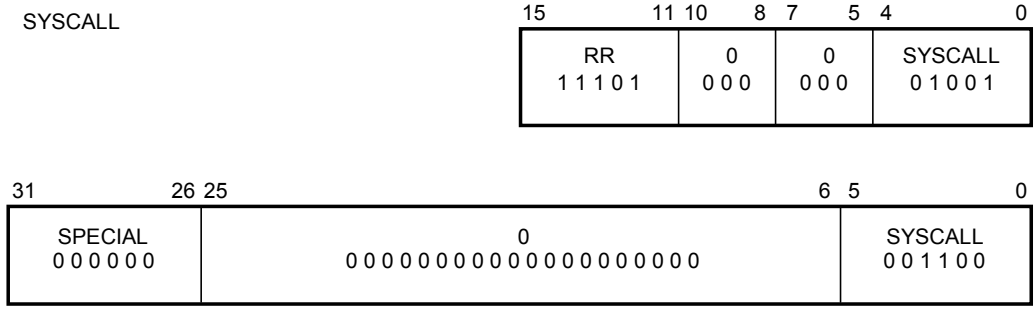
SUBU rz, rx, ry



SYSCALL

System Call

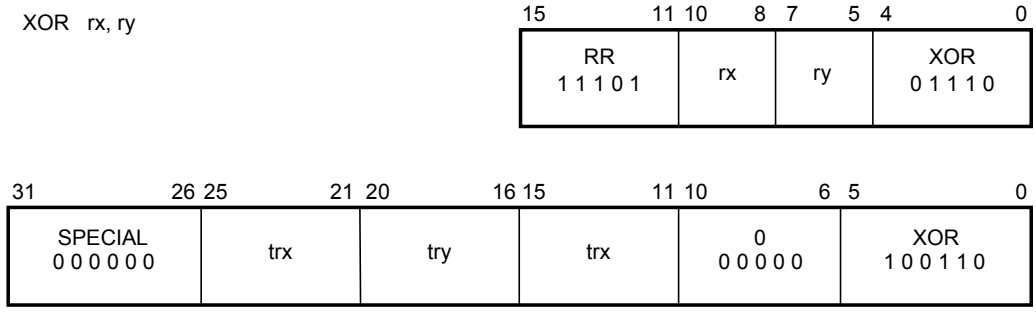
SYSCALL



XOR

Exclusive OR

XOR rx, ry



CHAPTER 11 COPROCESSOR 0 HAZARDS

The CPU core of the VR4100 Series avoids contention of its internal resources by causing a pipeline interlock in such cases as when the contents of the destination register of an instruction are used as a source in the succeeding instruction. Therefore, instructions such as NOP must not be inserted between instructions.

However, interlocks do not occur on the operations related to the CP0 registers and the TLB. Therefore, contention of internal resources should be considered when composing a program that manipulates the CP0 registers or the TLB. The CP0 hazards define the number of NOP instructions that is required to avoid contention of internal resources, or the number of instructions unrelated to contention. This chapter describes the CP0 hazards.

The CP0 hazards of the CPU core of the VR4100 Series are as or less stringent than those of the VR4000. Table 11-1 lists the Coprocessor 0 hazards of the CPU core of the VR4100 Series. Code that complies with these hazards will run without modification on the VR4000 Series.

The contents of the CP0 registers or the bits in the “Source” column of this table can be used as a source after they are fixed.

The contents of the CP0 registers or the bits in the “Destination” column of this table can be available as a destination after they are stored.

Based on this table, the number of NOP instructions required between instructions related to the TLB is computed by the following formula, and so is the number of instructions unrelated to contention:

(Destination Hazard number of A) – [(Source Hazard number of B) + 1]

As an example, to compute the number of instructions required between an MTC0 and a subsequent MFC0 instruction, this is:

(5) – (3 + 1) = 1 instruction

The CP0 hazards do not generate interlocks of pipeline. Therefore, the required number of instruction must be controlled by program.

Table 11-1. Coprocessor 0 Hazards (1/2)

(a) VR4121, VR4122, VR4181, and VR4181A

Operation	Source		Destination	
	Source Name	No. of cycles	Destination Name	No. of cycles
MTC0	–		CPR	5
MFC0	CPR	3	–	
TLBR	Index, TLB	2	PageMask, EntryHi, EntryLo0, EntryLo1	5
TLBWI TLBWR	Index or Random, PageMask, EntryHi, EntryLo0, EntryLo1	2	TLB	5
TLBP	PageMask, EntryHi	2	Index	6
ERET	EPC or ErrorEPC, TLB	2	Status[EXL], [ERL]	4
	Status	2		
CACHE Index_Load_Tag	–		TagLo, TagHi, PErr	5
CACHE Index_Store_Tag	TagLo, TagHi, PErr	3	–	
CACHE Hit ops.	cache line	3	cache line	5
Coprocessor usable test	Status[CU], [KSU], [EXL], [ERL]	2	–	
Instruction fetch	EntryHi[ASID], Status[KSU], [EXL], [ERL], [RE], Config[K0]	2	–	
	TLB	2		
Instruction fetch exception	–		EPC, Status	4
			Cause, BadVAddr, Context, XContext	5
Interrupt signals	Cause[IP], Status[IM], [IE], [EXL], [ERL]	2	–	
Loads/Stores	EntryHi[ASID], Status[KSU], [EXL], [ERL], [RE], Config[K0], TLB	3	–	
	Config[AD], [EP]	3		
	WatchHi, WatchLo	3		
Load/Store exception	–		EPC, Status, Cause, BadVAddr, Context, XContext	5
TLB shutdown (VR4181 only)	–		Status[TS]	2 (Inst.), 4 (Data)

Remark Brackets indicate a bit name or a field name of registers.

Table 11-1. Coprocessor 0 Hazards (2/2)

(b) VR4131

Operation	Source		Destination	
	Source Name	No. of cycles	Destination Name	No. of cycles
MTC0	–		CPR	6
MFC0	CPR	4	–	
TLBR	Index, TLB	3	PageMask, EntryHi, EntryLo0, EntryLo1	6
TLBWI TLBWR	Index or Random, PageMask, EntryHi, EntryLo0, EntryLo1	3	TLB	6
TLBP	PageMask, EntryHi	3	Index	6
ERET	EPC or ErrorEPC, TLB	5	Status[EXL], [ERL]	6
	Status	5		
CACHE Index_Load_Tag	–		TagLo, TagHi, PErr	6
CACHE Index_Store_Tag	TagLo, TagHi, PErr	4	–	
CACHE Hit ops.	cache line	4	cache line	6
Coprocessor usable test	Status[CU], [KSU], [EXL], [ERL]	2	–	
Instruction fetch	EntryHi[ASID], Status[KSU], [EXL], [ERL], [RE], Config[K0]	2	–	
	TLB	2		
Instruction fetch exception	–		EPC, Status	6
			Cause, BadVAddr, Context, XContext	6
Interrupt signals	Cause[IP], Status[IM], [IE], [EXL], [ERL]	2	–	
Loads/Stores	EntryHi[ASID], Status[KSU], [EXL], [ERL], [RE], Config[K0], TLB	4	–	
	Config[AD], [EP]	4		
	WatchHi, WatchLo	4		
Load/Store exception	–		EPC, Status, Cause, BadVAddr, Context, XContext	6

Remark Brackets indicate a bit name or a field name of registers.

Cautions 1. If the setting of the K0 bit in the Config register is changed by MTC0 for the kseg0 or ckseg0 area, the change is reflected at first to third instruction after MTC0.

2. The instruction following MTC0 must not be MFC0.

3. The five instructions following MTC0 to Status register that changes KSU bit and sets EXL and ERL bits may be executed in the new mode, and not Kernel mode. This can be avoided by setting EXL bit first, leaving KSU bit set to Kernel, and later changing KSU bit.

4. If interrupts are disabled by setting EXL bit in the Status register with MTC0, an interrupt may occur immediately after MTC0 without change of the contents of the EPC register. This can be avoided by clearing IE bit first, and later setting EXL bit.

5. There must be two non-load, non-CACHE instructions between a store and a CACHE instruction directed to the same primary cache line as the store.

The status during execution of the following instruction for which CP0 hazards must be considered is described below.

(1) MTC0

Destination: The completion of writing to a destination register (CP0) of MTC0.

(2) MFC0

Source: The confirmation of a source register (CP0) of MFC0.

(3) TLBR

Source: The confirmation of the status of TLB and the Index register before the execution of TLBR.

Destination: The completion of writing to a destination register (CP0) of TLBR.

(4) TLBWI, TLBWR

Source: The confirmation of a source register of these instructions and registers used to specify a TLB entry.

Destination: The completion of writing to TLB by these instructions.

(5) TLBP

Source: The confirmation of the PageMask register and the EntryHi register before the execution of TLBP.

Destination: The completion of writing the result of execution of TLBP to the Index register.

(6) ERET

Source: The confirmation of registers containing information necessary for executing ERET.

Destination: The completion of the processor state transition by the execution of ERET.

(7) CACHE Index_Load_Tag

Destination: The completion of writing the results of execution of this instruction to the related registers.

(8) CACHE Index_Store_Tag

Source: The confirmation of registers containing information necessary for executing this instruction.

(9) Coprocessor usable test

Source: The confirmation of modes set by the bits of the CP0 registers in the “Source” column.

Examples 1. When accessing the CP0 registers in User mode after the CU0 bit of the Status register is modified, or when executing an instruction such as TLB instructions, CACHE instructions, or Branch instructions that use the resource of the CP0.

2. When accessing the CP0 registers in the operating mode set in the Status register after the KSU, EXL, and ERL bits of the Status register are modified.

(10) Instruction fetch

Source: The confirmation of the operating mode and TLB necessary for instruction fetch.

Examples 1. When changing the operating mode from User to Kernel and fetching instructions after the KSU, EXL, and ERL bits of the Status register are modified.

2. When fetching instructions using the modified TLB entry after TLB modification.

(11) Instruction fetch exception

Destination: The completion of writing to registers containing information related to the exception when an exception occurs on instruction fetch.

(12) Interrupts

Source: The confirmation of registers judging the condition of occurrence of interrupt when an interrupt factor is detected.

(13) Loads/Sores

Source: The confirmation of the operating mode related to the address generation of Load/Store instructions, TLB entries, the cache mode set in the K0 bit of the Config register, and the registers setting the condition of occurrence of a Watch exception.

Example When Loads/Stores are executed in the kernel field after changing the mode from User to Kernel.

(14) Load/Store exception

Destination: The completion of writing to registers containing information related to the exception when an exception occurs on load or store operation.

(15) TLB shutdown (VR4181 only)

Destination: The completion of writing to the TS bit of the Status register when a TLB shutdown occurs.

Table 11-2 indicates examples of calculation.

Table 11-2. Calculation Example of CP0 Hazard and Number of Instructions Inserted

Destination	Source	Contending internal resource	Number of instructions inserted		Formula	
			VR4121, VR4122, VR4181, VR4181A	VR4131	VR4121, VR4122, VR4181, VR4181A	VR4131
TLBWR/TLBWI	TLBP	TLB Entry	2	2	5 – (2 + 1)	6 – (3 + 1)
TLBWR/TLBWI	Load or Store using newly modified TLB	TLB Entry	1	1	5 – (3 + 1)	6 – (4 + 1)
TLBWR/TLBWI	Instruction fetch using newly modified TLB	TLB Entry	2	3	5 – (2 + 1)	6 – (2 + 1)
MTC0 Status [CU]	Coprocessor instruction that requires the setting of CU	Status [CU]	2	3	5 – (2 + 1)	6 – (2 + 1)
TLBR	MFC0 EntryHi	EntryHi	1	1	5 – (3 + 1)	6 – (4 + 1)
MTC0 EntryLo0	TLBWR/TLBWI	EntryLo0	2	2	5 – (2 + 1)	6 – (3 + 1)
TLBP	MFC0 Index	Index	2	1	6 – (3 + 1)	6 – (4 + 1)
MTC0 EntryHi	TLBP	EntryHi	2	2	5 – (2 + 1)	6 – (3 + 1)
MTC0 EPC	ERET	EPC	2	0	5 – (2 + 1)	6 – (5 + 1)
MTC0 Status	ERET	Status	2	0	5 – (2 + 1)	6 – (5 + 1)
MTC0 Status [IE] Note	Instruction that causes an interrupt	Status [IE]	2	0	5 – (2 + 1)	6 – (5 + 1)

Note The number of hazards is undefined if the instruction execution sequence is changed by exceptions. In such a case, the minimum number of hazards until the IE bit value is confirmed may be the same as the maximum number of hazards until an interrupt request occurs that is pending and enabled.

Remark Brackets indicate a bit name or a field name of registers.

APPENDIX INDEX

[MEMO]

NEC

Facsimile Message

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.

From:

Name

Company

Tel. FAX

Address

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
Market Communication 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

P.R. China
NEC Electronics Shanghai, Ltd.
Fax: +86-21-6841-1137

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	❑	❑	❑	❑