NEC

User's Manual

VR5432™

64-Bit MIPS® RISC Microprocessor

Volume 1

μPD30541GD

Document No.U13751EU5V0UMJ1
Date Published: May 2001 CP (K)

NEC

VR5432 Microprocessor User's Manual
Document Number U13751EU5V0UMJ1

Revision History

February 1999: First release
August 1999: Version 2, Preliminary
December 1999: Version 3, Preliminary update
February 2000: Version 4, Document release
May 2000: Version 5, Document corrections

Contents

Volume 1

Preface

The VR5432™ microprocessor is an NEC VR Series™ RISC (reduced instruction set computer) microprocessor that implements the high-performance 64-bit MIPS® IV architecture. This manual describes the architecture and hardware functions of the VR5432 microprocessor.

Legend

Data significance:	Higher on left and lower on right
Active-high signal name:	XXX
Active-low signal name:	XXX*
Numeric representation:	binary ... XXXX or $XXXX_2$
	decimal ... XXXX
	hexadecimal ... 0xXXXX

Prefixes representing an exponent of 2 (for address space or memory capacity):

K (kilo)	$2^{10} = 1024$
M (mega)	$2^{20} = 1024^2$
G (giga)	$2^{30} = 1024^3$
T (tera)	$2^{40} = 1024^4$

Manual Overview

The manual is divided into two volumes. Volume 1 is the user manual, containing processor architectural and functional information and instructions. Volume 2 contains the instruction set information and appendixes.

Volume 1 (U13751E)

Chapter 1: Introduction provides an overview of the device features, CPU, Floating-Point Unit (FPU), and pipeline.

Chapter 2: Signal Descriptions discusses the pin configuration and functions of the VR5432 processor signals.

Chapter 3: Pipeline describes the dual-issue instruction pipeline stages, delays, and interlock and exception handling.

Chapter 4: Memory Management Unit discusses the processor's virtual and physical address spaces, the virtual-to-physical address translation, the translation lookaside buffer (TLB) process, and the system control coprocessor registers that provide the software interface to the TLB.

Chapter 5: Cache Organization and Operation describes the cache memory's place in the VR5432 memory configuration and individual cache organization.

Chapter 6: CPU Exceptions describes the processor's exception types, registers, vector offsets, processing handling, and interrupts.

Chapter 7: Floating-Point Unit describes the FPU coprocessor, including the programming model, instruction set and formats, and the pipeline.

Chapter 8: Floating-Point Exceptions discusses FPU exception types, exception trap processing, exception flags, saving and restoring states when handling an exception, and trap handlers for IEEE Standard 754 exceptions.

Chapter 9: Bus Interface describes how the processor accesses the external resources needed to satisfy cache misses and uncached operations, while permitting an external agent access to some of the processor's internal resources.

Chapter 10: System Interface Transactions (Native Mode) describes processor and external requests in the native system interface protocol of the VR5432 processor.

Chapter 11: System Interface Protocols (Native Mode) contains a cycle-by-cycle description of the system interface protocols for each type of processor and external request in the native protocol of the VR5432 processor.

Chapter 12: System Interface Transactions (R43K Mode) This section describes processor and external requests as they occur in R43K (VR4300 compatibility) mode.

Chapter 13: System Interface Protocols (R43K Mode) contains a cycle-by-cycle description of the system interface protocols for each type of processor and external request in R43K mode.

Chapter 14: Initialization Interface describes the processor reset and initialization signals.

Chapter 15: Clock Interface describes the basic system clocks, SysClock and PClock, and Phase-Locked Loop (PLL) and Bypass PLL modes.

Volume 2 (U15397E)

Chapter 16: Instruction Set Overview discusses the general attributes of the CPU, FPU, multimedia, and debugging instructions of the MIPS IV instruction set architecture (ISA) utilized by the VR5432 processor.

Chapter 17: CPU Instruction Set describes the details of the CPU instructions.

Chapter 18: Floating-Point Unit Instruction Set describes the details of the FPU instructions.

Chapter 19: Multimedia Instruction Set describes the details of the multimedia instructions.

Chapter 20: Debug and Test Features describes the VR5432 processor's debug and test functions, Debug mode, and debug instructions.

Appendix A: Sublock Order describes how a block of data elements (bytes, halfwords, words, or doublewords) can be retrieved from storage in sequential or nonsequential (sub-block) order.

Appendix B: Comparing the VR4300, VR5000, and VR5432 Processors delineates each processor's attributes.

Appendix C: PLL Analog Power Filtering illustrates the phase-locked loop circuit configuration.

Appendix D: Instruction Hazards identifies the VR5432 instruction hazards that occur with certain instruction and event combinations (such as pipeline delays, cache misses, interrupts, and exceptions).

Related Documents

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

Product	Data Sheet	User's Manual	
		Hardware Architecture	Instruction Set
VR5432	U13504E	U13751E	U15397E
VR5000	U12031E	U11761E	U12754E
VR10000	U12703E	U10278E	U12754E

Introduction

1

The VR5432™ microprocessor is an NEC VR Series™ RISC (reduced instruction set computer) microprocessor that implements the high-performance 64-bit MIPS® IV architecture.

The instruction set for the VR5432 processor is compatible with those of the VR3000™ and VR4000™ microprocessor families, which are based on the MIPS III architecture. It is also compatible with the MIPS IV architecture used on the VR5000™ and VR10000™ microprocessors. Therefore, existing applications can be implemented easily with the VR5432 processor.

The VR5432 processor (part number μPD30541GD) implements a 32-bit system interface, which can operate in both VR5432 Native mode and R43K mode (which emulates the interface used in the VR4300™ microprocessor). For most VR4300 applications, the VR5432 offers substantial performance improvement, minimal product redesign, and fast time to market.

1.1 Device Features

The VR5432 has the following features:

- MIPS IV instruction set with MACC (multiply and accumulate) and DSP/multimedia extension
- 0.25-µm static CMOS technolog
- 64-bit architecture with 32-bit split-transaction external data bus
- Dual-issue superscalar implementatio
- 4K-entry branch prediction table with 2-bit saturating counter mechanism
- 4-entry nonblocking data cache miss queue
- 4-entry transaction buffer (4 doublewords total
- 32 KB, 2-way set-associative, line-locked, 32-byte/block instruction cache
- 32 KB, 2-way set-associative, write-through, write-back, line-locked 32-byte/block data cache
- 48-entry translation lookaside buffer (TLB), mapping two pages per entry
- 4-entry instruction micro-TLB
- 4-entry data micro-TLB
- 40-bit virtual address space
- Physical address space: 36 bits are internal; the lower 32 bits are external
- Single- and double-precision IEEE-754 floating-point operations
- Up to 83 MHz external bus with on-chip clock multiplier for internal frequency of ×2, ×2.5, ×3, or ×4 the external clock
- N-Wire and N-Trace hardware debugging interfac
- Upward compatibility with VR300 ™ and VR4000 devices
- Bus protocol compatibility mode for R4300 system interface
- 2.5 V core logic with 3.3 V external interfac

1.2 Internal Architecture

1.2.1 Configuration

Figure 1-1 is an internal block diagram of the VR5432 processor. Descriptions of each block follow.

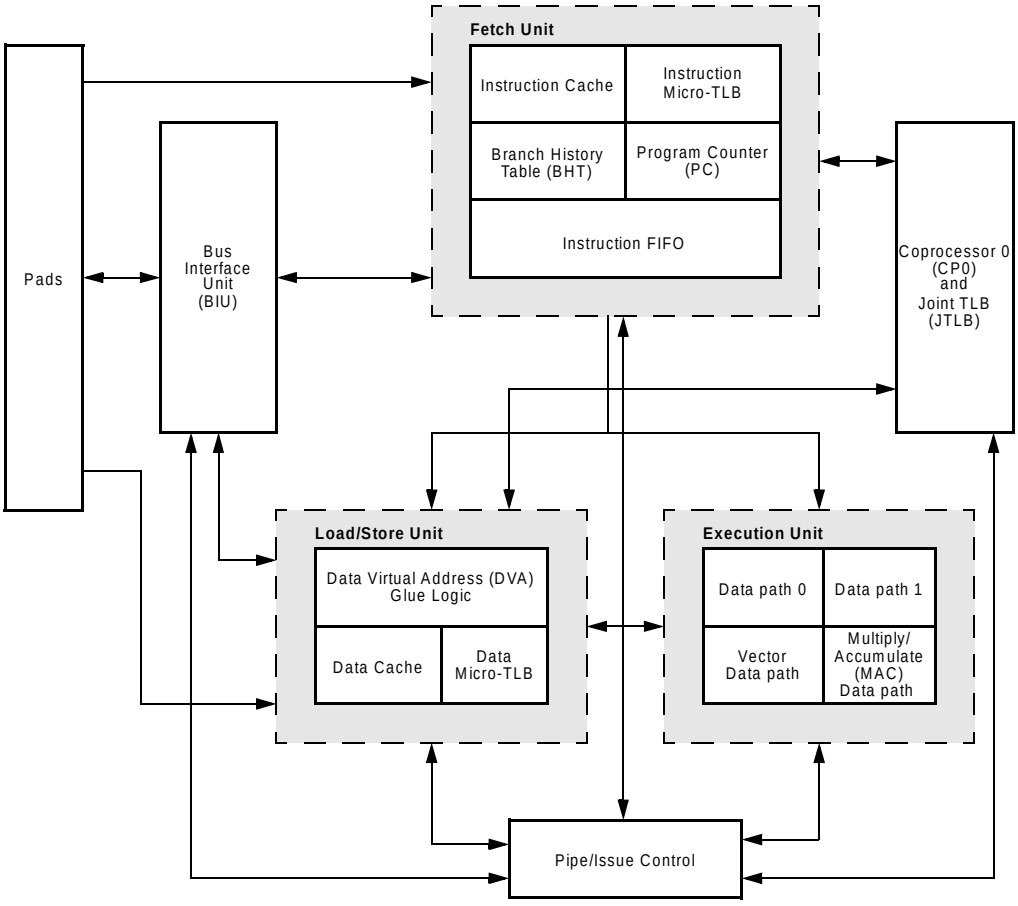


Figure 1-1 Block Diagram

Fetch Unit

The fetch unit transfers instructions from the instruction cache to the load/store unit (LSU) or the execution unit. The fetch unit includes the following elements.

Instruction Cache. The instruction cache is 32 KB and 2-way set associative, with line-locking capability. It has a 64-bit read and a 64-bit write port. It is always accessed on a doubleword boundary, providing two instructions each time it is accessed.

Instruction Micro-TLB. The instruction micro-TLB (ITLB) has four entries, each of which supports a variable page size and is refilled from the joint TLB (JTLB) using a pseudo LRU algorithm. The instruction virtual address (IVA) is translated by the ITLB. This TLB is completely invisible to software.

Branch History Table. The branch history table (BHT) has one entry per instruction cache doubleword. The BHT implements a 2-bit saturating counter branch prediction scheme.

Program Counter. The program counter (PC) keeps track of the program flow and generates new program counters. The new program counters access instructions from the instruction cache or from the main memory.

Instruction FIFO. This FIFO isolates instruction cache accesses from instruction issue logic. As mentioned above, the VR5432 processor can fetch up to two instructions per cycle. In some cases, both of these instructions cannot be issued. The FIFO holds these instructions until they are ready to be issued; however, the fetch unit continues to access instructions during this time.

Load/Store Unit

The load/store unit (LSU) loads and stores data to/from the data cache or main memory. This unit also contains data alignment logic for Load instructions. The LSU includes the following elements.

Data Virtual Address. When the LSU receives a Load/Store instruction from the fetch unit, it calculates an address from which data is to be retrieved or where data is to be stored. This address is the data virtual address (DVA). The LSU contains the adder that calculates this address.

Data Cache. The data cache is 32 KB and 2-way set associative, with line-locking capability. It has a 64-bit read and a 64-bit write port. It is always accessed on a doubleword boundary, providing a doubleword each time it is accessed.

Data Micro-TLB. The data micro-TLB (DTLB) has four entries, each of which supports a variable page size and is refilled from the joint TLB (JTLB) using a pseudo LRU algorithm. The DVA is translated by the DTLB. This TLB is completely invisible to software.

Execution Unit

The VR5432 is a superscalar microprocessor that can issue two instructions simultaneously. There are four execution datapaths in this processor. Data path 0 (DP0) and Data path 1 (DP1) execute both integer and floating-point instructions. The multiply/accumulate (MAC) data path executes VR5432 processor-specific MAC instructions. The vector data path works on VR5432 processor-specific multimedia instructions.

Pipe/Issue Control

Pipeline flow and instruction execution are handled by the logic, which resides in this block. This block also contains control logic, which handles interruptions that affect the pipeline flow.

Coprocessor 0 and Joint TLB

The VR5432 processor executes the MIPS instruction set. MIPS architecture defines several control and status registers, which are used to control the processor. These registers reside in the Coprocessor 0 (CP0) block. The processor also includes a 48-entry, MIPS-compatible JTLB, which supports even/odd page sizes.

Bus Interface Unit

The bus interface unit (BIU) minimizes processor stalls and maximizes bus utilization. The BIU isolates the bus from the processor core and makes the most efficient use of the bus, performing burst transfers whenever possible.

1.2.2 CPU Registers

The processor provides the following registers:

- 32 64-bit general-purpose registers (GPR
- 32 64-bit floating-point registers (FPR

In addition, the processor provides the following special registers:

- 64-bit HI register, to receive the integer multiply an divide high-order doubleword result
- 64-bit LO register, to receive the integer multiply and divide low-order doubleword result
- 1-bit Load/Link (LL) Bit register
- 192-bit Accumulator register, for operating on media instruction
- 32-bit floating-point Implementation/Revision register (FCR0
- 32-bit floating-point Control/Status register (FCR31)

Two of the CPU general-purpose registers have assigned functions:

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

Furthermore, the processor contains dedicated registers in the system control processor (CP0) that support exception processing and address management. CPU registers can operate as either 32- or 64-bit registers, depending on the VR5432 processor operation mode.

Figure 1-2 shows the VR5432 processor registers.

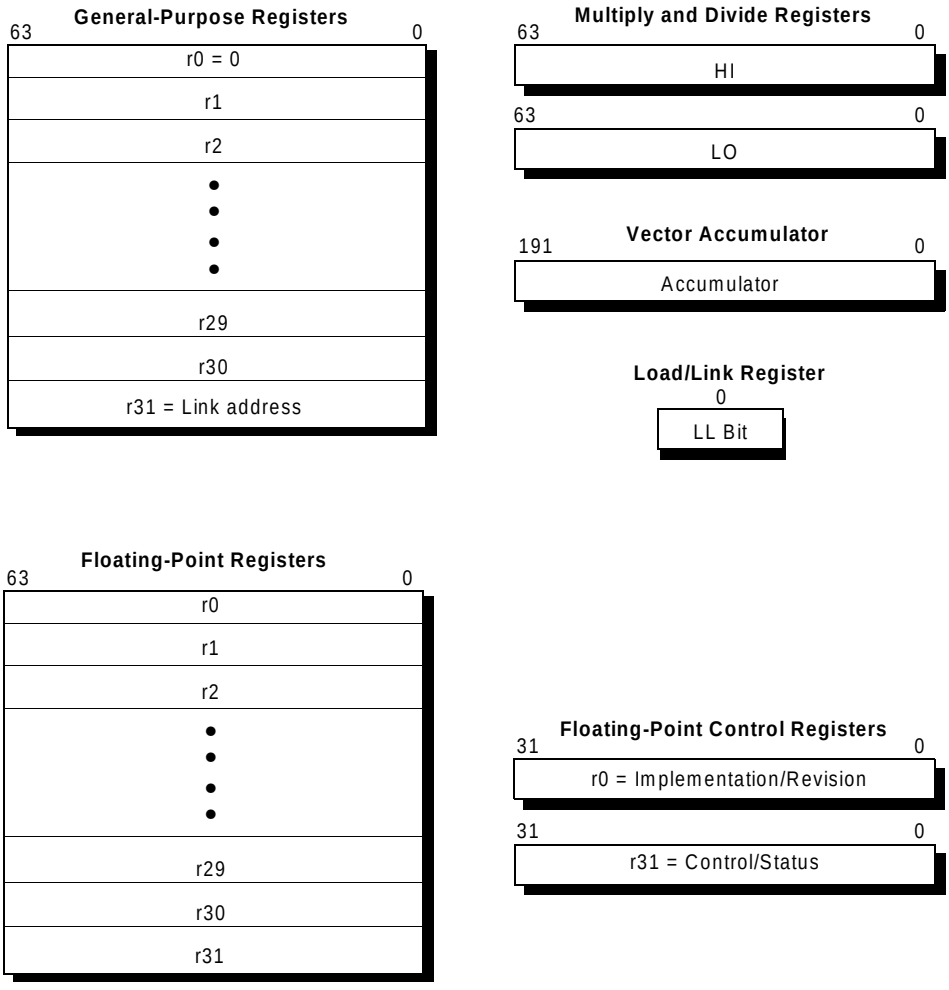


Figure 1-2 VR5432 Processor Registers

The VR5432 processor has no Program Status Word (PSW) register as such; this function is covered by the Status and Cause registers included within the system control coprocessor (CP0). CP0 registers are described later in this chapter.

1.2.3 CPU Instruction Set Overview

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

- Immediate (I-type)
- Jump (J-type)
- Register (R-type)

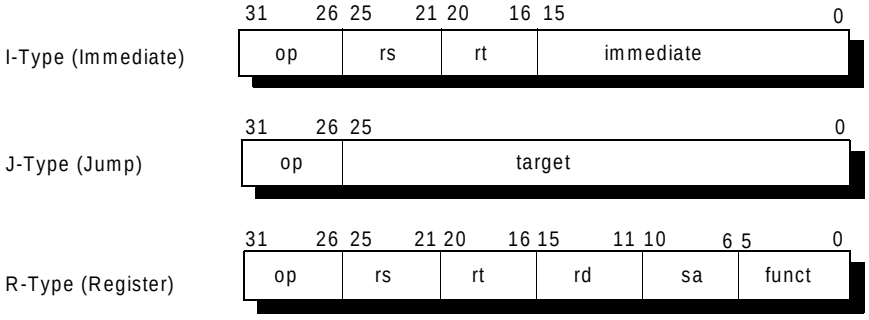


Figure 1-3 CPU Instruction Formats

The instructions can be further classified as follows.

- **Load** and **Store** instructions move data between memory and general-purpose registers. They are all immediate (I-type instructions, since the only addressing mode supported for general-purpose-registers is base register plus 16-bit, signed immediate offset (Indexed addressing is supported for floating-point registers.)
- **Computational** instructions perform arithmetic, logical, shift multiply, and divide operations on values in registers. They includ register (R-type, in which both the operands and the result are stored in registers) and immediate (I-type, in which one operand is a 16-bit signed immediate value) formats.
- **Jump** and **Branch** instructions change the control flow of a program. Jumps are always made to an address formed by combining a 26-bit target address with the high-order bits of the program counter (J-type format) or register address (R-type format). Branch instructions ar performed to the 16-bit offset address relative to the program counter (I-type). Jump and Link instructions save their return address in register 31.
- **Coprocessor** instructions were originally defined in the MIP architecture to perform operations in coprocessors. However, “true” coprocessors (in the sense of functional units that operate in parallel with the CPU) are not implemented. For example, the original coprocessor 1 was the FPU, but in the VR5432 the FPU instructions are executed using the same data paths used for integer instructions The Coprocessor 2 opcodes have been used for the multimedi instruction set extensions
- **Coprocessor 0** (system coprocessor, CP0) instructions perfor operations on CP0 registers to control the memory management and exception handling facilities of the processor.
- **Exception** instructions perform System Call exception an Breakpoint exception operations, cause a branch to the general exception handling vector based upon the result of a comparison, or implement the MAC instruction set extensions. These instructions occur in both R-type (both the operands and the result are registers) and I-type (one operand is a 16-bit immediate value) formats

Refer to Chapter 16 for an overview of instructions, Chapter 17 for CPU instruction details, Chapter 18 for FPU instruction details, Chapter 19 for multimedia instruction details, and Chapter 20 for debug and test instructions.

1.2.4 Data Formats and Addressing

The VR5432 processor uses four data formats: a 64-bit doubleword, a 32-bit word, a 16-bit halfword, and an 8-bit byte. Byte ordering within all of the larger data formats—halfword, word, doubleword—can be configured as either big or little endian. When the VR5432 processor is configured as a big-endian system, byte 0 is the most-significant (left-most) byte, thereby providing compatibility with Motorola's MC 68000™ and IBM's System/370™ conventions. Figure 1-4 shows this configuration.

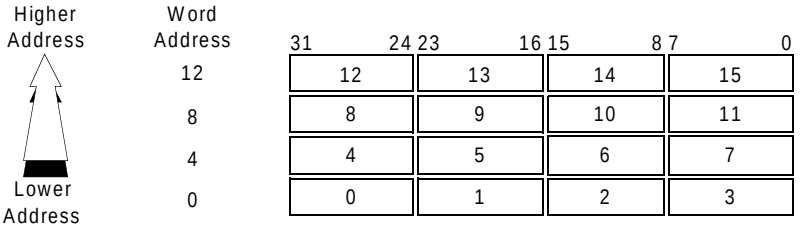


Figure 1-4 Big-Endian Byte Ordering

Note: The most-significant byte has the lowest address. A word is addressed by the address of the most-significant byte.

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

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

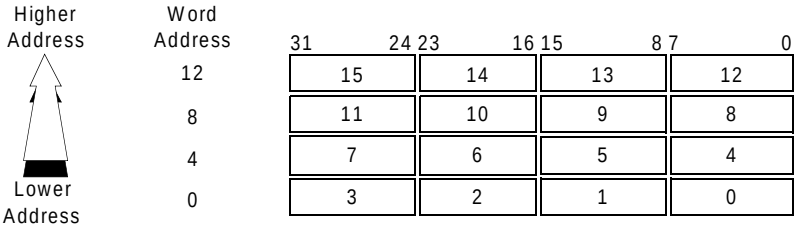


Figure 1-5 Little-Endian Byte Ordering

Note: The least-significant byte has the lowest address. A word is addressed by the address of the least-significant byte.

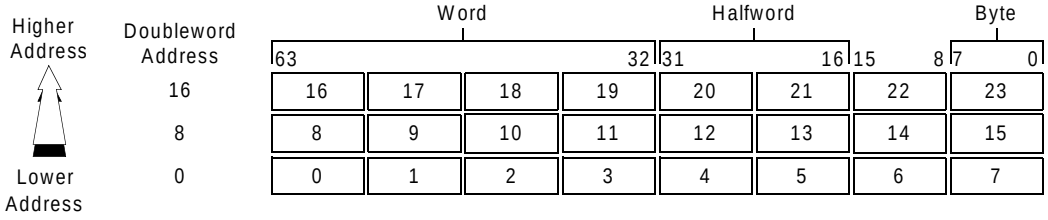


Figure 1-6 Big-Endian Data in a Doubleword

Note: The most-significant byte has the lowest address. A word is addressed by the address of the most-significant byte.

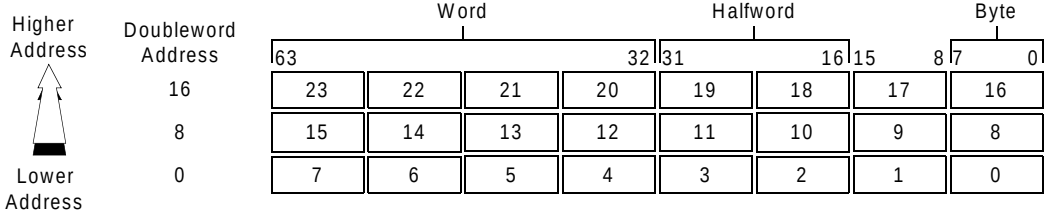


Figure 1-7 Little-Endian Data in a Doubleword

Note: The least-significant byte has the lowest address. A word is addressed by the address of the least-significant byte.

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

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

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

LWL LWR SWL SWR

LDL LDR SDL SDR

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

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

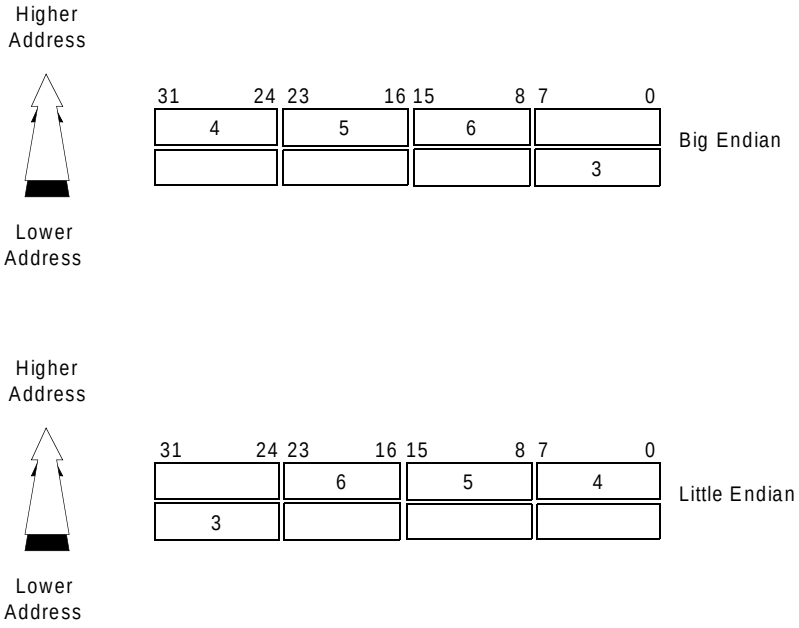


Figure 1-8 Misaligned Word Addressing

1.2.5 System Control Coprocessor (CP0)

The original MIPS architecture defined up to four coprocessors (CP0 through CP3). Coprocessor 1 was the FPU, and coprocessors 2 and 3 were reserved for future use. However, with the introduction of the MIPS IV instruction set, the coprocessor 3 opcodes were assigned to extensions of the floating-point instruction set. On the VR5432 (and specific to this implementation), the coprocessor 2 opcodes have been used for multimedia instruction set extensions. Coprocessor 0 (CP0) remains the internal system control coprocessor and supports the virtual memory system and exception processing. The virtual memory system uses the on-chip TLB and CP0 registers.

CP0 converts virtual addresses into physical addresses, controls the operating mode (Kernel, Supervisor, or User mode), and performs exception processing. It also controls the cache subsystem to analyze causes and return execution from error processing. CP0 also includes implementation-dependent test and debug features.

Figure 1-9 shows the CP0 registers; Table 1-1 briefly explains each register. For the details of the registers related to the virtual memory system, refer to Chapter 4; for the details of the registers used for exception processing, refer to Chapter 6**.**

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

* For memory management
** For exception processing
RFU Reserved for future use

Figure 1-9 CP0 Registers

Table 1-1 System Control Coprocessor (CP0) Register Definitio n

Register Number	Register Name	Description
0	Index	Index into the TLB
1	Random	Random pointer to the TLB
2	EntryLo0	Even PFN and page attributes for TLB entry
3	EntryLo1	Odd PFN and page attributes for TLB entry
4	Context	Pointer to kernel PTE (32-bit addressing)
5	PageMask	TLB page size mask
6	Wired	Number of wired (locked) TLB entries
7	—	Unused
8	BadVAddr	Bad virtual address
9	Count	Timer count
10	EntryHi	VPN and ASID for TLB entry
11	Compare	Timer compare
12	SR	Status register
13	Cause	Cause of last exception
14	EPC	Exception program counter
15	PRId	Processor revision identifier
16	Config	Configuration register
17	LLAddr	Load linked address
18	WatchLo	Memory reference trap address lower bits
19	WatchHi	Memory reference trap address upper bits
20	XContext	Pointer to the kernel PTE (64-bit addressing)
21–24	—	Unused
25	PerfCtr	Performance counter registers: Performance Event Specifier/Control 0 Performance Counter 0 Performance Event Specifier/Control 1 Performance Counter 1
26	PErr	Parity error in cache
27	CacheErr	Cache error register
28	TagLo	Cache tag register

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

Register Number	Register Name	Description
29	TagHi	Cache tag register (reserved)
30	ErrorEPC	Error exception program counter
31	—	Unused

1.2.6 Floating-Point Unit (FPU)

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

The FPU includes:

- **Full 64-bit operation**. The FPU can contain sixteen 64-bit registers to hold single-precision or double-precision values. Another sixteen floating-point registers can be used by setting the *FR* bit of the Status register to 1. In addition, a 32-bit Control/Status register is provided, conforming to the IEEE exception processing standard.
- **Load and Store instruction set**. Like the CPU, the FPU uses a load and store-based instruction set. Floating-point operations are starte in a single cycle. Unlike the CPU, some FPU Load and Stor instructions are R-type

1.2.7 Internal Cache

The VR5432 has an instruction cache and a data cache to enhance the efficiency of pipelining. Each cache has a data width of 64 bits and can be accessed in 1 clock. The instruction cache and data cache can be accessed in parallel. Both the instruction cache and data cache have a capacity of 32 KB.

For the details of each cache, refer to Chapter 5.

1.3 Memory Management Unit (MMU)

The VR5432 processor has a usable 32-bit physical addressing range of 4 GB. However, since it is rare for systems to implement a physical memory space this large, the CPU provides a logical expansion of memory space to the programmer by translating addresses into the large virtual address space. The VR5432 processor supports the following two address translation mechanisms:

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

A detailed description of these address spaces is given in Chapter 4.

1.3.1 Translation Lookaside Buffer (TLB)

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

The TLB can hold both instruction and data addresses, and is sometimes referred to as a joint TLB (JTLB). There are also two 4-entry micro-TLBs, one for instructions and one for data. On a miss to either micro-TLB, the pipeline is stalled while the micro-TLB is loaded from the TLB. Loading of the micro-TLBs is handled in the hardware transparently to software.

A virtual address is concatenated with a process identifier and both are sent to the JTLB for translation. If there is no matching entry in the JTLB, an exception occurs and software writes the entry contents to the on-chip JTLB from a page table in memory. The JTLB entry to be rewritten is selected by a value in either the Random or Index register.

1.3.2 Operating Modes

The VR5432 processor has three operating modes:

- User mode
- Supervisor mode
- Kernel mode

The manner in which memory addresses are translated or mapped depends on the operating mode of the CPU; this is described in Chapter 4.

1.4 Instruction Pipeline

The VR5432 incorporates a dual-issue superscalar architecture that allows two integer or floating-point instructions to be issued simultaneously to a five-stage instruction pipeline. For details, refer to Chapter 3.

Signal Descriptions

2

This chapter describes the VR5432 hardware interface signals. The signals include the power inputs and the interfaces for the system, clock, interrupt, initialization, N-Wire/N-Trace hardware debugging, and Joint Test Action Group (JTAG) testing.

The VR5432 supports two interface protocols: VR5432 native mode and R43K (a compatibility mode that emulates the interface used on the VR4300). The mode is selected by the level sampled on the OptionR43K* pin during a cold reset. As described in subsequent sections, some signals are named and behave differently in the two modes.

Figure 2-1 shows the processor signals by function. Signal names used in R43 mode are shown in brackets.

Note: In this manual, active-low signal names are spelled with a trailing asterisk (e.g., the active-low, read-ready signal is RdRdy*).

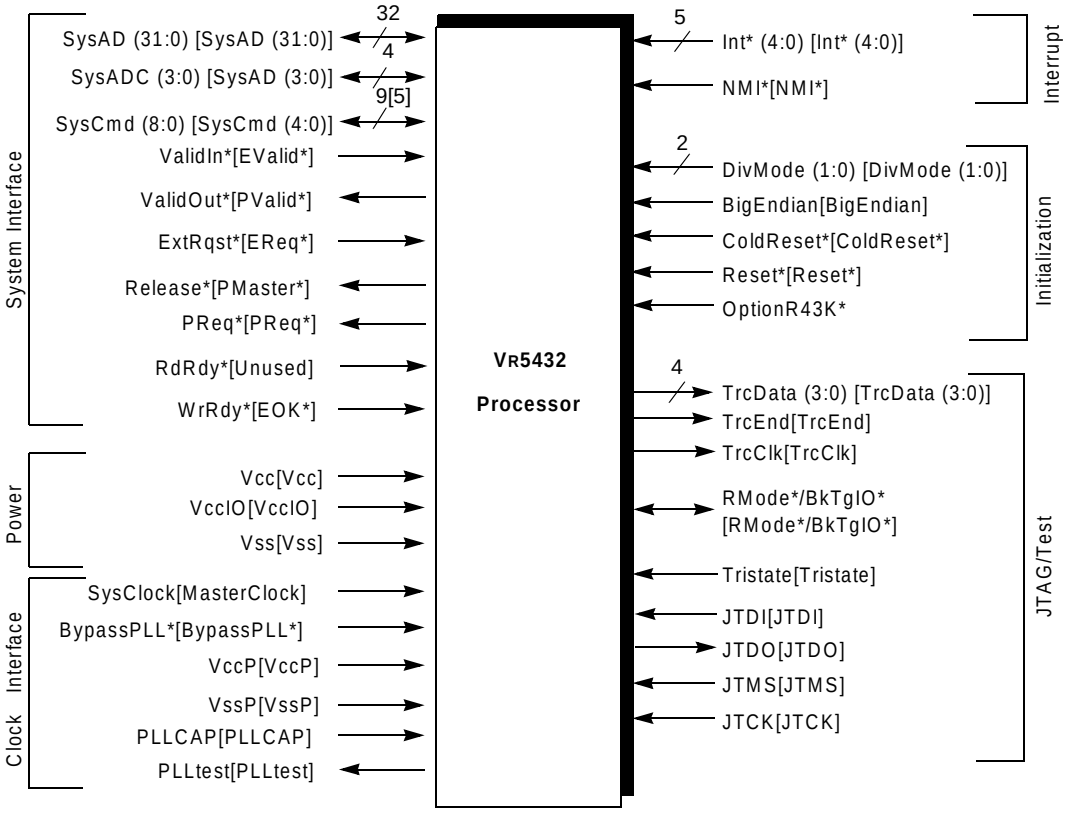


Figure 2-1 VR5432 Processor Signals

2.1 System Interface Signals

System interface signals provide connections between the VR5432 processor and the other components in the system. Table 2-1 lists the system interface signals.

Table 2-1 System Interface Signal

Name	Definition	Direction	Description
ExtRqst*/ [EReq*]	External request (OptionR43K* is high)	Input	An external agent asserts ExtRqst* to request use of the system interface. The processor grants the request by asserting Release*.
	External request (OptionR43K* is low)		An external agent asserts EReq* to request use of the system interface. The processor grants the request by asserting PMaster*.
Release*/ [PMaster*]	Release interface (OptionR43K* is high)	Output	In response to the assertion of ExtRqst*, the processor asserts Release*, signaling to th requesting device that th system interface is available. Release* is also asserted for an uncompelled change to slave state when one or more read requests is outstanding.
	Processor master (OptionR43K* is low)		Indicates the processor is the master of the system interface bus. In response to the assertion o EReq*, the processor deasserts PMaster*, signaling to the requesting devi that th system interface is available. PMaster* is also deasserted for an uncompelled change to slave state when one read request is outstanding.
PReq*/[PReq*]	Processor request (OptionR43K* is high)	Output	Indicates that the processor has another request that is pending. This is used to indicate that the processor would like to send another transaction. It is up to the external agent to grant the request by releasing the system interface with an external null request.
	Processor request (OptionR43K* is low)		Indicates the processor is requesting system interface bus ownership. Also, when the processor experiences a protocol error (the processor detects that an external agent has violated the SysAD bus protocol), the processor continuously toggles PReq*.

Table 2-1 System Interface Signals (continued)

Name	Definition	Direction	Description
RdRdy*/ [Unused]	Read ready (OptionR43K* is high)	Input	The external agent asserts RdRdy* to indicate that it can accept processor read requests.
	Unused (OptionR43K* is low)		The signal is not used when the processor's interface protocol is compatible with the VR4300 interface protocol.
SysAD (31:0)/ [SysAD (31:0)]	System address/data bus (OptionR43K* is high)	Input/ Output	A 32-bit address and data bus fo communication between the processor and an external agent
	System address/data bus (OptionR43K* is low)		A 32-bit address and data bus fo communication between the processor and an external agent
SysADC (3:0)/ [SysADC (3:0)]	System address/ data check bus (OptionR43K* is high)	Input/ Output	A 4-bit bus containing parity for the SysAD bus. SysADC is valid on data cycles only.
	Cache test (OptionR43K* is low)		These pins are for cache test only.
SysCmd (8:0)/ [SysCmd (4:0)]	System command/ data identifier (OptionR43K* is high)	Input/ Output	A 9-bit bus for command and data identifier transmission between the processor and an external agent
	System command/ data identifier (OptionR43K* is low)		A 5-bit bus for command and data identifier transmission between the processor and an external agent. SysCmd (8:5) are unused.
ValidIn*/ [EValid*]	Valid input (OptionR43K* is high)	Input	The external agent asserts ValidIn when it is driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus.
	External valid input (OptionR43K* is low)		The external agent asserts EValid* when it is driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus.

Table 2-1 System Interface Signals (continued)

Name	Definition	Direction	Description
ValidOut*/ [PValid*]	Valid output (OptionR43K* is high)	Output	The processor asserts ValidOut* when it is driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus to the external agent.
	Processor valid output (OptionR43K* is low)		The processor assert PValid* when it is driving a valid address or data on the SysAD bus and a valid command or data identifier on the SysCmd bus to the external agent.
WrRdy*/ [EOK*]	Write ready (OptionR43K* is high)	Input	The external agent asserts WrRdy* when it can accept a processor write request.
	External OK for read/write (OptionR43K* is low)		The external agent asserts EOK* to indicate that it can accept processor read/write requests.

2.2 Power Inputs

The VR5432 requires two power sources. The internal core logic runs off a 2.5 V power supply to optimize internal speed and power. The processor I/O pins use a 3.3 V power supply to provide compatibility with logic technologies commonly used in a system-level design. Table 2-2 lists the two types of power inputs.

Table 2-2 Power Inputs

Name	Definition	Direction	Description
Vss/[Vss]	Vss for processor core and processor I/O	Input	Ground for the internal core logic and processor I/O interface
Vcc/[Vcc]	Vcc for processor core	Input	2.5 V power for the internal core logic
VccIO/[VccIO]	Vcc for processor I/O	Input	3.3 V power for the processor I/O interface

2.3 Clock Interface Signals

The clock interface signals connect the on-chip clock generator to the off-chip clock source. Table 2-3 lists the clock interface signals.

Table 2-3 Clock Interface Signals

Name	Definition	Direction	Description
SysClock/ [MasterClock]	System clock (OptionR43K* is high)	Input	System clock (SysClock) input that establishes the system interface operating frequency and phase during normal operation. In Bypass PLL mode, this input becomes the PClock internally and the system interface operates at half the input frequency.
	System clock (OptionR43K* is low)		System clock [MasterClock] input that establishes the system interface operating frequency and phase during normal operation. In Bypass PLL mode, this input becomes the PClock internally and the system interface operates at half the input frequency.
BypassPLL*/ [BypassPLL*]	Bypass PLL (Independent from OptionR43K*)	Input	Forces SysClock input to bypass the PLL and connect directly to the internal processor clock buffers. The system interface operates at half the SysClock input frequency in this mode.
PLLtest/ [PLLtest]	PLL test (Independent from OptionR43K*)	Output	PLL testing

Table 2-3 Clock Interface Signals (continued)

Name	Definition	Direction	Description
VccP/[VccP]	Quiet Vcc for PLL (Independent from OptionR43K*)	Input	Quiet Vcc for the internal phase-locked loop. This is 2.5 V power. Each internal PLL requires a quiet Vcc.
VssP/[VssP]	Quiet Vss for PLL (Independent from OptionR43K*)	Input	Quiet Vss for the internal phase-locked loop. Each internal PLL requires a quiet Vss.
PLLCAP/ [PLLCAP]	PLL capacitor (Independent from OptionR43K*)	Input	A resistor/capacitor network is connected between PLLCAP and VssP to ensure the proper operation of the phase-locked loop. See Appendix C for details.

2.4 JTAG and Test Interface Signals

The JTAG and test interface signals include IEEE-1149.1 JTAG interface signals as well as signals for specific test features of the VR5432 microprocessor. Table 2-4 lists the JTAG and test interface signals.

Table 2-4 JTAG and Test Interface Signals

Name	Definition	Direction	Description
TrcData (3:0)/ [TrcData(3:0]	Trace data port (Independent from OptionR43K*)	Output	This bus is used to output all trace data codes generated as a result of processor execution.
TrcEnd/ [TrcEnd]	Trace end (Independent from OptionR43K*)	Output	Assertion of this signal indicates the end of a trace data packet from the TrcData port. Trace packets can consist of a single clock cycle of data from the TrcData port, or multiple cycles of data from the TrcData port.
TrcClk/ [TrcClk]	Trace clock (Independent from OptionR43K*)	Output	The trace clock is the same as the system clock. This output is generated for the benefit of test equipment that requires the clock reference for trace information.
RMode*/ BkTgIO*/ [RMode*/ BkTgIO*]	Reset mode Break, Trigger I/O (Independent from OptionR43K*)	Input/ Output	This pin supports the N-Wire Reset mode, as well as break and trigger functions. The pin carries the RMode* signal until ColdReset* is deasserted. It then carries the BkTgIO* signal and serves as a break or trigger, as well as an input or output, depending on the setup in various Debug registers. See Chapter 20 for complete details.
Tristate/ [Tristate]	Tristate all outputs (Independent from OptionR43K*)	Input	This signal tristates all VR5432 outputs to allow a board-level test to isolate the VR5432 processo .
JTDI/[JTDI]	Test data in (Independent from OptionR43K*)	Input	Data is serially scanned in through this signal.

Table 2-4 JTAG and Test Interface Signals (continued)

Name	Definition	Direction	Description
JTCK/[JTCK]	Test clock input (Independent from OptionR43K*)	Input	The processor accepts a serial clock on JTCK. On the rising edge of JTCK, both JTDI and JTMS are sampled. The maximum frequenc of JTCK is 33 MHz, and it runs asynchronously to the SysClock.
JTDO/ [JTDO]	Test data out (Independent from OptionR43K*)	Output	Data is serially scanned out through this signal on the falling edge of JTCK.
JTMS/ [JTMS]	Test mode select (Independent from OptionR43K*)	Input	JTAG Test mode select signal

2.5 Interrupt Interface Signals

The interrupt interface signals make up the interface used by external agents to interrupt the VR5432 processor. Table 2-5 lists the interrupt interface signals.

Table 2-5 Interrupt Interface Signals

Name	Definition	Direction	Description
Int* (4:0)/ [Int* (4:0)]	Interrupt (Independent from OptionR43K*)	Input	General processor interrupts, bitwise ORed with bits 4:0 of the Interrupt register
NMI*/ [NMI*]	Nonmaskable interrupt (Independent from OptionR43K*)	Input	Nonmaskable interrupt, ORed with bit 6 of the Interrupt register

2.6 Initialization Interface Signals

The initialization interface signals make up the interface by which an external agent initializes the processor operating parameters. Table 2-6 lists the initialization interface signals.

Table 2-6 Initialization Interface Signals

Name	Definition	Direction	Description
OptionR43K*	VR4300 mode	Input	When OptionR43K* is driven active-low, the VR5432 operates with the VR4300 protocol.
DivMode (1:0)/ [DivMode (1:0)]	Divide mode (Independent from OptionR43K*)	Input	Sets the PClock-to-SysClock ratio. See Table 2-7 for the ratio that these pins indicate.
BigEndian/ [BigEndian]	Endian mode select (Independent from OptionR43K*)	Input	Sets the VR5432 addressing mode to either Big Endian or Little Endian.
ColdReset*/ [ColdReset*]	Cold reset (Independent from OptionR43K*)	Input	This signal must be asserted for a power-on reset or a cold reset. ColdReset* must be deasserted synchronously with SysClock.
Reset*/ [Reset*]	Reset (Independent from OptionR43K*)	Input	This signal must be asserted for any reset sequence. It can be asserted synchronously or asynchronously for a cold reset, or synchronously to initiate a warm reset. Reset* must be deasserte synchronously with SysClock.

Table 2-7 DivMode (1:0) Settings and Frequencies

DivMode (1:0)	PClock	SysClock	Ratio
11	167 MHz	42 MHz	4:1
10	167 MHz	56 MHz	3:1
01	167 MHz	66 MHz	2.5:1
00	167 MHz	83 MHz	2:1

2.7 Pin Orientation

Figure 2-2 shows the pin orientation for the 208-pin plastic quad flat package (PQFP). Table 2-8 lists the signal names by pin number.

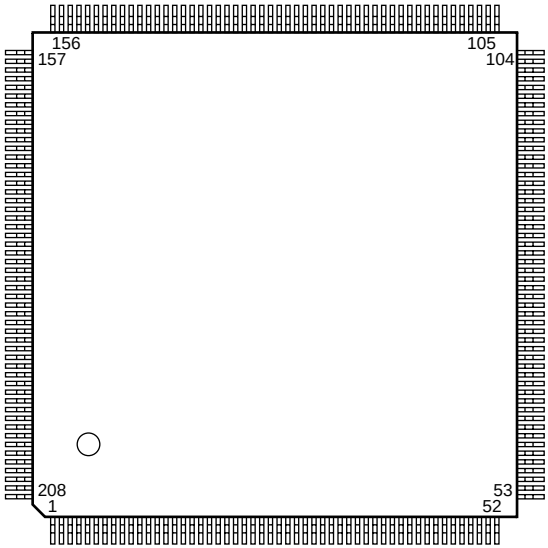


Figure 2-2 208-Pin PQFP Orientation (Top View)

Table 2-8 VR5432 208-Pin PQFP Pin Assignment

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	Vss	44	Int2*	87	DivMode0	130	VccIO	173	ExtRqst*
2	Vss	45	Vcc	88	VccIO	131	Vcc	174	Vss
3	Sys_AD2	46	Int3*	89	BigEndian	132	Sys_AD3	175	Sys_Cmd0
4	VccIO	47	Vss	90	Vss	133	Vss	176	VccIO
5	Sys_AD2	48	Int4*	91	ColdReset*	134	Sys_AD2	177	Sys_Cmd1
6	Vcc	49	Vcc	92	Vcc	135	Vss	178	Vcc
7	Sys_AD1	50	RMode*/ BkTgIO*	93	Tristate	136	Sys_AD1	179	Reset*
8	Vss	51	VccIO	94	VccIO	137	VccIO	180	Vss
9	Vss	52	Vss	95	JTMS	138	Vss	181	VccIO
10	Sys_AD1	53	Vss	96	Vss	139	Vcc	182	OptionR43K*
11	VccIO	54	Vss	97	Vcc	140	Sys_AD0	183	Vcc
12	Sys_AD1	55	VccIO	98	JTCK	141	Vss	184	ValidIn*
13	Vss	56	TrcData0	99	Vss	142	Preq*	185	Vss
14	Sys_AD1	57	Vcc	100	JTDI	143	VccIO	186	Sys_Cmd2
15	Vcc	58	TrcData1	101	VccIO	144	Sys_AD31	187	VccIO
16	Sys_AD1	59	Vss	102	Vcc	145	Vss	188	Sys_Cmd3
17	Vss	60	TrcCLK	103	Vss	146	ValidOut*	189	Vss
18	Sys_AD1	61	Vss	104	Vss	147	Vcc	190	Sys_Cmd4
19	VccIO	62	VccIO	105	Vss	148	Sys_AD30	191	Vcc
20	Vss	63	Vcc	106	Vss	149	VccIO	192	Sys_Cmd5
21	Sys_AD1	64	TrcData2	107	JTDO	150	RdRdy*	193	VccIO
22	Vss	65	Vss	108	Vss	151	Vss	194	Sys_Cmd6
23	Sys_AD1	66	TrcData3	109	Sys_ADC0	152	WrRdy*	195	Vss
24	Vcc	67	Vss	110	VccIO	153	Vcc	196	Sys_Cmd7
25	VccIO	68	VccIO	111	Sys_ADC1	154	Sys_AD29	197	Vss
26	Vss	69	TrcEnd	112	Vcc	155	Vss	198	Sys_Cmd8
27	Sys_AD1	70	VccIO	113	Sys_ADC2	156	VccIO	199	VccIO
28	Vss	71	Vss	114	Vss	157	Vss	200	Vcc
29	Sys_AD1	72	VccIO	115	Sys_ADC3	158	Vss	201	Sys_AD24
30	Vss	73	Vss	116	VccIO	159	Vcc	202	Vss
31	VccIO	74	PLLtest	117	Vss	160	Sys_AD28	203	Sys_AD23
32	Vcc	75	Vcc	118	Vcc	161	Vss	204	Vss
33	Sys_AD	76	SysClock	119	Vss	162	VccIO	205	Vcc
34	Vss	77	Vss	120	Sys_AD7	163	Vss	206	Sys_AD22
35	Sys_AD	78	VccP	121	VccIO	164	Sys_AD27	207	VccIO
36	VccIO	79	PLLCAP	122	Sys_AD6	165	Vcc	208	Vss
37	Vss	80	VssP	123	Vss	166	Sys_AD26		
38	NMI*	81	VccIO	124	Sys_AD5	167	Vss		
39	Vcc	82	Vss	125	Vcc	168	Release*		
40	Int0*	83	BypassPLL*	126	VccIO	169	VccIO		
41	VccIO	84	Vcc	127	Vss	170	Sys_AD25		
42	Int1*	85	DivMode1	128	Sys_AD4	171	Vcc		
43	Vss	86	Vss	129	Vss	172	Vss		

Pipeline

3

3.1 Pipeline Stages

The VR5432 instruction pipeline follows these five stages:

- IC: Instruction Fetch
- RF: Register Fetch
- EX: Executi
- DC: Data Cache Fetch
- WB: Write-back

Each stage takes one PCycle (one cycle of PClock, which runs at a multiple of the frequency of SysClock). Therefore, the execution of each instruction takes at least five PCycles (see Figure 3-1). An instruction can take longer—for example, if the required data is not in the cache, the data must be retrieved from main memory.

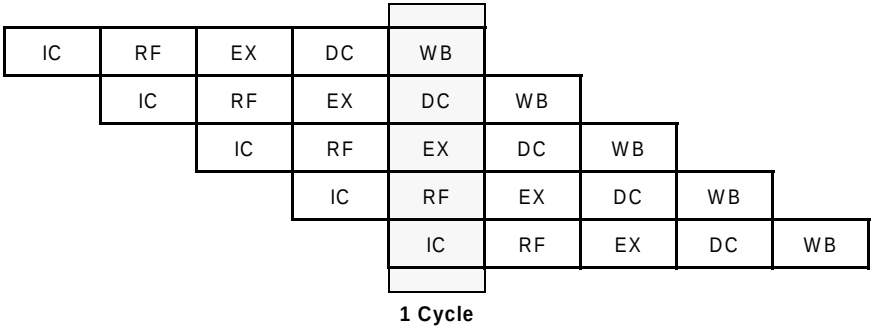


Figure 3-1 Instruction Pipeline Stages

Once the dual-issue pipelines have been filled, ten instructions can be executed simultaneously. Figure 3-2 shows a simplified diagram of the dual-issue mechanism. The blocks listed below are part of the execution unit.

The **Load/Store unit** handles load/store operations that load data from memory to the general-purpose register or store data from the general-purpose register to memory.

The **dual data path units (DP0 and DP1)** have the hardware resources to execute integer and IEEE-754 single- and double-precision floating-point operations. The symmetric, two-issue superscalar design allows two integer or floating-point instructions to be issued simultaneously.

The **vector unit** handles parallel operations on packed vectors of unsigned 8-bit integers for rapid processing of data for multimedia, such as sound and graphics.

The **MAC unit** handles DSP-like multiply-accumulate operations on signed or unsigned integers.

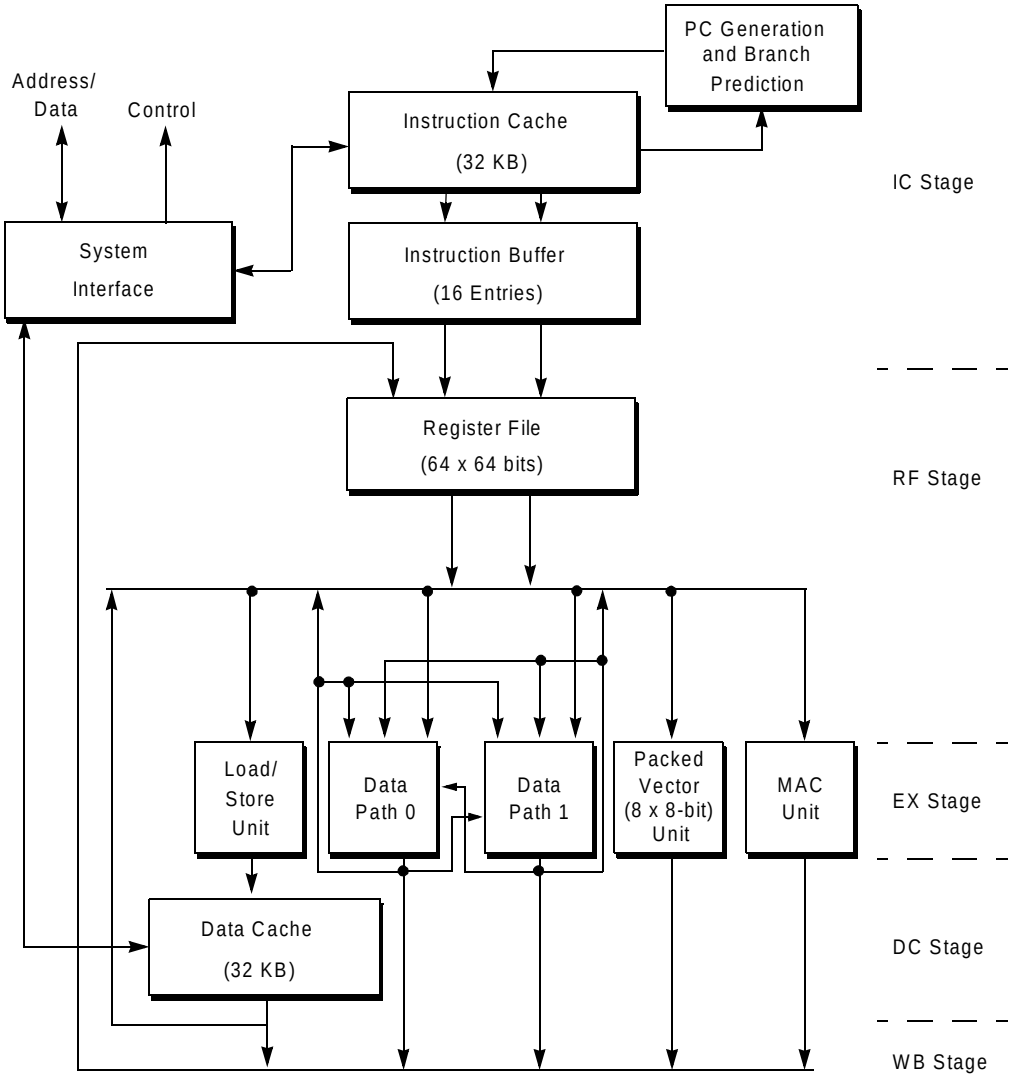


Figure 3-2 Dual-Issue Mechanism

3.2 Branch Delay

The CPU pipeline has a delay of three cycles for mispredicted branches. The two-cycle branch delay is a result of the branch comparison logic operating during the EX pipeline phase of the Branch instruction. Figure 3-3 illustrates the branch delay condition. Whenever the processor correctly predicts a branch, there is a 0-cycle (0 instruction) branch delay.

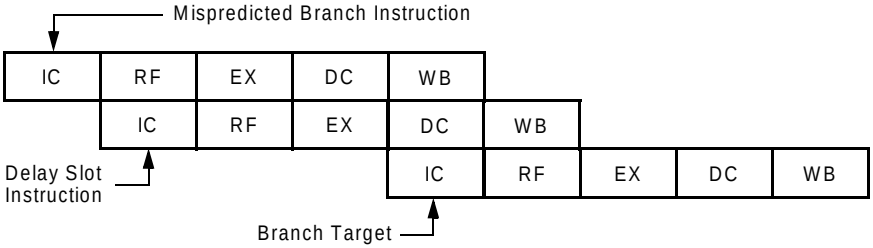


Figure 3-3 CPU Pipeline Branch Delay

3.3 Load Delay

The CPU pipeline has a load delay of one cycle. The one-cycle load delay is a result of the load completing in DC, while the instruction requesting the use of the load data needs it in RF. Note that because the processor has a two-way superscalar architecture, two to three instructions following the load cannot use the load data. The processor will interlock if the instructions in the load delay slots are dependent on the load data itself. This penalty could and should be minimized with appropriate scheduling by the compiler. Figure 3-4 shows the load delay of one pipeline stage.

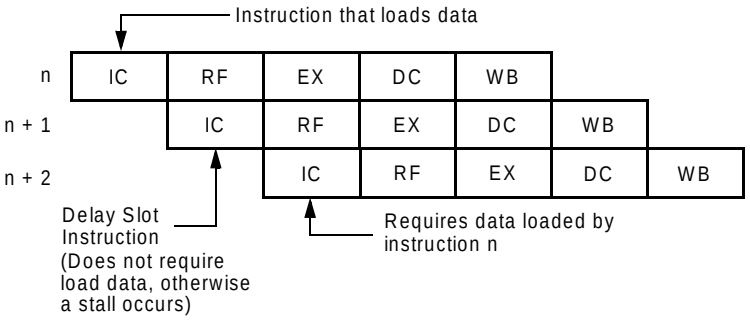


Figure 3-4 CPU Pipeline Load Delay

The nominal miss penalty for loads is (9.5 + M) processor cycles in 2:1 mode, where M is the external memory latency excluding the address and data transfer time on the system bus. Loads are nonblocking, with hits under miss allowed. Store misses incur an additional one-cycle penalty.

3.4 Interlock and Exception Handling

The VR5432 has a basic dual five-stage pipe that allows for a good overlap of instruction execution. When a data dependency or resource conflict is encountered, the smooth pipeline flow is interrupted. Interruptions handled using hardware, such as cache misses, are referred to as *interlocks*, while those that are handled using software are called *exceptions*. During each cycle, exception and interlock conditions are checked for all active instructions.

Because each exception or interlock condition corresponds to a particular pipeline stage, a condition can be traced back to the particular instruction in the exception/interlock stage. For instance, a Reserved Instruction (RI) exception is raised in the execution (EX) stage. Table 3-1 summarizes all stall and exception conditions. Table 3-2 describes the stall and exception conditions.

Table 3-1 Pipeline Stalls and Exceptions

Stall	Description	Exception	Description
ICM	I cache miss	IADE	Instruction address error
ITM	Target branch address I cache miss	IBE	Instruction bus error
BT	Branch take	IPErr	Instruction parity error
CP0BI	CP0 bypass interlock	BP	Breakpoint instruction
DPI	Data path bypass interlock	CpU	Coprocessor unusabl
LDI	Load data interlock	DADE	Data address error
MBI	Multiply bypass interlock	OVF	Arithmetic overflow
MCI	Multicycle cycle interlock	RI	Reserved instruction
COp	Cache operation	SysCall	System call instruction
DCM	Data cache miss	Trap	Trap instruction
ITLB	Instruction TLB miss	CPE	Coprocessor error
DTLB	Data TLB miss	DBE	Data bus error
		DPErr	Data parity error
		Mod	TLB modification
		Int	Interrupt
		NMI	Nonmaskable interrupt
		Reset	Reset
		Watch	Reference to watch addr.
		ITLBEx	Instruction TLB exception
		DTLBEx	Data TLB exception

Table 3-2 Relationship of Pipeline Stage to Interlock and Exception Conditions

State	Pipeline Stage				
	IC	**RF**	**EX**	**DC**	**WB**
Stall	ICM	BT	MCI	COp	
	ITM	CP0BI		DCM	
	ITLB	DPI		DTLB	
		LDI			
		MBI			
Exceptions	IADE	IBE	BP	CPE	
	ITLBEx	IPErr	CpU	DBE	
			DADE	DPErr	
			OVF	Mod	
			RI	Int	
			SysCall	NMI	
			Trap	Reset	
				Watch	
				DTLBEx	

3.4.1 Exception Conditions

When an exception condition occurs, the processor aborts the instruction that caused the exception and all those that follow it in the pipeline are canceled. 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 canceled instruction. When an exception-generating instruction reaches the WB stage, three events occur:

- Writes are made to various CP0 registers with the exception state and cause.
- The exception vector program counter address is calculated.
- The exception bits of earlier pipeline stages are cleared

This implementation allows all instructions that occurred before the exception to complete and all instructions that occurred after the exception to be aborted. The value of the EPC (or ErrorEPC, if the *ERL* bit in the CP0 Status register is set) is such that execution can be restarted at the point of exception. In addition, all exceptions are guaranteed to be taken in order. Figure 3-5 illustrates the exception detection mechanism for a Reserved Instruction (RI) exception.

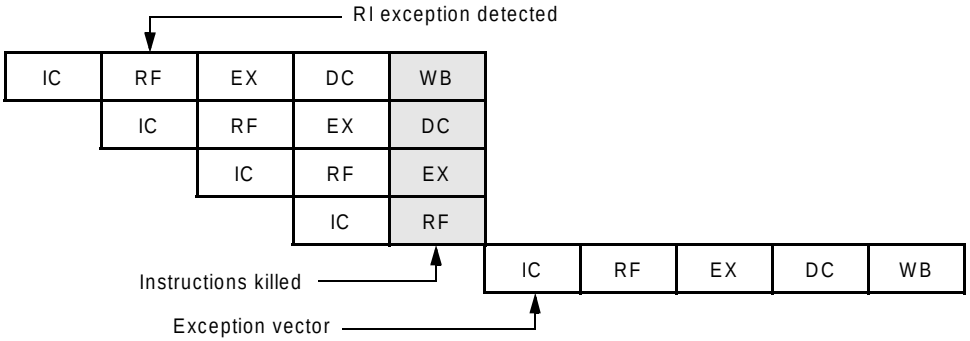


Figure 3-5 Exception Detection Mechanism

3.4.2 Interrupt Latency

External interrupts are processed in the WB stage of the pipeline. When an enabled interrupt is detected, an Interrupt exception is taken, assuming there are no higher priority exceptions. In some cases, the interrupt will not be processed until the transaction buffer is empty. The nominal delay from when an enabled external interrupt is received at the processor pins to when the processor fetches the first instruction of the Interrupt Exception vector is 6 + N cycles, where N is the number of cycles needed to empty the transaction buffer. N depends on the types of transactions in the buffer and the system response time to perform those transactions.

3.4.3 Stall Conditions

A stall condition is used to suspend the pipeline for conditions that require additional cycles to complete, such as a bypass data interlock. When the interlock condition is detected, the processor will stall the issue of the dependent instruction in the RF stage. During the time the instruction issue is stalled, the processor continues to execute instructions already issued. Figure 3-6 shows a data interlock stall.

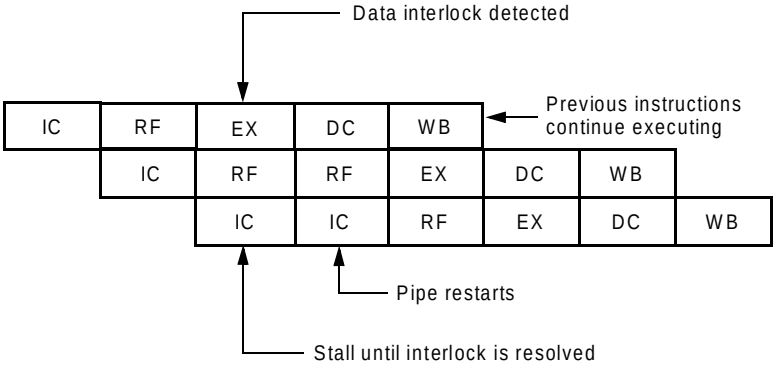


Figure 3-6 Servicing a Data Interlock Stall

3.5 Transaction Buffer

The processor contains a transaction buffer that improves the performance of operations to external memory. All system interface operations use the transaction buffer. It is organized as a four-deep FIFO and can hold up to 256 bits of data. This transaction buffer can be used to hold up to four read or uncached write operations or one cache line write-back.

On a cache miss requiring a write-back, the entire cache line (write-back data) is dumped into the transaction buffer, which allows the processor to proceed in parallel with the memory update. For uncached and write-through stores, the transaction buffer decouples the CPU from the write to memory. If the transaction buffer is full, additional stores are stalled until there is room for them in the transaction buffer. This transaction buffer mechanism is completely invisible to software.

Memory Management Unit

4

The VR5432 processor provides a full-featured memory management unit (MMU) that uses an on-chip translation lookaside buffer (TLB) to translate virtual addresses into physical addresses.

This chapter describes the processor's virtual and physical address spaces, the virtual-to-physical address translation, the TLB translation process, and the system control coprocessor (CP0) registers that provide the software interface to the TLB.

4.1 Translation Lookaside Buffer

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

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

4.1.1 Hits and Misses

If there is a virtual address match or hit in the TLB, the physical page number is extracted from the TLB and concatenated with the offset to form the physical address (see Figure 4-1).

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

4.2 Processor Modes

The VR5432 has three processor operating modes, an instruction set mode, and an addressing mode. All are described in this section.

† There are virtual-to-physical address translations that occur outside of the TLB. For example, addresses in the *kseg0* and *kseg1* spaces are unmapped translations.

4.2.1 Processor Operating Modes

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

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

User mode is used for application programs. Kernel mode is used for operating system functions, such as exception handlers. Supervisor mode is an intermediate privilege level between the operating system and the application, which can be used for operating systems that implement both a kernel layer and a layer for operating system extensions, such as device drivers.

The processor's operating mode is controlled by the Status register's *KSU* field and the *EXL* and *ERL* bits. The *EXL* and *ERL* bits are set during the service of an exception or error, respectively. Table 4-1 relates the operating mode to the states of *KSU*, *EXL*, and *ERL*.

Table 4-1 Processor Operating Modes

KSU (1:0)	EXL	ERL	Mode
10	0	0	User mode
01	0	0	Supervisor mode
00	0	0	Kernel mode
—	—	1	
—	1	—	

On an exception or error, the *EXL* and *ERL* bits are set independently of the *KSU* bits. Interrupts are disabled whenever either of these bits is set. If the exception handler clears *EXL*, for example to allow handling nested interrupts, the operating mode will revert from Kernel mode to that specified by the current value of the *KSU* bits. Therefore, it may be necessary for the exception handler to change the value of the *KSU* bits before clearing *EXL*.

4.2.2 Instruction Set Mode

The processor's instruction set mode determines which instruction set is enabled. By default, the processor implements the MIPS IV instruction set architecture (ISA). For compatibility with earlier machines, however, it can be limited to the MIPS III ISA or the MIPS I/II ISAs. The current instruction set mode is controlled by the processor operating mode and the *UX*, *SX*, and *CU3* bits of the Status register, as shown in Table 4-2. (Dashes indicate "Don't care".)

Table 4-2 Instruction Set Mode

Operating Mode	UX	SX	CU3	MIPS I, II	MIPS III	MIPS IV
User mode	0	—	0	Yes	No	No
	0	—	1	Yes	No	Yes
	1	—	0	Yes	Yes	No
	1	—	1	Yes	Yes	Yes
Supervisor mode	—	0	—	Yes	No	Yes
	—	1	—	Yes	Yes	Yes
Kernel mode	—	—	—	Yes	Yes	Yes

4.2.3 Addressing Modes

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

Refer to Table 4-3 for the following addressing mode encodings:

- In **Kernel mode**, the *KX* bit enables 64-bit addressing; all instructions are always valid.
- In **Supervisor mode,** the *SX* bit enables 64-bit addressing and th MIPS III instructions
- In **User mode**, the *UX* bit enables 64-bit addressing and the MIPS II instructions; the *CU3* bit enables the new MIPS IV instruction set extensions.

Table 4-3 Addressing Modes

Operating Mode	UX	SX	CU3	32/64-bit Addressing
User mode	0	—	—	32
	1	—	—	64
Supervisor mode	—	0	—	32
	—	1	—	64
Kernel mode	—	—	0	32
	—	—	1	64

4.3 Addresses and Address Spaces

This section describes virtual and physical addresses, the manner in which virtual addresses are translated into physical addresses by the TLB, and the three address spaces: User, Supervisor, and Kernel.

4.3.1 Virtual Addresses

The processor has three types of address space: User, Supervisor, and Kernel. Each space can be independently configured to be a 32-bit or 64-bit space by the *KX*, *SX*, and *UX* bits in the Status register.

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

Figure 4-1 shows the translation of a virtual address into a physical address.

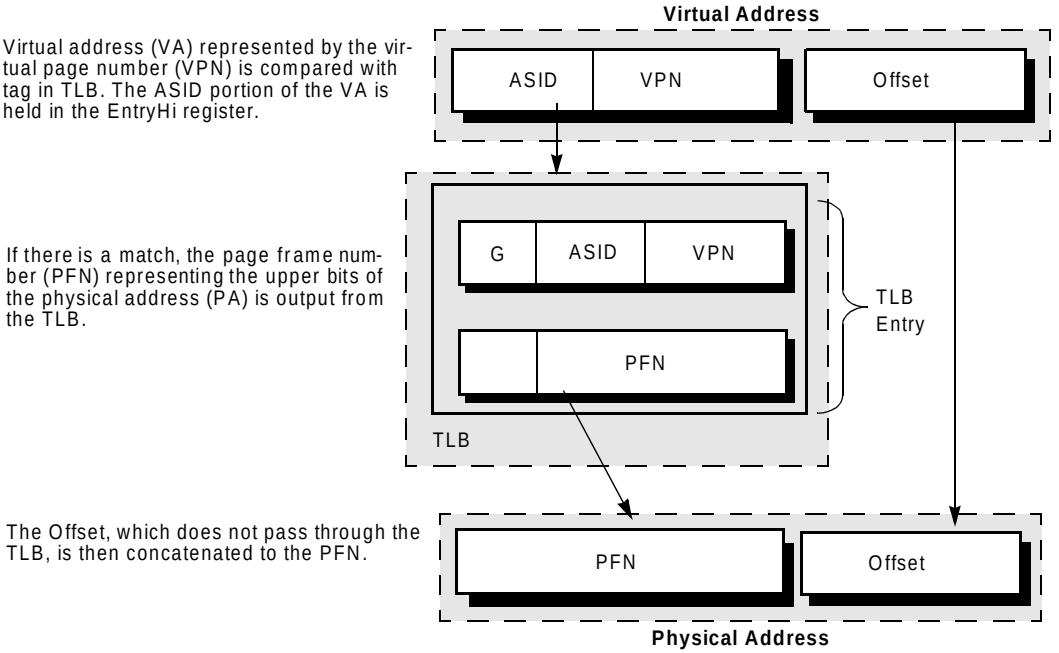


Figure 4-1 Virtual-to-Physical Address Translation

As shown in Figure 4-1, the virtual address is extended with an 8-bit address space identifier (*ASID* bit), which reduces the frequency of TLB flushing when switching contexts. This 8-bit ASID is in the CP0 EntryHi register. The *Global* bit (*G*) is in each TLB entry.

4.3.2 Physical Addresses

Using a 32-bit address, the processor physical address space encompasses 4 GB.

Caution: Although the VR5400 processor family MMU is designed to handle up to 36 bits of physical address space, the VR5432 physical address is limited to 32 bits, due to its System Multiplex Address/Data (SysAD) bus size. A 32-bit physical address space provides 4 GB of physical addressing. Any attempt to reference a physical address with bits 35:32 not set to 0 will force an Address Error exception.

4.3.3 Virtual-to-Physical Address Translation

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

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

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

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

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

4.3.4 32-Bit Mode Virtual Address Translation

Figure 4-2 shows the virtual-to-physical address translation of a 32-bit mode address.

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

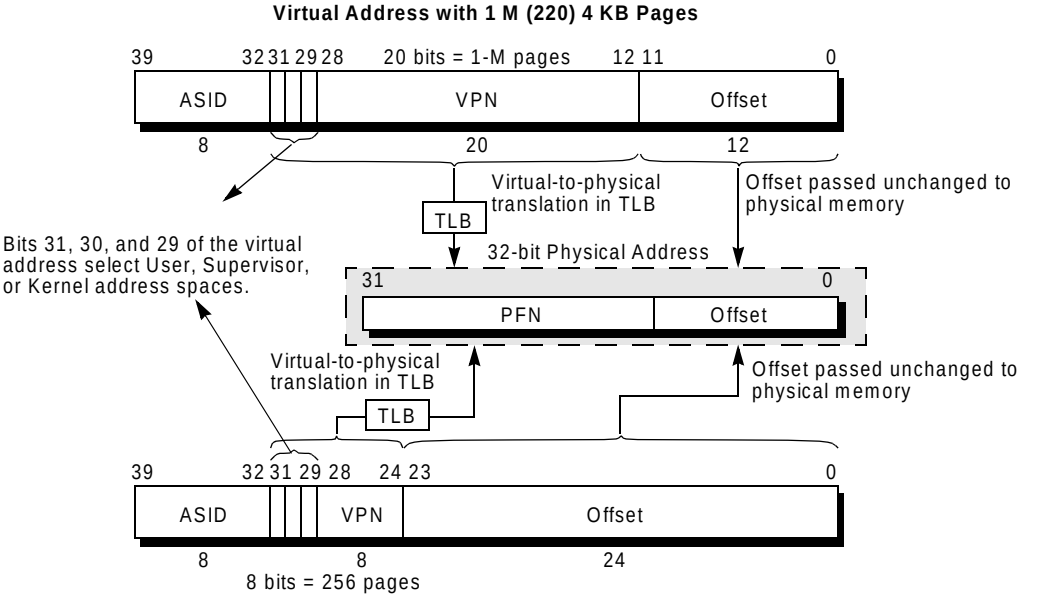


Figure 4-2 32-Bit Mode Virtual Address Translation

Note: In setting up the TLB entry for 32- or 64-bit mode virtual address translation, set the upper four bits of the page frame number to 0, to guarantee that the translated physical address will have bits 35:32 set to 0. (See Figure 4-3 and Figure 4-4.)

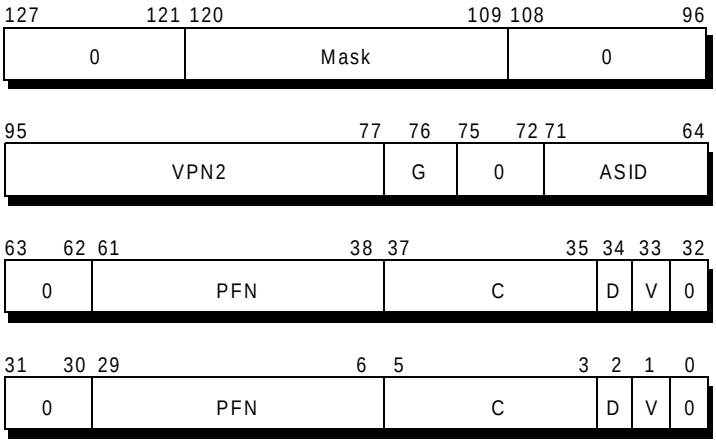


Figure 4-3 TLB Entry Format in 32-Bit Mode

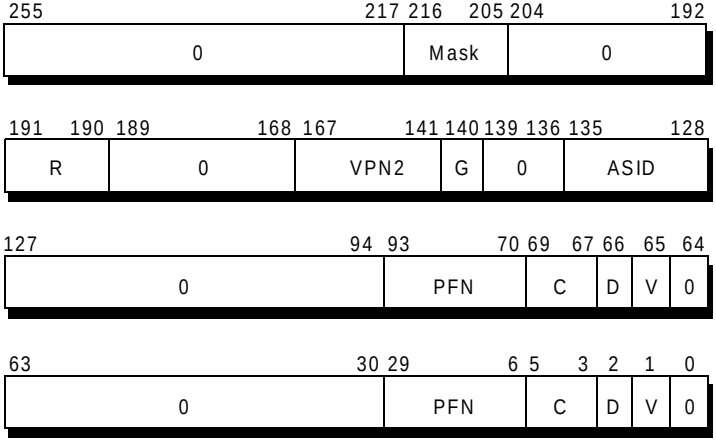


Figure 4-4 TLB Entry Format in 64-Bit Mode

4.3.5 64-Bit Mode Virtual Address Translation

Figure 4-5 shows the virtual-to-physical address translation. This figure illustrates the two extremes in the range of possible page sizes: a 4 KB page (12 bits) and a 16 MB page (24 bits).

- The top portion of Figure 4-5 shows a virtual address with a 12-bit or 4 KB page size, labeled Offset. The remaining 28 bits of the address represent the VPN, and index the 256 M-entry page table.
- The bottom portion of Figure 4-5 shows a virtual address with a 24-bit or 16 MB page size, labele Offset. The remaining 16 bits of th address represent the VPN, and index the 64 K-entry page table

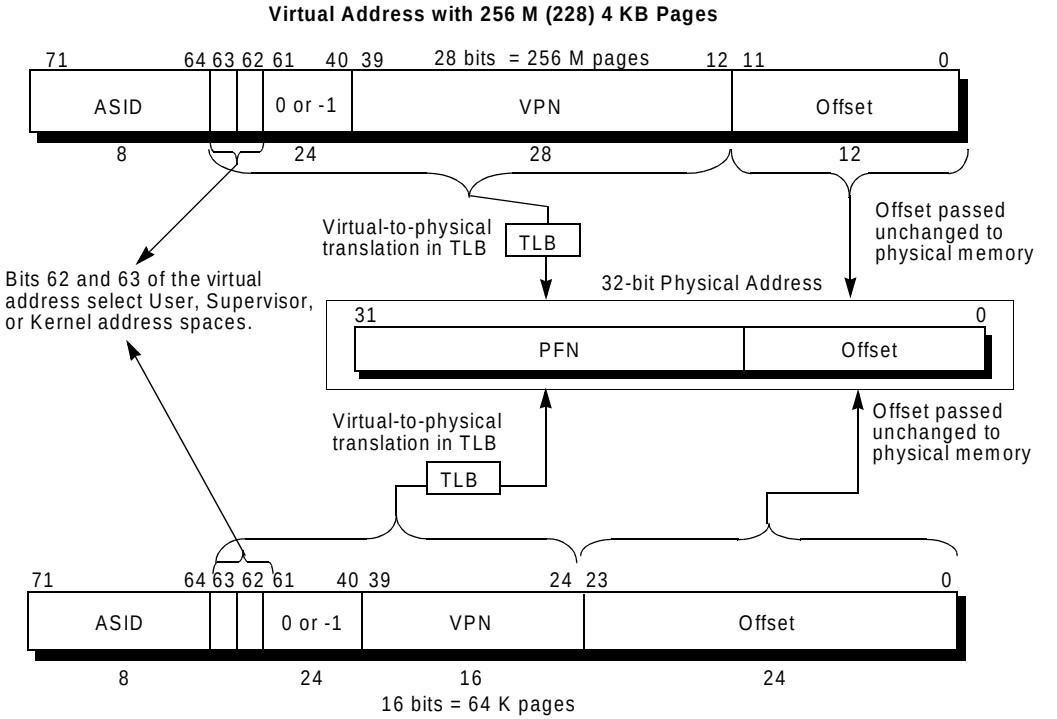


Figure 4-5 64-Bit Mode Virtual Address Translation

Note: In setting up the TLB entry for 32- or 64-bit mode virtual address translation, the upper four bits of the physical frame number must be set to 0, to guarantee that the translated page address will have bits 35:32 set to 0. (See Figure 4-3 and Figure 4-4.)

4.3.6 User Address Space

In User address space, a single, uniform virtual address space labeled user segment (*useg*) is available. Its size is:

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

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

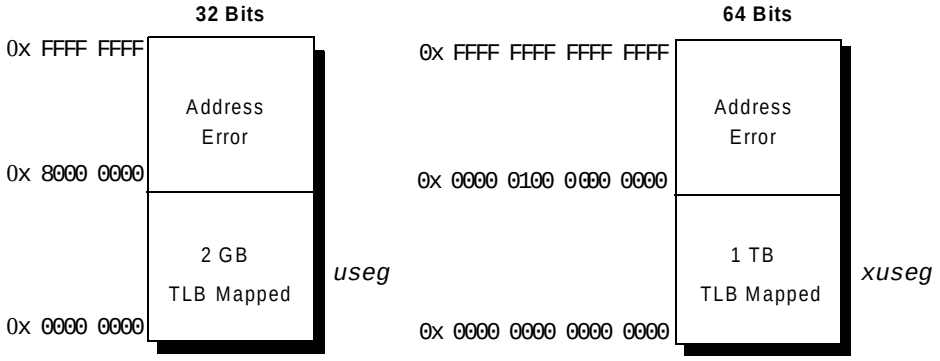


Figure 4-6 Virtual Address Space in User Mode

User space can be accessed from the User, Supervisor, and Kernel modes.

The user segment starts at address 0 and the current active user process resides in either *useg* (32-bit mode) or *xuseg* (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 all of the following values:

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

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

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

Table 4-4 lists the characteristics of the two user address spaces, *useg* and *xuseg*.

Table 4-4 32-Bit and 64-Bit User Address Space Segment

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

4.3.6.1 32-bit user space (*useg*)

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

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

4.3.6.2 64-bit user space (*xuseg*)

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

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

4.3.7 Supervisor Space

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

The processor operates in Supervisor mode when the Status register contains all of the following values:

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

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

- When *SX* = 0, 32-bit supervisor space is selected and TLB misses o supervisor space, addresses are handled by the 32-bit TLB Refill exception handler.
- When *SX* = 1, 64-bit supervisor space is selected and TLB misses o Supervisor space, addresses are handled by the 64-bit XTLB Refill exception handler. Figure 4-7 shows supervisor address mapping Table 4-5 lists the characteristics of the supervisor space segments; descriptions of the address spaces follow.

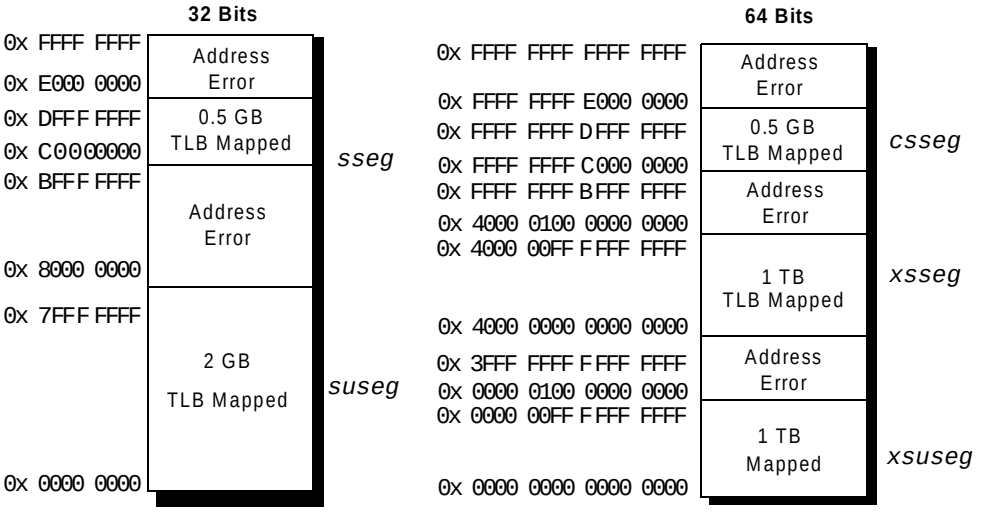


Figure 4-7 User and Supervisor Address Spaces in Supervisor Mode

Table 4-5 Supervisor Mode Addressi n

Address Bit Values	SX	Segment Name	Address Range	Segment Size
32 bits A (31) = 0	0	*suseg*	0x 0000 0000 through 0x 7FFF FFFF	2 GB (2^{31} bytes)
32 bits A (31:29) = 110_2	0	*sseg*	0x C000 0000 through 0x DFFF FFFF	512 MB (2^{29} bytes)
64 bits A (63:62) = 00_2	1	*xsuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 terabyte (2^{40} bytes)
64 bits A (63:62) = 01_2	1	*xsseg*	0x4000 0000 0000 0000 through 0x4000 00FF FFFF FFFF	1 terabyte (2^{40} bytes)
64 bits A (63:62) = 11_2	1	*csseg*	0xFFFF FFFF C000 0000 through 0xFFFF FFFF DFFF FFFF	512 MB (2^{29} bytes)

4.3.7.1 32-bit supervisor, user space (*suseg*)

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

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

4.3.7.2 32-bit supervisor, supervisor space (*sseg*)

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

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

4.3.7.3 64-bit supervisor, user space (*xsuseg*)

In Supervisor mode, when *SX* = 1 in the Status register and bits 63:62 of the virtual address are set to 00_2, the *xsuseg* virtual address space is selected; it covers the full 2^{40} bytes (1 terabyte) 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.

4.3.7.4 64-bit supervisor, current supervisor space (*xsseg*)

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

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

4.3.7.5 64-bit supervisor, separate supervisor space (*csseg*)

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

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

4.3.8 Kernel Space

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

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

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

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

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

Kernel virtual address space is divided into regions differentiated by the high-order bits of the virtual address, as shown in Figure 4-8. Table 4-6 and Table 4-7 list the characteristics of the Kernel mode segments.

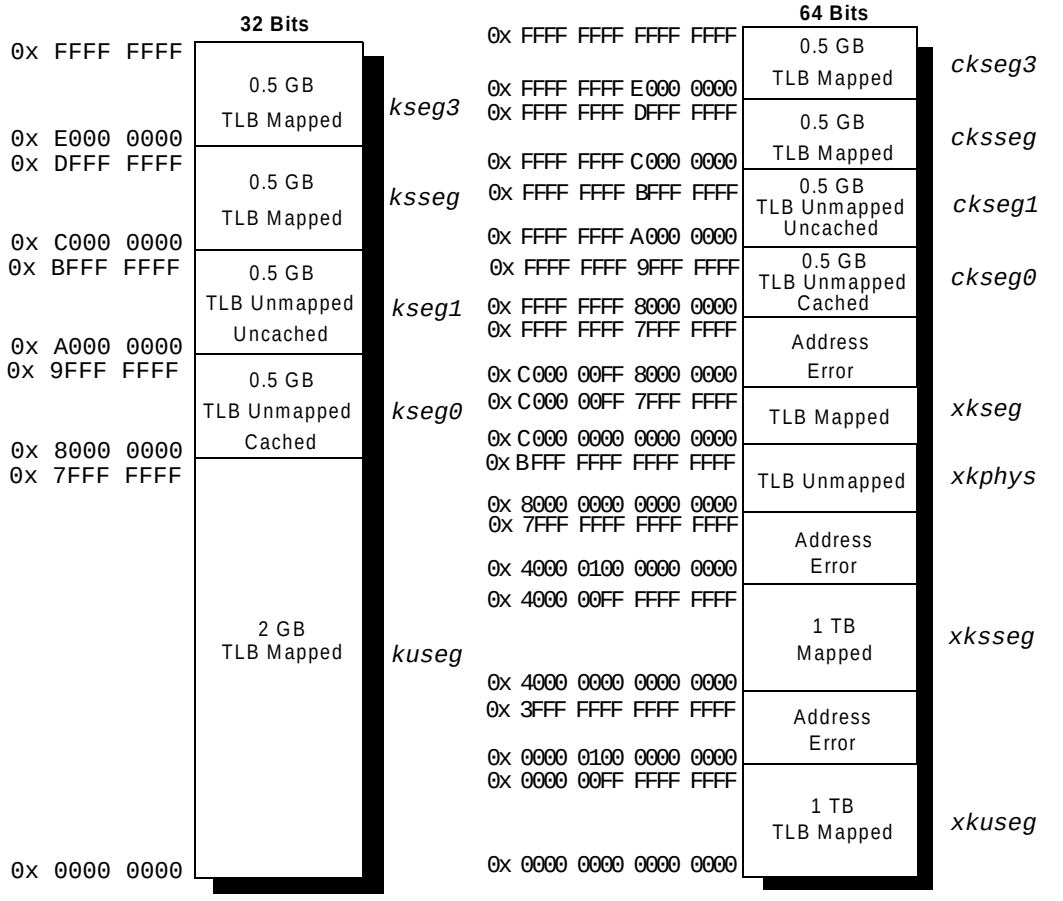


Figure 4-8 User, Supervisor, and Kernel Address Spaces in Kernel Mode

Table 4-6 32-Bit Kernel Mode Addressing

Address Bit Values	KX	Segment Name	Address Range	Segment Size
32 bits A (31) = 0	0	*kuseg*	0x 0000 0000 through 0x 7FFF FFFF	2 GB (2^{31} bytes)
32 bits A (31:29) = 100_2	0	*kseg0*	0x 8000 0000 through 0x 9FFF FFFF	512 MB (2^{29} bytes)
32 bits A (31:29) = 101_2	0	*kseg1*	0x A000 0000 through 0x BFFF FFFF	512 MB (2^{29} bytes)
32 bits A (31:29) = 110_2	0	*ksseg*	0x C000 0000 through 0x DFFF FFFF	512 MB (2^{29} bytes)
32 bits A (31:29) = 111_2	0	*kseg3*	0x E000 0000 through 0x FFFF FFFF	512 MB (2^{29} bytes)

Table 4-7 64-Bit Kernel Mode Addressing

Address Bit Values	KX	Segment Name	Address Range	Segment Size
64 bits A (63:62) = 00_2	1	*xkuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 terabyte (2^{40} bytes)
64 bits A (63:62) = 01_2	1	*xksseg*	0x4000 0000 0000 0000 through 0x4000 00FF FFFF FFFF	1 terabyte (2^{40} bytes)
64 bits A (63:62) = 10_2	1	*xkphys*	0x8000 0000 0000 0000 through 0xBFFF FFFF FFFF FFFF	8 2^{32} byte spaces
64 bits A (63:62) = 11_2	1	*xkseg*	0xC000 0000 0000 0000 through 0xC000 00FF 7FFF FFFF	($2^{40} - 2^{31}$) bytes
64 bits A (63:62) = 11_2 A (61:31) = -1	1	*ckseg0*	0xFFFF FFFF 8000 0000 through 0xFFFF FFFF 9FFF FFFF	512 MB (2^{29} bytes)
64 bits A (63:62) = 11_2 A (61:31) = -1	1	*ckseg1*	0xFFFF FFFF A000 0000 through 0xFFFF FFFF BFFF FFFF	512 MB (2^{29} bytes)
64 bits A (63:62) = 11_2 A (61:31) = -1	1	*ckseg2*	0xFFFF FFFF C000 0000 through 0xFFFF FFFF DFFF FFFF	512 MB (2^{29} bytes)
64 bits A (63:62) = 11_2 A (61:31) = -1	1	*ckseg3*	0xFFFF FFFF E000 0000 through 0xFFFF FFFF FFFF FFFF	512 MB (2^{29} bytes)

4.3.8.1 32-bit kernel, user space (*kuseg*)

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

4.3.8.2 32-bit kernel, kernel space 0 (*kseg0*)

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

4.3.8.3 32-bit kernel, kernel space 1 (*kseg1*)

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

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

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

4.3.8.4 32-bit kernel, supervisor space (*ksseg*)

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

4.3.8.5 32-bit kernel, kernel space 3 (*kseg3*)

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

4.3.8.6 64-bit kernel, user space (*xkuseg*)

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

When *ERL* = 1 in the Status register, the user address region becomes a 2^{31}-byte unmapped (interpreted as a physical address without translation), uncached address space.

4.3.8.7 64-bit kernel, current supervisor space (*xksseg*)

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

4.3.8.8 64-bit kernel, physical spaces (*xkphys*)

In Kernel mode, when *KX* = 1 in the Status register and bits 63:62 of the 64-bit virtual address are 10_2, the *xkphys* virtual address space is selected; it is a set of eight 2^{32}-byte kernel physical spaces. Accesses with address bits 58:32 not equal to 0 cause an Address Error exception.

References to this space are not mapped; the physical address selected is taken from virtual address bits 35:0 internally, 31:0 externally. Bits 61:59 of the virtual address specify the cache attributes, as shown in Table 4-8.

Table 4-8 Cacheability and Coherency Attributes

Value (61:59)	Cacheability Attributes	Starting Address
0	Reserved	0x8000 0000 0000 0000
1	Cacheable, write-through	0x880 00000 0000 0000
2	Uncached	0x900 00000 0000 0000
3	Cacheable, write-back	0x9800 0000 0000 0000
4–6	Reserved	0xA000 0000 0000 0000
7	Uncached, accelerated	0xB800 0000 0000 0000

4.3.8.9 64-bit kernel, kernel space (*xkseg*)

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

- *xkseg*, the current kernel virtual space (the virtual address is extended with the contents of the 8-bit *ASID* field to form a unique virtual address)
- One of the four 32-bit kernel compatibility spaces, as described in the next sectio

4.3.8.10 64-bit kernel, compatibility spaces

In Kernel mode, when *KX* = 1 in the Status register, bits 63:62 of the 64-bit virtual address are 11_2, and bits 61:31 of the virtual address equal –1, the lower two bytes of the address, as shown in Figure 4-8, select one of the following 512 MB compatibility spaces.

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

4.4 System Control Coprocessor

The system control coprocessor (CP0) is implemented as an integral part of the CPU and supports memory management, address translation, exception handling, and other privileged operations. CP0 contains the registers shown in Figure 4-9, plus a 48-entry TLB. The sections that follow describe how the processor uses the memory management-related registers.

Figure 4-9 lists each CP0 register and its identification number. (For instance, the PageMask register is register number 5.)

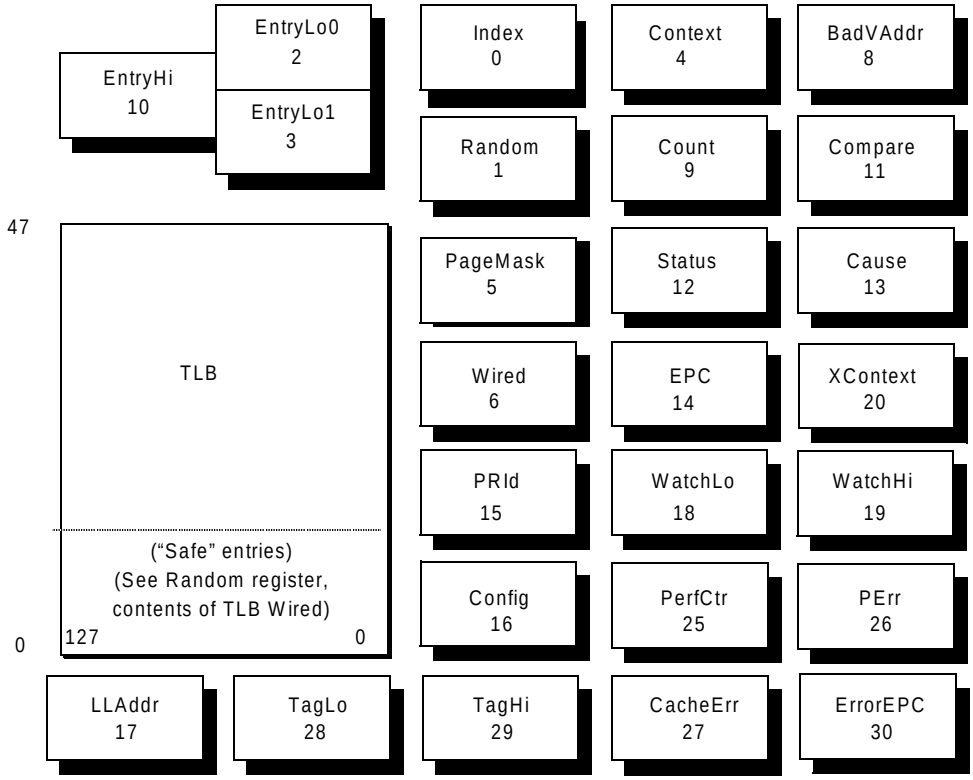


Figure 4-9 CP0 Registers and the TLB

4.4.1 TLB Entry Format

Figure 4-10 shows the TLB entry formats for both 32- and 64-bit modes. Each field of an entry has a corresponding field in the EntryHi, EntryLo0, EntryLo1, or PageMask registers.

Caution: Although the VR5432 processor MMU is designed to handle up to 36 bits of physical address space, the VR5432 physical address is limited to 32 bits, due to its System Multiplex Address/Data (SysAD) bus size. A 32-bit physical address space provides 4 GB of physical addressing. Any attempt to reference a physical address with bits 35:32 not set to 0 will force an Address Error exception; therefore, bits 35:32 should be set to zero.

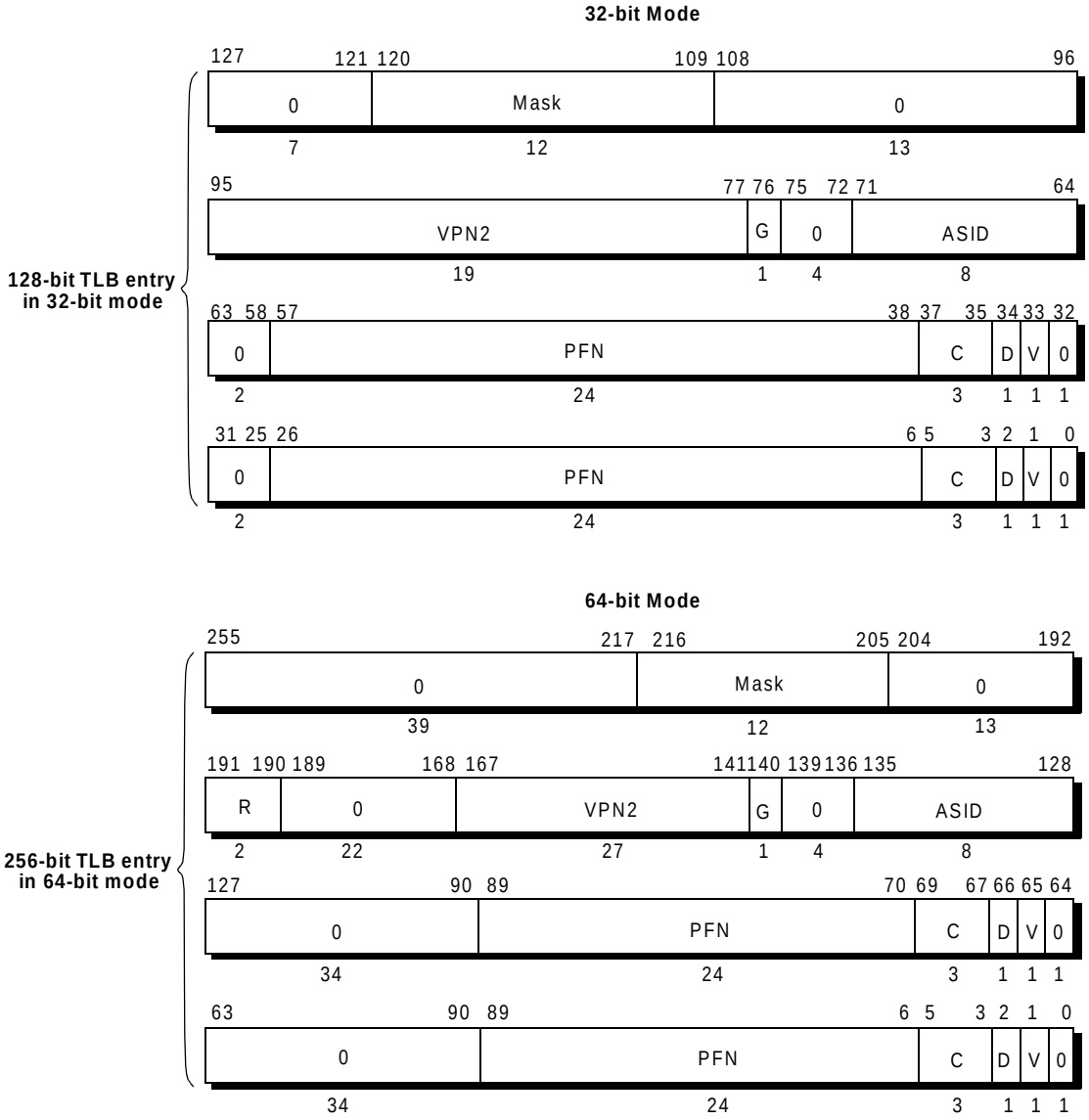
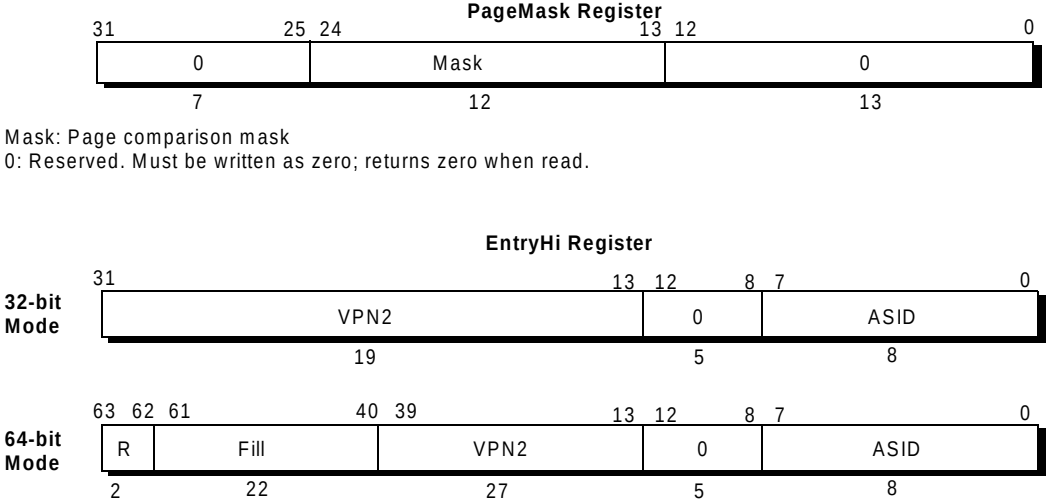


Figure 4-10 Format of a TLB Entry

Note: In setting up the TLB entry for 32- or 64-bit mode virtual address translation, the upper four bits of the physical frame number must be set to 0, to guarantee that the translated physical address will have bits 35:32 set to 0. (See Figure 4-3 and Figure 4-4.)

The format of the EntryHi, EntryLo0, EntryLo1, and PageMask registers is nearly the same as the TLB entry. The one exception is the *Global* field (*G* bit), which is used in the TLB, but is reserved in the EntryHi register. Figure 4-11 and Figure 4-12 describe the TLB entry fields shown in Figure 4-10.

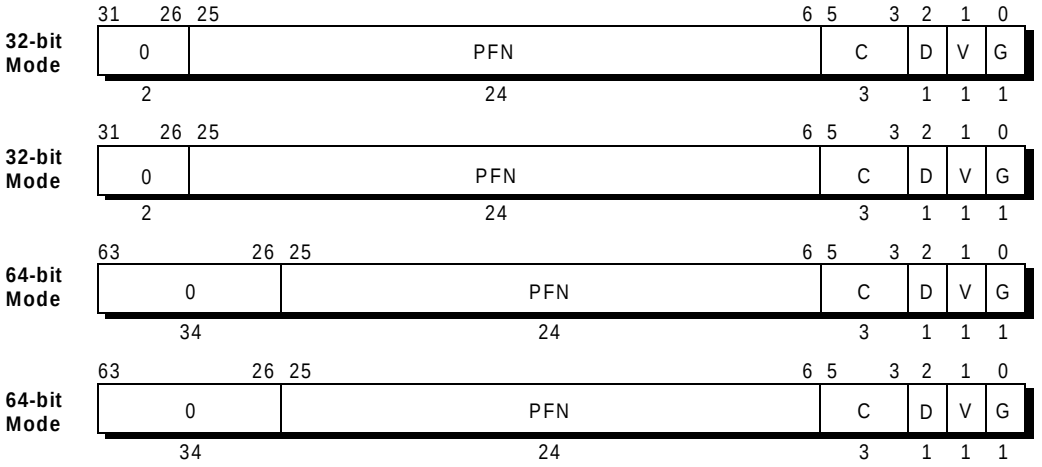


Mask: Page comparison mask
0: Reserved. Must be written as zero; returns zero when read.

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

Figure 4-11 Fields of the PageMask and EntryHi Registers

EntryLo0 and EntryLo1 Registers



PFN: Page frame number; the upper bits of the physical address
C: Specifies the TLB page cache attribute; see Table 4-9.
D: Dirty. If this bit is set, the page is marked as dirty and, therefore, writable. This bit is actually a write-protect bit that software can use to prevent alteration of data.
V: Valid. If this bit is set, it indicates that the TLB entry is valid; otherwise, a TLBL or TLBS miss occurs.
G: Global. If this bit is set in both Lo0 and Lo1, then the processor ignores the ASID during TLB lookup.
0: Reserved. Must be written as zero; returns zero when read.

Figure 4-12 Fields of the EntryLo0 and EntryLo1 Registers

The TLB page cache attribute (*C*) field specifies whether references to the page should be cached and, if so, the caching algorithm to use. Table 4-9 shows the cache attributes selected by the *C* field.

Table 4-9 TLB Page Cache Attribute (C) Field Encodings

***C* (5:3) Value**	**Page Cache Attribute**
0	Reserved
1	Cacheable, write-through, write allocate
2	Uncached
3	Cacheable write-back
4	Reserved
5	Reserved
6	Reserved
7	Uncached accelerated

4.4.2 Instruction and Data Micro-TLBs

In addition to the 48 double-entry TLB, the processor also implements two four-entry micro-TLBs dedicated to instruction and data address translations. If there is a miss in one of the micro-TLBs, there will be a pipeline stall while the ne TLB entry is transferred from the TLB to the micro-TLB. The four-entry micro-TLBs are fully associative with a psuedo Least Recently Used replacement algorithm. Each micro-TLB entry contains a mapping in any of the supported page sizes. The micro-TLBs are always guaranteed to be a subset of the TLB.

4.5 CP0 Registers

The following sections describe the CP0 registers. For a complete list of the CP0 registers, see Figure 4-9. The CP0 registers are assigned specifically as a software interface with memory management.

- Index register (0)
- Random register (1)
- EntryLo0 (2) and EntryLo1 (3) registers
- PageMask register (5)
- Wired register (6)
- EntryHi register (10)
- Config register (16)

CP0 also includes the following registers:

- PRId register (15)
- LLAddr register (17)
- TagLo (28) and TagHi (29) registers

4.5.1 Index Register (0)

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

The Index register also specifies the TLB entry affected by TLB Read (TLBR) or TLB Write Index (TLBWI) instructions. In a successful TLB Probe instruction, this register holds the index of a TLB entry whose contents match those of the EntryHi register. Although six bits are available to specify a TLB entry, the TLB has only 48 entries. Specifying an index greater than 47 will fail to match any TLB entry.

Figure 4-13 shows the format of the Index register; Table 4-10 describes the Index register fields.

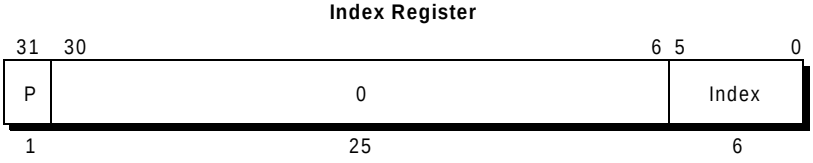


Figure 4-13 Index Register

Table 4-10 Index Register Field Descriptions

Field	Description
P	Probe failure. Set to 1 when the previous TLB Probe (TLBP) instruction was unsuccessful.
Index	Index to the TLB entry affected by the TLB Read and TLB Write instructions
0	Reserved. Must be written as zero; returns zero when read.

4.5.2 Random Register (1)

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

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

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

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

Figure 4-14 shows the format of the Random register. Table 4-11 describes the Random register fields.

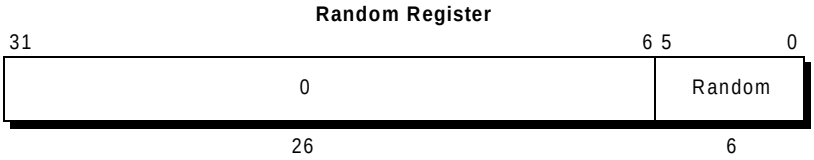


Figure 4-14 Random Register

Table 4-11 Random Register Field Descriptions

Field	Description
Random	TLB random index
0	Reserved. Must be written as zero; returns zero when read.

4.5.3 EntryLo0 (2) and EntryLo1 (3) Registers

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

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

The EntryLo0 and EntryLo1 registers are read/write registers. They hold the physical page frame number (PFN) of the TLB entry for even and odd pages, respectively, when performing TLB read and write operations. Figure 4-12 shows the format of these registers.

4.5.4 PageMask Register (5)

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

TLB read and write operations use this register as either a source or a destination; when virtual addresses are presented for translation into physical addresses, the corresponding bits in the TLB identify which virtual address bits among bits 24:13 are used in the comparison. When the *Mask* field is not one of the values shown in Table 4-13, the operation of the TLB is undefined.

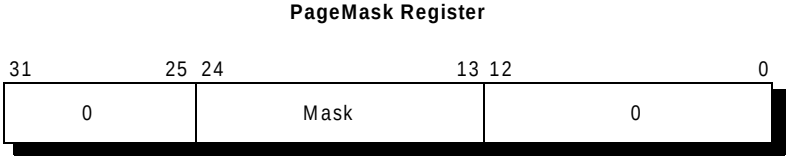


Figure 4-15 PageMask Register

Table 4-12 Random Register Field Descriptions

Field	Description
Mask	Mask for virtual page number. Table 4-13 provides mask values for all seven page sizes.
0	Reserved. Must be written as zero; returns zero when read.

Table 4-13 Mask Field Values for Page Sizes

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

4.5.5 Wired Register (6)

The Wired register is a read/write register that specifies the boundary between the wired and random entries of the TLB, as shown in Figure 4-16. Wired entries are fixed, nonreplaceable entries, which cannot be overwritten by a TLB write operation. Random entries can be overwritten.

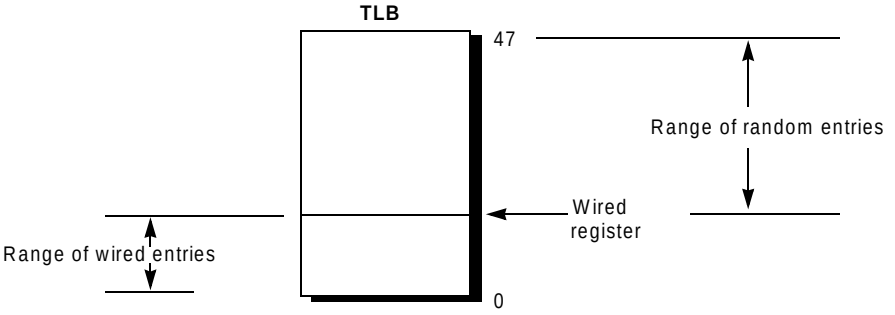


Figure 4-16 Wired Register Boundary

The Wired register is set to 0 upon system reset. Writing this register also sets the Random register to the value of its upper bound (Section 4.5.2). Figure 4-17 shows the format of the Wired register; Table 4-14 describes the register fields.

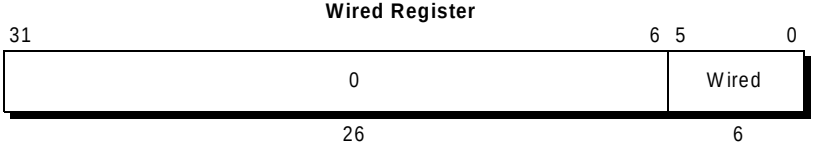


Figure 4-17 Wired Register

Table 4-14 Wired Register Field Descriptions

Field	Description
Wired	TLB wired boundary
0	Reserved. Must be written as zero; returns zero when read.

4.5.6 EntryHi Register (10)

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

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

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

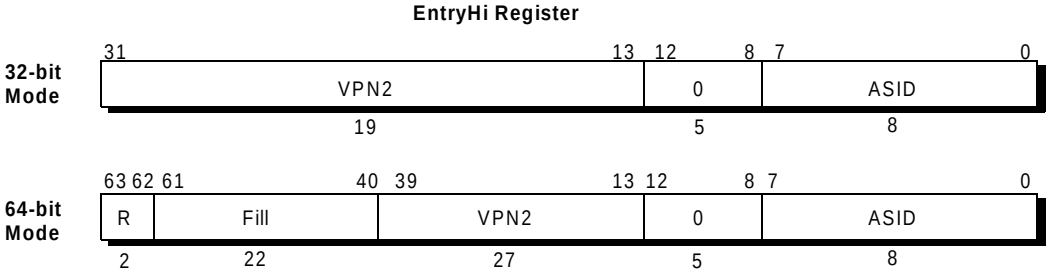


Figure 4-18 EntryHi Register

Table 4-15 EntryHi Register Field Descriptions

Field	Description
VPN2	Virtual page number divided by two (maps to two pages)
ASID	Address space ID field. An 8-bit field that lets multiple processes share the TLB; each process has a distinct mapping of otherwise identical virtual page numbers.
R	Region. (00 → User, 01 → Supervisor, 11 → Kernel) used to match vAddr 63:62
Fill	Reserved. 0 on read; ignored on write.
0	Reserved. Must be written as zero; returns zero when read.

4.5.7 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 4-17 shows the format of the PRId register; Table 4-16 describes the PRId register fields.

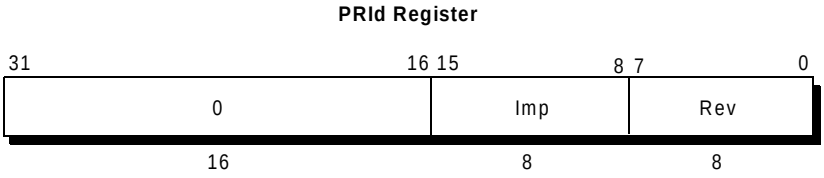


Figure 4-19 Processor Revision Identifier Register Format

Table 4-16 PRId Register Fields

Field	Description
Imp	Implementation number
Rev	Revision number
0	Reserved. Must be written as zero; returns zero when read.

The low-order byte (bits 7:0) of the PRId register is interpreted as a revision number, and the high-order byte (bits 15:8) is interpreted as an implementation number. The implementation number of the VR5432 processor is 0x54. The contents of the high-order halfword (bits 31:16) of the register are reserved.

The revision number is 13. The revision number can distinguish some chip revisions; however, there is no guarantee that changes to the chip will necessarily be reflected in the PRId register, or that changes to the revision number necessarily reflect real chip changes. For this reason, these values are not listed and software should not rely on the revision number in the PRId register to characterize the chip.

4.5.8 Config Register (16)

The Config register specifies various configuration options that can be selected.

The Config register must only be initialized during reset initialization while operating in uncached space; otherwise, the operation of the processor is undefined. The *EP*, *EM*, and *K0* fields are readable and writable by software and there is no other mechanism for writing to these fields. Their values are undefined following a cold or warm reset. The *EC* and *BE* fields are initialized by mode settings on processor input pins and cannot be modified by software.

Figure 4-20 shows the format of the Config register; Table 4-17 describes the Config register fields. Bits shown as 0 read as 0 and must be written as 0. Bits shown as 1 read as 1 and must be written as 1.

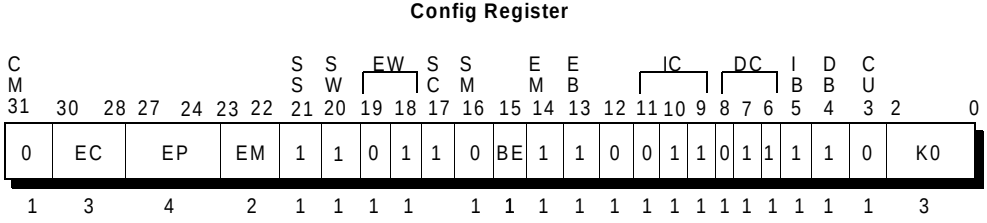


Figure 4-20 Config Register Format

Table 4-17 Config Register Field

Field	Description
EC	Processor clock to system clock ratio, read only: 000 → 2:1 001 → 2.5:1 010 → 3:1 011 → 4:1 All others→ Reserved and undefined
EP	Transmit data pattern for write-back data (Native mode, read/write): 0000 → WWWWWWWW 0001 → WWxWWxWWxWWx 0010 → WWxxWWxxWWxxWWxx 0011 → WxWxWxWxWxWxWxWx 0100 → WWxxxWWxxxWWxxxWWxxx 0101 → WWxxxxWWxxxxWWxxxxWWxxxx 0110 → WxxWxxWxxWxxWxxWxxWxxWxx 0111 → WWxxxxxxWWxxxxxxWWxxxxxxWWxxxxxx 1000 → WxxxWxxxWxxxWxxxWxxxWxxxWxxxWxxx All others→ Reserved and undefined (W = cycle in which a word transfer occurs; x = cycle in which no transfer occurs)
	Transmit data pattern for write-back data (VR4300 compatibility mode): 0000 → WWWWWWWW 0110 → WxxWxxWxxWxxWxxWxxWxxWxx All others→ Reserved and undefined (W = cycle in which a word transfer occurs; x = cycle in which no transfer occurs)
EM	SysAD mode (ignored in VR4300 compatibility mode): 00 → R4x00 compatible 01 → Multiple-split reads 10 → Pipelined writes 11 → Write reissue

Table 4-17 Config Register Fields (continued)

Field	Description
BE	Big-endian mode: 0 → Little endian 1 → Big endian
K0	*kseg0* coherency algorithm (see EntryLo0 and EntryLo1 registers and the *C* field of Table 4-9) (software writable): 001 → Cached, write-through 010 → Uncached 011 → Cached, write-back 111 → Uncached, accelerated 0 must be written as 0; returns zero when read. 1 must be written as 1; returns one when read.

4.5.9 Load Linked Address (LLAddr) Register (17)

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

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

Figure 4-21 shows the format of the LLAddr register; the PAddr field represents bits of the physical address, PA (35:4). PA (35:32) are always zero.

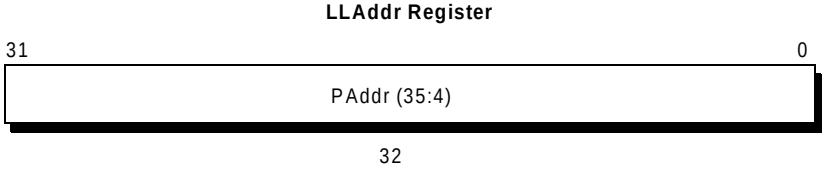


Figure 4-21 LLAddr Register Format

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

The TagLo and TagHi registers are 32-bit read/write registers used during cache initialization and diagnostics to hold the cache tag and parity. The Tag registers are written by the CACHE and MTC0 instructions.

The CACHE Index Store Tag instruction copies the *PTagLo*, *PState*, *L*, *R*, and *P* bits into the cache. The CACHE Index Load Tag instruction copies the cache tag *PTagLo*, *PState*, *L*, *R*, and *P* bits into the TagLo register.

Figure 4-22 shows the format of these registers. Table 4-18 lists the field definitions of the TagLo and TagHi registers. The TagHi register is defined in the architecture but is not used (it is reserved for future use).

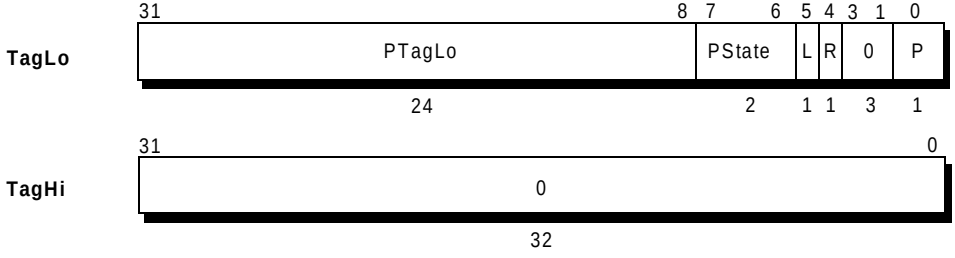


Figure 4-22 TagLo and TagHi Register Formats

Table 4-18 Cache Tag Register Fields

Field	Description
PTagLo	Specifies the physical address bits 35:12; bits 35:32 are always zero.
PState	Specifies the cache state 0 → Invalid 1 → Reserved 2 → Clean 3 → Dirty
L	Cache line lock bit
R	Cache line *LRU* bit
P	Cache line even parity bit
0	Reserved. Must be written as zero; returns zero when read.

4.6 Virtual-to-Physical Address Translation Process

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

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

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

Figure 4-23 illustrates the TLB address translation process.

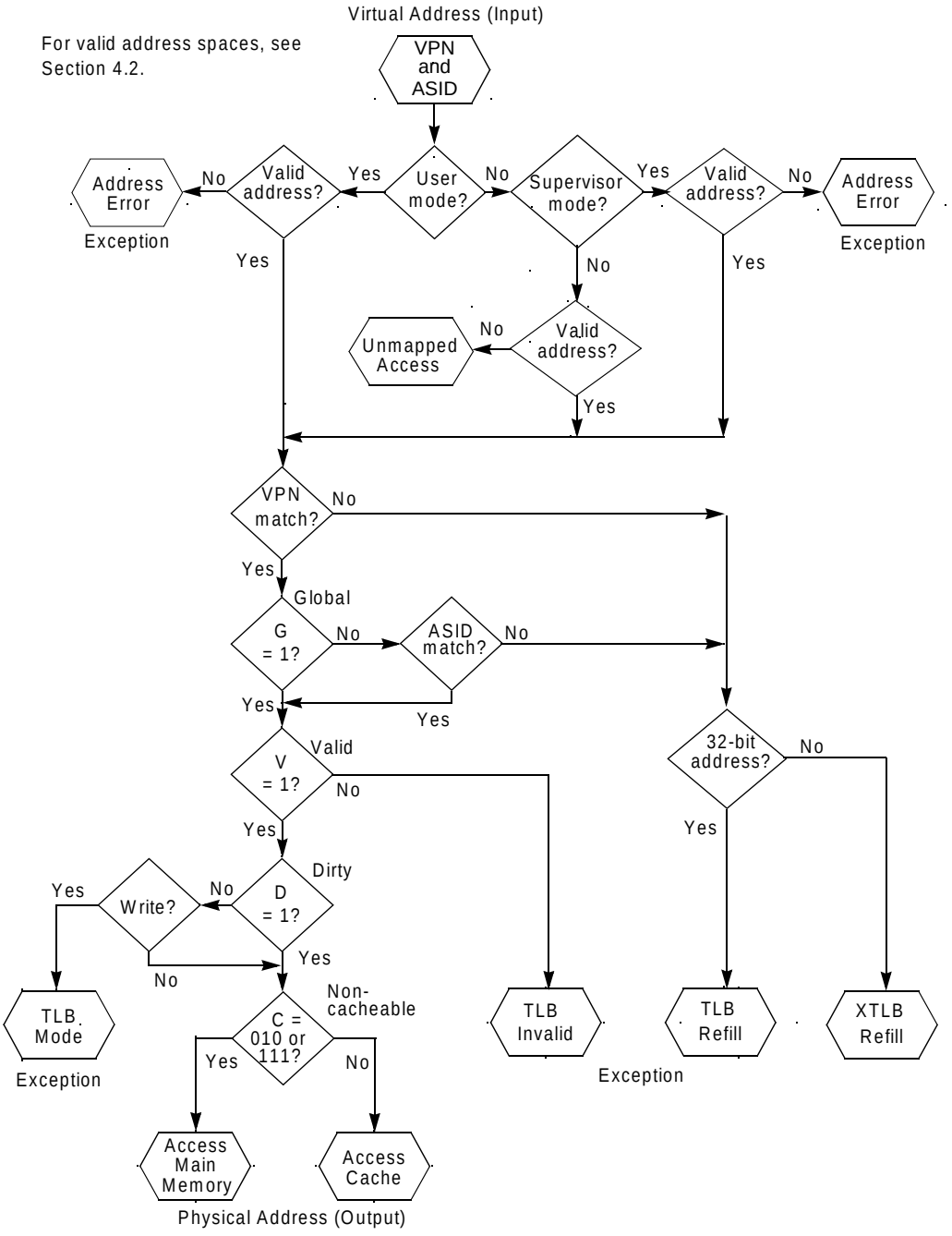


Figure 4-23 TLB Address Translation

4.7 TLB Exceptions

If there is no TLB entry that matches the virtual address, a TLB Miss exception occurs. If the access control bits (*D* and *V*) indicate that the access is not valid, a TLB Modification or TLB Invalid exception occurs. If the *C* bits equal 010_2, the physical address that is retrieved accesses main memory, bypassing the cache.

4.8 TLB Instructions

Table 4-19 lists the instructions that the CPU provides for working with the TLB.

Table 4-19 TLB Instructions

Opcode	Description
TLBP	Translation Lookaside Buffer Probe
TLBR	Translation Lookaside Buffer Read
TLBWI	Translation Lookaside Buffer Write Index
TLBWR	Translation Lookaside Buffer Write Random

Cache Organization and Operation

5

This chapter describes the cache memory's place in the VR5432 memory configuration and individual cache organization.

5.1 Memory Organization

Figure 5-1 shows the VR5432 system memory hierarchy. In the logical memory hierarchy, the caches lie between the CPU and main memory. They are designed to make the speed-up of memory accesses transparent to the user.

Each functional block in Figure 5-1 has the capacity to hold more data than the block above it. For instance, 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.

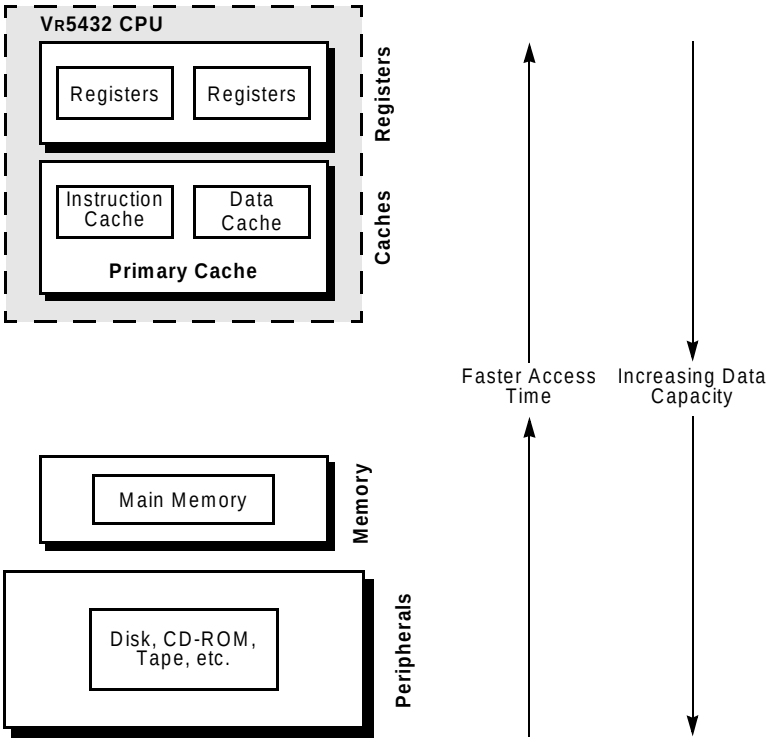


Figure 5-1 Logical Hierarchy of Memory

The VR5432 processor has two on-chip caches: one holds instructions (the instruction cache, or I-cache), and the other holds data (the data cache, or D-cache). The instruction and data caches can be read in one PClock cycle.

Data writes are pipelined and can complete at a rate of one per PClock cycle. In the first stage of the cycle, the store address is translated and the tag is checked; in the second stage, the data is written into the data RAM.

Figure 5-2 provides a block diagram of the VR5432 cache and memory model.

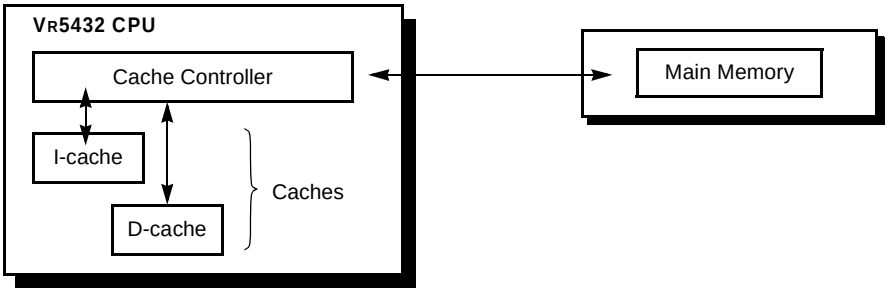


Figure 5-2 VR5432 Cache Support

5.2 Primary Cache Organization

This section describes the organization of the on-chip data and instruction caches.

5.2.1 Cache Line Lengths

A cache line is the smallest unit of information that can be fetched from main memory for the cache and is represented by a single tag.

The line size for the instruction and data caches is 32 bytes each.

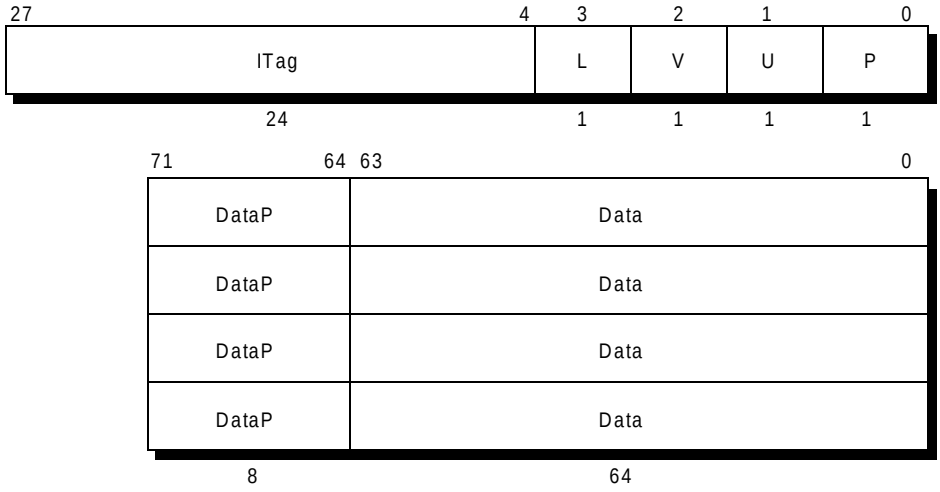
5.2.2 Cache Sizes

The VR5432 instruction and data caches are 32 KB each.

5.2.3 Instruction Cache Organization

The VR5432 processor I-cache has the following characteristics:

- Two-way set-associative structur
- Virtual address index
- Physical tag checks
- 32-byte line organizati
- Line-lockable opti



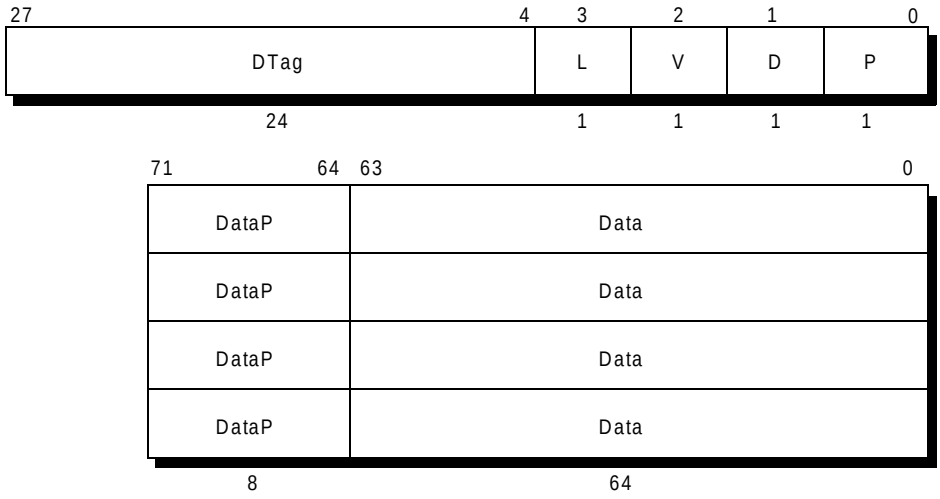
Itag: Instruction tag
L: Lock
V: Valid
U: Unused
P: Even parity for ITag
DataP: Even parity for the data
Data: I-cache data

Figure 5-3 Primary Instruction Cache Line Format

5.2.4 Data Cache Organization

The VR5432 processor D-cache has the following characteristics:

- Write-back or write-through cache algorithms
- Two-way set-associative structur
- Virtual address index
- Physical tag checks
- 32-byte line organizati
- Line-lockable opti



Dtag: Data tag
L: Lock
V: Valid
D: Dirty
P: Even parity for DTag
DataP: Even parity for the data
Data: D-cache data

Figure 5-4 Primary Data Cache Line Format

CPU Exceptions

6

This chapter describes CPU exception processing, including 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 one of these exceptions, the normal sequence of instruction execution is suspended and the processor enters Kernel mode.

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

When an exception occurs, the CPU loads the Exception Program Counter (EPC) register with a location where execution can restart after the exception has been serviced. The restart location in the EPC register is the address of the instruction that caused the exception or, if the instruction was executing in a branch delay slot, the address of the Branch instruction immediately preceding the delay slot.

The registers described later in the section assist in this exception processing by retaining address, cause, and status information.

6.2 Exception Processing Registers

This section describes the CP0 registers that are used in exception processing. Table 6-1 lists these registers, along with their numbers—each register has a unique identification number that is referred to as its register number. The remaining CP0 registers are used in memory management.

Software examines the CP0 registers during exception processing to determine the cause of the exception and the state of the CPU at the time the exception occurred. The registers in Table 6-1 are used in exception processing, and are described in the sections that follow.

Table 6-1 CP0 Exception Processing Registers

Register Name	Register Number
Context	4
BadVAddr (Bad Virtual Address)	8
Count	9
Compare	11
Status	12
Cause	13
EPC (Exception Program Counter)	14
WatchLo	18
WatchHi	19
XContext	20
Performance Counter	25
PErr (Parity Error)	26
CacheErr (Cache Error and Status)	27
ErrorEPC (Error Exception Program Counter)	30

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

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; this array is an operating system data structure that stores virtual-to-physical address translations. When there is a TLB miss, the operating system loads the TLB with the missing translation from the PTE array. Normally, the operating system uses the Context register to address the current page map, which resides in the kernel-mapped segment, *kseg3*. The Context register duplicates some of the information provided in the BadVAddr register, but the information is arranged in a form that is more useful for a software TLB exception handler. Figure 6-1 shows the format of the Context register; Table 6-2 describes the Context register fields.

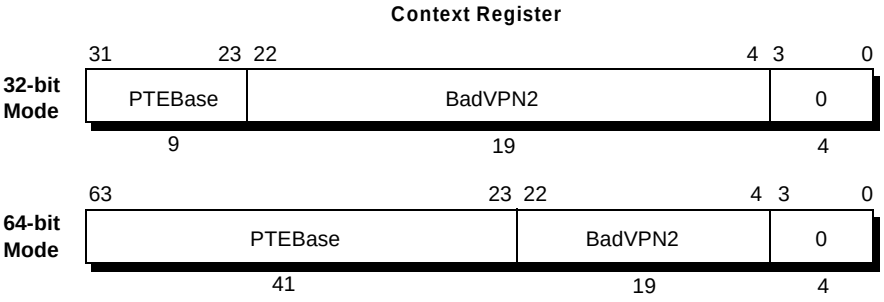


Figure 6-1 Context Register Format

Table 6-2 Context Register Fields

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

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

6.2.2 Bad Virtual Address Register (BadVAddr) (8)

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

Figure 6-2 shows the format of the BadVAddr register.

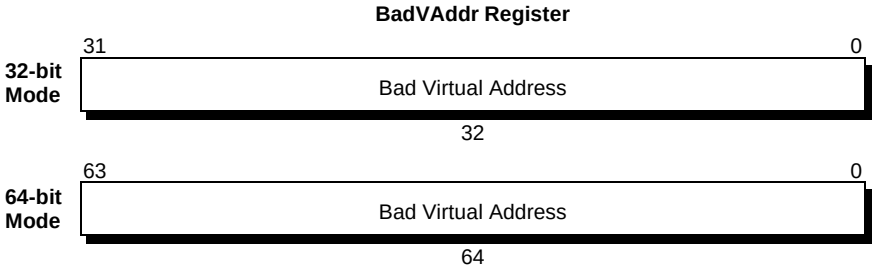


Figure 6-2 BadVAddr Register Format

Note: The BadVAddr register does not save information for bus errors, since bus errors are not addressing errors. Also, in the case of an Instruction Virtual Address (IVA), bits 58:40 are fill bits, but in the case of a Data Virtual Address (DVA), all of the 64 bits are calculated.

6.2.3 Count Register (9)

The Count register acts as a timer, incrementing at a constant rate whether or not an instruction is executed, retired, or any forward progress is made through the pipeline. On the VR5432, the timer counts at half the maximum issue rate (i.e., half the PClock rate).

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

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

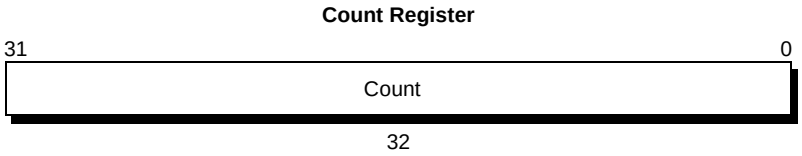


Figure 6-3 Count Register Format

6.2.4 Compare Register (11)

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

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

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

For diagnostic purposes, the Compare register is a read/write register. In normal use however, the Compare register is write only. Figure 6-4 shows the format of the Compare register.

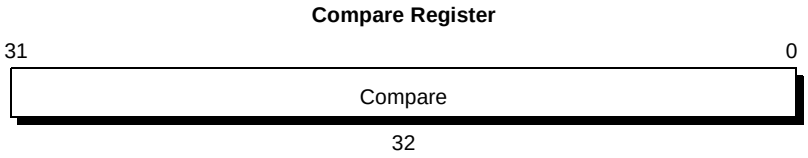


Figure 6-4 Compare Register Format

IP (7) is also used by the Performance Counter. The interrupt handler for *IP* (7) should take this into account.

6.2.5 Status Register (12)

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

- The 8-bit *Interrupt Mask* (*IM*) field controls the enabling of eight interrupt conditions. Interrupts must be enabled before they can b asserted, and the corresponding bits are set in both th *Interrupt Mask* field of the Status register and the *Interrupt Pending* field of the Cause register. *IM* (1:0) are software interrupt masks, *IM* (6:2) correspond to *Int* (4:0), and *IM* (7) is a timer interrupt mask.
- The 4-bit *Coprocessor Usability* (*CU*) field controls the usability of possible coprocessors. Regardless of the *CU0* bit setting, CP0 is always usable in Kernel mode. For all other cases, an access to a unusable coprocessor causes an exception *CU3* selects MIPS IV.
- The 9-bit *Diagnostic Status* (*DS*) field is used for self-testing, and checks the cache and virtual memory system.

6.2.5.1 Status register format

Figure 6-5 shows the format of the Status register. Table 6-3 describes the Status register fields. Figure 6-6 and Table 6-4 provide additional information on the *Diagnostic Status* (*DS*) field. All bits in the *DS* field except *TS* are readable and writable. Bits shown as 0 read as 0 and must be written as 0.

Bits 27, 25, 23, and 19 are unused and reserved. However, they are readable and writable, and therefore must be written as 0 during the initialization process.

Bit 16 is unsused and reserved. However, it is readable and writable. Write a 1 to this bit during initialization.

Special consideration must be taken when porting to an operating system or application written for other versions of MIPS processors. For example, when the VR5432 is configured to run in VR43xx Bus Protocol mode, the VR5432 can be integrated with the system that was designed to run with a VR43xx processor. In this case, the original operating system may interpret Status register bit 27 as Reduced Power (RP) mode, as defined in the VR43xx processor specification.

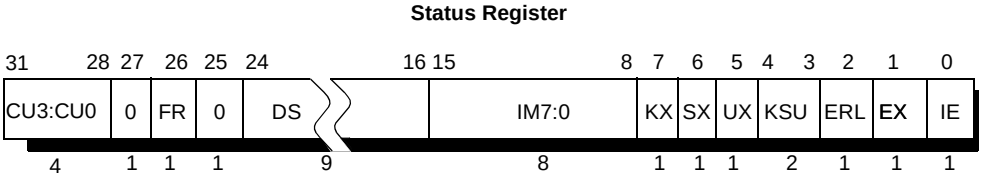


Figure 6-5 Status Register

Table 6-3 Status Register Fields

Field	Description
CU3:CU0	Originally controlled the usability of each of the four coprocessor unit numbers; redefined for the MIPS IV instruction set. Setting *CU3* enables the MIPS IV instruction set extensions. CP0 is always usable when in Kernel mode, regardless of the setting of the *CU0* bit. 1 → usable 0 → unusable
FR	Enables additional floating-point registers 0 → 16 registers 1 → 32 registers
DS	Diagnostic status field (see Figure 6-6)
IM7:0	Interrupt Mask: Controls the enabling of each of the external, internal, and software interrupts. An interrupt occurs if interrupts are enabled, and the corresponding bits are set in both the *Interrupt Mask* field of the Status register and the *Interrupt Pending* field of the Cause register. 0 → disabled 1→ enabled
KX	Enables 64-bit addressing in Kernel mode. The extended-addressing TLB Refill exception is used for TLB misses on kernel addresses. 0 → 32-bit 1 → 64-bit
SX	Enables 64-bit addressing and operations in Supervisor mode. The extended-addressing TLB Refill exception is used for TLB misses on supervisor addresses. 0 → 32-bit 1 → 64-bit

Table 6-3 Status Register Fields (continued)

Field	Description
UX	Enables 64-bit addressing and operations in User mode. The extended-addressing TLB Refill exception is used for TLB misses on user addresses. 0 → 32-bit 1 → 64-bit
KSU	Mode bits 10_2 → User 01_2 → Supervisor 00_2 → Kernel
ERL	Error Level; set by the processor when Reset, Soft Reset, NMI, or Cache Error exceptions are taken. 0 → normal 1 → error When *ERL* is set: Interrupts are disabled. The *KSU* bits will indicate that the processor is in Kernel mode. The ERET instruction will use the return address held in ErrorEPC instead of EPC. *kuseg* and *xkuseg* are treated as unmapped and uncached regions. This allows main memory to be accessed in the presence of cache errors.
EXL	Exception Level; set by the processor when any exception other than Reset, Soft Reset, NMI, or Cache Error exceptions are taken. 0 → normal 1 → exception When EXL is set: Interrupts are disabled. The *KSU* bits will indicate that the processor is in Kernel mode. TLB Refill exceptions will use the general exception vector instead of the TLB Refill vector. EPC will not be updated if another exception is taken.
IE	Interrupt Enable 0 → disable interrupts 1 → enables interrupts

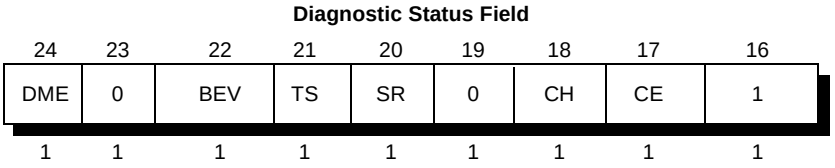


Figure 6-6 Status Register DS Field

Table 6-4 Status Register Diagnostic Status Bits

Bit	Description
DME	Enables entry into Debug mode. 0 → debug break events are ignored, and the DBREAK instruction causes a Reserved Instruction exception 1 → debug break events and the DBREAK instruction cause entry into Debug mode
BEV	Controls the location of TLB Refill and general exception vectors. 0 → normal 1 → bootstrap
TS	Indicates that a TLB shutdown has occurred (read only); used to avoid damage to the TLB if more than one TLB entry matches a single virtual address. 0 → did not occur 1 → did occur After a TLB shutdown, the processor must be reset to restart. TLB shutdowns can occur even when a TLB entry with which the virtual address has matched is set to be invalid (*V* bit of the entry is cleared).
SR	1→ Indicates that a Soft Reset or NMI has occurred
CH	CP0 condition bit. Read/write access to software, unaffected by hardware events.
CE	Create parity error for cache diagnostics. When set, the contents of the PErr register are loaded into the cache parity bits rather than the computed parity bits.
DE	DE bit .

6.2.5.2 Status register modes and access states

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

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

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

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

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

- The processor is in User mode whe *KSU* = 10_2, *EXL* = 0, and *ERL* = 0.
- The processor is in Supervisor mode when *KSU* = 01_2, *EXL* = 0, and *ERL* = 0.
- The processor is in Kernel mode when *KSU* = 00_2, or *EXL* = 1, or *ERL* = 1.

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

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

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

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

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

6.2.5.3 Status register reset

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

- *ERL* and *BEV* = 1

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

6.2.6 Cause Register (13)

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

Figure 6-7 shows the fields of this register. Table 6-5 describes the Cause register fields.

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

Table 6-5 Cause Register Fields

Field	Description
BD	Indicates whether the EPC was adjusted because the last exception taken occurred in a branch delay slot. 1 → delay slot 0 → normal
CE	Coprocessor unit number referenced when a Coprocessor Unusable exception is taken.
IP7:0	Indicates that an interrupt is pending. *IP* (7) indicates a timer interrupt or Performance Counter overflow. *IP* (6:2) are the external interrupts that are set by an interrupt signal or a write on the SysAD bus. *IP* (1:0) are software interrupts, which may be written to set or clear interrupts. 1 → interrupt pending 0 → no interrupt
ExcCode	Exception code field (see Table 6-6)
0	Reserved. Must be written as 0; returns 0 when read.

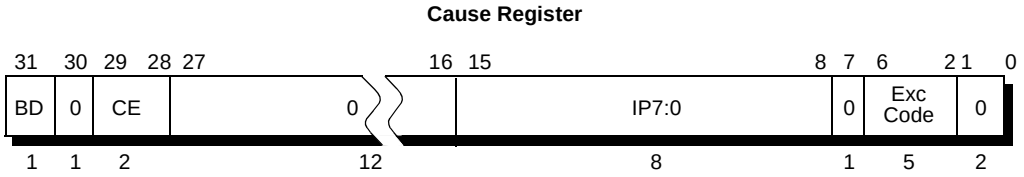


Figure 6-7 Cause Register Format

Table 6-6 Cause Register ExcCode Field

Exception Code Value	Mnemonic	Description
0	Int	Interrupt
1	Mod	TLB Modified exception
2	TLBL	TLB exception (load or instruction fetch)
3	TLBS	TLB exception (store)
4	AdEL	Address Error exception (load or instruction fetch)
5	AdES	Address Error exception (store)
6	IBE	Bus Error exception (instruction fetch)
7	DBE	Bus Error exception (data reference: load or store)
8	Sys	System Call exception
9	Bp	Breakpoint exception
10	RI	Reserved Instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Arithmetic Overflow exception
13	Tr	Trap exception
14	—	Reserved
15	FPE	Floating-Point exception
16–22	—	Reserved
23	Watch	Reference to WatchHi/WatchLo address
24–31	—	Reserved

6.2.7 Exception Program Counter (EPC) Register (14)

The Exception Program Counter (EPC) is a read/write register that contains the address where processing resumes after an exception has been serviced.

For synchronous exceptions, the EPC register contains either:

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

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

Figure 6-8 shows the format of the EPC register. In 32-bit mode, bits 40 through 58 will be the sign-extended 32-bit address. In 64-bit mode, these bits will be all ones if the segment is *ckseg0*, *ckseg1*, *cksseg*, or *ckseg3*. For all other segments, these bits will all be zero.

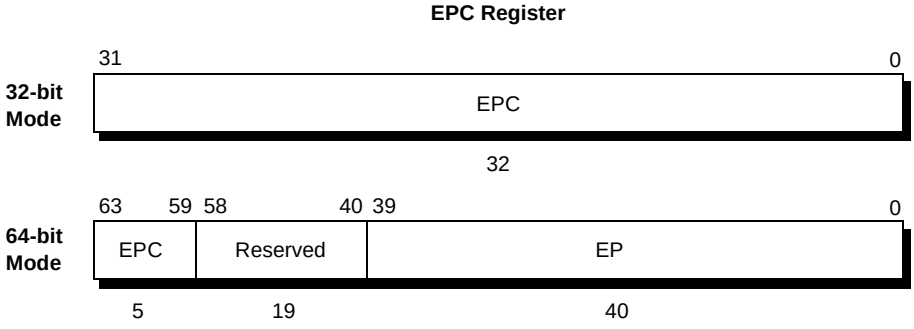


Figure 6-8 EPC Register Format

6.2.8 XContext Register (20)

The read/write XContext register contains a pointer to an entry in the page table entry (PTE) array, an operating system data structure that stores virtual-to-physical address translations. When there is a TLB miss, the operating system software loads the TLB with the missing translation from the PTE array. The XContext register duplicates some of the information provided in the BadVAddr register and puts it in a form useful for a software TLB exception handler. The XContext register is for use with the XTLB refill handler, which loads TLB entries for references to a 64-bit address space and is included solely for operating system use. The operating system sets the *Page Table Entry Base* (*PTEBase)* field in the register, as needed. Figure 6-9 shows the format of the XContext register; Table 6-7 describes the XContext register fields.

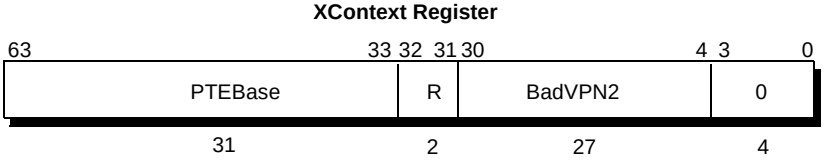


Figure 6-9 XContext Register Format

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

Table 6-7 XContext Register Fields

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

6.2.9 WatchLo and WatchHi Registers (18 and 19)

The processor provides a debugging feature to detect references to a selected physical address; load and store operations can be programmed to cause a Watch exception. Figure 6-10 shows the format of the WatchLo and WatchHi registers. Table 6-8 describes the WatchLo and WatchHi register fields.

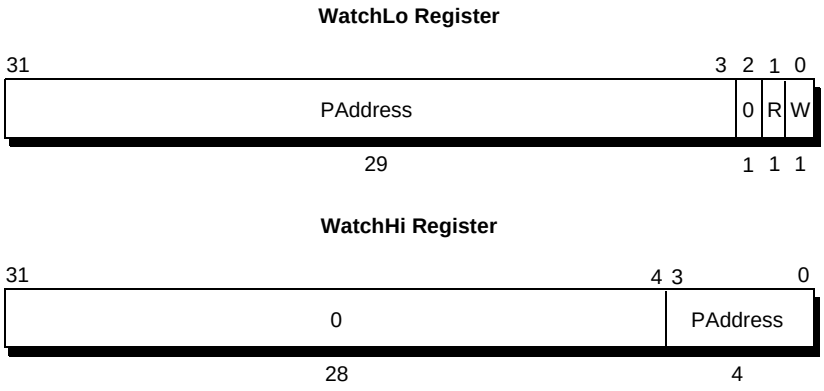


Figure 6-10 WatchLo and WatchHi Register Formats

Table 6-8 WatchLo and WatchHi Register Fields

Field	Description
PAddress	Physical address bits [35:3] for triggering Watch exception
R	If set, triggers an exception on reads.
W	If set, triggers an exception on writes.
0	Reserved. Must be written as 0; returns 0 when read.

6.2.10 Performance Counter Registers (25)

The processor has two Performance Counter registers and two associated Performance Counter Control registers, which are mapped into CP0 register 25. Use the MTPS/MFPS/MTPC/MFPC instructions to read or write the Performance Counter and Control registers.

Each counter is a 32-bit read/write register and is incremented by one each time the countable event, specified in its associated control register, occurs. Each counter can independently count one type of event at a time. The format of the Control register is shown in Figure 6-11; the register fields are described in Table 6-9.

The counter asserts an interrupt (IP7), when the counter overflows and the associated Performance Control register enables the interrupt. Counting continues after counter overflow whether or not an interrupt is signaled.

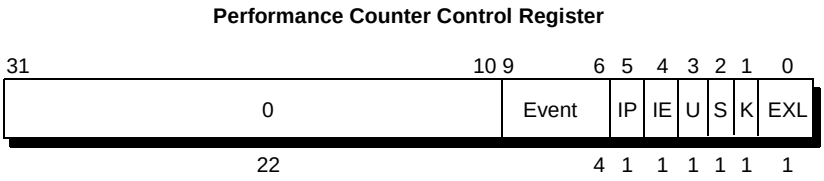


Figure 6-11 Performance Counter Control Register Format

Table 6-9 Performance Counter Control Register Fields

Field	Description
Event	Specifies the event to be counted. See Table 6-10.
IP	Interrupt pending.
IE	Enables the IP7 interrupt when the associated counter overflows.
U, S, K, EXL	Specifies a processor mode in which the event is counted. U → User mode S → Supervisor mode K → Kernel mode (except when *ERL* or *EXL* are 1) EXL → Kernel mode (when *EXL* is 1) The processor mode count enables can be set individually.
0	Reserved. Must be written as 0; returns 0 when read.

Table 6-10 Event Field Encoding

Event Code	Description
0	Processor cycles (PClock)
1	(Instructions executed)/2 and truncated
2	Load, prefetch/CacheOps execution (no sync)
3	Store execution
4	Branch execution (no jumps or Jump registers)
5	(Floating-point instruction execution)/2 and truncated. The instruction count includes COP1 and COP1X, but not LWC1, LDC1, SWC1, or SDC1.
6	Doublewords flushed to main memory (no uncached stores)
7	JTLB refills
8	Data cache misses (no I-cache misses)
9	Instruction cache misses (no D-cache misses)
10	Branches mispredicted

When using these counters, the following rules apply:

- An instruction is considered executed even if the instruction causes an exception
- The Performance Counter registers may be preloaded with an MTPC instruction
- The *Interrupt Enabl* bit must be set to trigger an IP7 interrupt
- The interrupt handler routine must check the Performance Counter registers to determine if the interrupt was caused by a Performance Counter or by the Count register matching the Compare register.

6.2.11 Parity Error (PErr) Register (26)

The 8-bit Parity Error (PErr) register reads or writes the cache parity bits for cache initialization, cache diagnostics, or cache error processing.

The PErr register is loaded from the cache parity bits by the Cache Load Index Tag operation. The contents of the PErr register are written into the cache parity bits on Store instructions (instead of the computed parity) when the *CE* bit of the Status register is set.

Figure 6-12 shows the format of the PErr register; Table 6-11 describes the register fields.

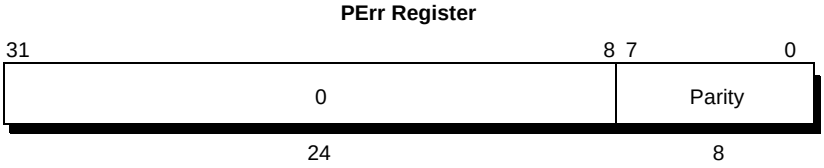


Figure 6-12 PErr Register Format

Table 6-11 PErr Register Fields

Field	Description
Parity	An 8-bit field specifying the parity bits read from or written to a cache
0	Reserved. Must be written as 0; returns 0 when read.

6.2.12 Cache Error (CacheErr) Register (27)

The 32-bit read-only CacheErr register holds information for processing parity errors in the cache. Parity errors cannot be corrected.

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

Figure 6-13 shows the format of the CacheErr register and Table 6-12 describes the CacheErr register fields.

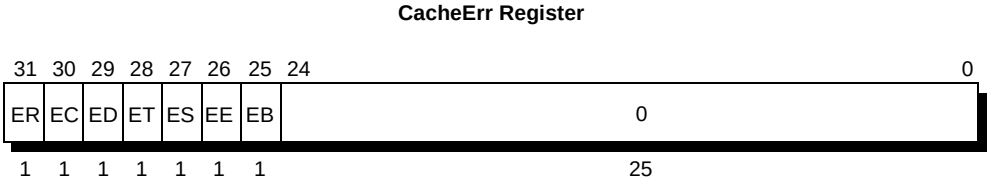


Figure 6-13 CacheErr Register Format

Table 6-12 CacheErr Register Fields

Field	Description
ER	Type of reference 0 → instruction 1 → data
EC	Cache level of the error 0 → primary (i.e., on-chip cache) 1 → reserved
ED	Indicates if a data field error occurred 0 → no error 1 → error
ET	Indicates if a tag field error occurred 0 → no error 1 → error
ES	This bit is set if the error occurred on the first doubleword. 0 → not first doubleword 1 → first doubleword

Table 6-12 CacheErr Register Fields (continued)

Field	Description
EE	This bit is set if the error occurred on the SysAD bus. 0 → not on SysAD bus 1 → on SysAD bus
EB	This bit is set if a data error occurred in addition to the instruction error (indicated by the remainder of the bits). If so, this requires flushing the data cache after fixing the instruction error. 0 → error in one cache 1 → error in both caches
0	Reserved. Must be written as 0; returns 0 when read.

6.2.13 Error Exception Program Counter (ErrorEPC) Register (30)

The ErrorEPC register is similar to the EPC register, except that ErrorEPC is used on Parity Error exceptions. It is also used to store the program counter (PC) on Reset, Soft Reset, and Nonmaskable Interrupt (NMI) exceptions.

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

- The virtual address of the instruction that caused the exceptio
- The virtual address of the immediately preceding Branch or Jump instruction, when this address is in a branch delay slo

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

Error EPC is not logged when parity is disabled.

Figure 6-14 shows the format of the ErrorEPC register. In 32-bit mode, bits 40 through 58 will be the sign-extended 32-bit address. In 64-bit mode, these bits will be all ones if the segment is *ckseg0*, *ckseg1*, *cksseg*, or *ckseg3*. For all other segments, these bits will all be zero.

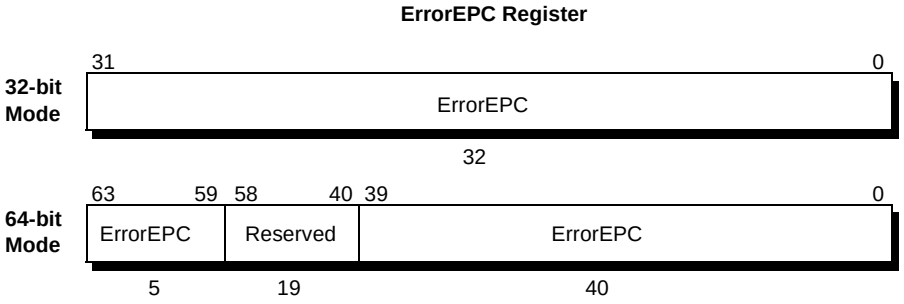


Figure 6-14 ErrorEPC Register Format

6.3 Processor Exceptions

This section describes the processor exceptions; it describes the cause of each exception, its processing by the hardware, and servicing by a handler (software). The exception types and exception processing operations are described in Section 6.3.1.

6.3.1 Exception Types

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

- Reset
- Soft Reset
- Nonmaskable Interrupt (NMI)
- Cache Error
- Remaining processor exception

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

When the processor takes an exception, the *EXL* bit is set to 1, which means the system is in Kernel mode. After saving the appropriate state, the exception handler typically changes *KSU* to Kernel mode and resets the *EXL* bit back to 0 to re-enable exceptions. By executing an ERET instruction, the handler restores the previous value of the *KSU* field and sets the *EXL* bit back to 0.

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

6.3.1.1 Reset exception process

Figure 6-15 shows the Reset exception process.

```
T:  undefined
    Random ← TLBENTRIES–1
    Wired ← 0
    Config ← 0 || EC || xxxxxx || 110110 || BE || 110011011110 || xxx
    ErrorEPC ← PC
    SR ← xxxxxxx || 0 || x || 1 || 0 || 0 || xxxxxxxxxxxxxxxx || 1 || xx
    PC ← 0xFFFF FFFF BFC0 0000
```

Figure 6-15 Reset Exception Process

6.3.1.2 Cache Error exception process

Figure 6-16 shows the Cache Error exception process.

```
T: ErrorEPC ← PC
   CacheErr ← ER || EC || ED || ET || ES || EE || EB || 0^25
   SR ← SR31:3 || 1 || SR1:0
   if SR22 = 1 then      /*What is the BEV bit setting*/
       PC ← 0xFFFF FFFF BFC0 0200 + 0x100 /*Access boot-PROM area*/
   else
       PC ← 0xFFFF FFFF A000 0000 + 0x100 /*Access main memory area*/
   endif
```

Figure 6-16 Cache Error Exception Process

6.3.1.3 Soft Reset and NMI exception process

Figure 6-17 shows the Soft Reset and NMI exception process.

```
 T: ErrorEPC ← PC
    SR ← SR31:23 || 1 || SR21 || 1 || SR19:3 || 1 || SR1:0
    PC ← 0xFFFF FFFF BFC0 0000
```

Figure 6-17 Soft Reset and NMI Exception Process

6.3.1.4 General exception process

Figure 6-18 shows the process used for exceptions other than Reset, Soft Reset, NMI, and Cache Error.

```
T:   Cause ← BD || 0 || CE || 0^12 || Cause15:8 || ExcCode || 0^2
if SR1 = 0 then/* System is in User or Supervisor mode, no current exception */
           EPC ← PC
    endif
    SR ← SR31:2 || 1 || SR0
    if SR22 = 1 then
        PC ← 0xFFFF FFFF BFC0 0200 + vector  /*access to uncached space*/
    else
        PC ← 0xFFFF FFFF 8000 0000 + vector  /*access to cached space*/
    endif
```

Figure 6-18 General Exception Processing

6.3.2 Exception Vector Locations

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

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

Table 6-13 shows the 64-bit mode vector base address for all exceptions; the 32-bit mode address is the low-order 32 bits (for instance, the base address for NMI in 32-bit mode is 0xBFC0 0000).

Table 6-14 shows the vector offset added to the base address to create the exception address.

Table 6-13 Exception Vector Base Addresses

BEV Bit	VR5432 Processor Vector Base Address
0	0xFFFF FFFF 8000 0000
1	0xFFFF FFFF BFC0 0200

6.4 Exception Vector Offsets

Table 6-14 Exception Vector Offsets

Exception	VR5432 Processor Vector Offset
TLB Refill, *EXL* = 0	0x000
XTLB Refill, *EXL* = 0 (X = 64-bit TLB)	0x080
Cache Error	0x100
Others	0x180

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

6.4.1 TLB Refill Vector Selection

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

- TLB Refill: References to 32-bit address space
- XTLB Refill: References to 64-bit address space

The TLB Refill vector selection is based on the address space of the address (user, supervisor, or kernel) that caused the TLB miss, and the value of the corresponding extended addressing bit in the Status register (*UX*, *SX*, or *KX*). The current operating mode of the processor is not important, except that it plays a part in specifying in which address space an address resides. The Context and XContext registers are entirely separate page-table-pointer registers that point to and refill from two separate page tables. For all TLB exceptions (Refill, Invalid, TLBL, or TLBS), the *BadVPN2* fields of both registers are loaded as they were in the VR4000.

In contrast to the VR5432, the VR4000 processor selects the vector based on the current operating mode of the processor (User, Supervisor, or Kernel) and the value of the corresponding extended addressing bit in the Status register (*UX*, *SX*, or *KX*). In addition, the Context and XContext registers are not implemented as entirely separate registers; the *PTEbase* fields are shared. A miss to a particular address goes through either TLB Refill or XTLB Refill, depending on the source of the reference. There can be only a single page table unless the refill handlers execute address deciphering and page table selection in software.

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

Table 6-15 lists the TLB Refill vector locations, based on the address that caused the TLB miss and its corresponding mode bit.

Table 6-15 TLB Refill Vectors

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

6.4.2 Priority of Exceptions

Table 6-16 describes exceptions, ranging from highest to lowest priority. Although more than one exception can occur for a single instruction, only the exception with the highest priority is reported.

Table 6-16 Exception Priority Order

Reset *(highest priority)*
Soft Reset
Nonmaskable Interrupt (NMI)
Debug Break Event—Instruction fetch
Address Error—Instruction fetch
TLB Refill—Instruction fetch
TLB Invalid—Instruction fetch
Cache Error—Instruction fetch
Bus Error—Instruction fetch
Integer Overflow, Trap, System Call, Breakpoint, Reserved Instruction, Coprocessor Unusable, Floating-point, or Debug
Address Error—Data access
TLB Refill—Data access
TLB Invalid—Data access
TLB Modified—Data write
Cache Error—Data access
Bus Error—Data access
Watch
Interrupt *(lowest priority)*

Generally speaking, the exceptions described in the following sections are processed by hardware; these exceptions are then serviced by software.

Note: This table shows the priority of the processor control logic. When the external exception signals (Ints, NMI) are asserted with the same timing, the arriving timing for the control logic may be different.

6.4.3 Reset Exception

Cause

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

Processing

The CPU provides a special interrupt vector for this exception:

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

The Reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to process this exception. It also means the processor can fetch and execute instructions while the caches and virtual memory are in an undefined state.

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

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

Table 6-17 Register State

Register Name	Cold Reset (Power-On Reset)	Soft Reset	NMI
Status (*SR*)	0	1	1
(*ERL*)	1	1	1
(*BEV*)	1	1	1
(*TS*)	0	unchanged	unchanged
Random	47	unchanged	unchanged
Wired	0	unchanged	unchanged
Config (*EC*)	hardwired	hardwired	hardwired
(*EM*)	0	unchanged	unchanged
(*BE*)	hardwired	hardwired	hardwired
PRId (*Impl*)	0x54	0x54	0x54
ErrorEPC	undefined	restart PC	restart PC
Other registers	All other register bits are undefined.	All other register bits are undefined.	All other register bits are undefined.

Servicing

The Reset exception is serviced by:

- Initializing all processor registers, coprocessor registers, caches, an the memory system
- Performing diagnostic test
- Bootstrapping the operating system

6.4.4 Soft Reset Exception

Cause

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

The Soft Reset exception is not maskable.

Processing

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

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

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

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

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

Servicing

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

6.4.5 Nonmaskable Interrupt (NMI) Exception

Cause

The NMI exception occurs in response to the falling edge of the NMI* signal, or an external write to the *Int* (6) bit of the Interrupt register. The NMI exception is not maskable and occurs regardless of the settings of the *EXL*, *ERL*, and *IE* bits in the Status register.

Processing

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

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

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

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

Servicing

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

6.4.6 Address Error Exception

Cause

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

- Load or store a doubleword that is not aligned on a doubleword boundary
- Load, fetch, or store a word that is not aligned on a word boundar
- Load or store a halfword that is not aligned on a halfword boundary
- Reference the kernel address space from User or Supervisor mode
- Reference the supervisor address space from User mode

This exception is not maskable.

Processing

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

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

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

Servicing

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

6.4.7 TLB Exceptions

Three types of TLB exceptions can occur:

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

The following three sections describe these TLB exceptions.

6.4.7.1 TLB Refill exception

Cause

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

Processing

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

When this exception occurs, the BadVAddr, Context, XContext, and EntryHi registers hold the virtual address that failed address translation. The EntryHi register also contains the ASID from which the translation fault occurred. The Random register normally contains a valid location in which to place the replacement TLB entry. The contents of the EntryLo register are undefined. The EPC register contains the address of the instruction that caused the exception, unless this instruction is in a branch delay slot, in which case the EPC register contains the address of the preceding Branch instruction and the *BD* bit of the Cause register is set.

Servicing

To service this exception, the contents of the Context or XContext register are used as a virtual address to fetch memory locations containing the physical page frame and access control bits for a pair of TLB entries. The two entries are placed into the EntryLo0/EntryLo1 register; the EntryHi and EntryLo registers are written into the TLB.

It is possible that the virtual address used to obtain the physical address and access control information is on a page that is not resident in the TLB. This condition is processed by allowing a TLB Refill exception in the TLB refill handler. This second exception goes to the common exception vector because the *EXL* bit of the Status register is set.

6.4.7.2 TLB Invalid exception

Cause

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

Processing

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

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

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

Servicing

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

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

After servicing the cause of a TLB Invalid exception, the TLB entry is located with TLBP (TLB Probe) and replaced by an entry with that entry's *Valid* bit set.

6.4.7.3 TLB Modified exception

Cause

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

Processing

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

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

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

Servicing

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

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

6.4.8 Bus Error Exception

Cause

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

The Bus error exception occurs asynchronously because of the processor nonblock cache structure. The Bus Error exception resulting must be reported using the general interrupt mechanism.

Processing

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

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

Servicing

The EPC register may not have the instruction which caused the Bus error because of the nonblocking cache structure. Therefore, it is very difficult to trace the error address. The Bus error should be used for a fatal error and the system should be rebooted. The Bus error is not recoverable.

6.4.9 Integer Overflow Exception

Cause

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

Processing

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

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

Servicing

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

6.4.10 Trap Exception

Cause

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

Processing

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

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

Servicing

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

6.4.11 System Call Exception

Cause

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

Processing

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

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

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

Servicing

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

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

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

6.4.12 Breakpoint Exception

Cause

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

Processing

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

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

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

Servicing

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

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

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

6.4.13 Reserved Instruction Exception

Cause

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

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

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

This exception is not maskable.

Processing

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

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

Servicing

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

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

This exception is not maskable.

Processing

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

Servicing

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

- If the process is entitled access to the coprocessor, the coprocessor is marked usable and the corresponding user state is restored to the coprocessor.
- If the process is entitled access to the coprocessor, but th coprocessor does not exist or has failed, interpretation of the coprocessor instruction is possible.
- If the *BD* bit is set in the Cause register, the Branch instruction must be interpreted; then the coprocessor instruction can be emulated an execution resumed with the EPC register advanced past th coprocessor instruction
- If the process is not entitled access to the coprocessor, the proces executing at the time receives an illegal instruction/privileged instruction fault signal. This error is usually fatal

6.4.15 Floating-Point Exception

Cause

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

Processing

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

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

Servicing

This exception is cleared by clearing the appropriate bit in the Floating-Point Control/Status register.

For an Unimplemented Operation exception, the kernel should emulate the instruction; for other exceptions, the kernel should pass the exception to the user program that caused the exception.

6.4.16 Watch Exception

Cause

The Watch exception occurs when a Load or Store instruction references the physical address specified in the WatchLo and WatchHi system control coprocessor registers. The WatchLo register also specifies whether Load, Store, both, or neither initiated this exception. The CACHE instruction never causes a Watch exception. The exception is postponed while the *EXL* bit is set in the Status register. This exception is maskable only by setting *EXL* in the Status register.

Processing

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

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

Servicing

This exception is intended as a debugging aid. Typically, the exception handler will transfer control to a debugger, allowing the user to examine the situation. To continue, the Watch exception must be disabled for the execution of the faulting instruction and then re-enabled. Execution of the faulting instruction may be accomplished by interpretation or by setting a breakpoint.

6.4.17 Interrupt Exception

Cause

The Interrupt exception occurs when one of the eight interrupt conditions is asserted. The significance of these interrupts is dependent upon the specific system implementation.

Each of the eight interrupts can be masked by clearing the corresponding bit in the *Int-Mask* field of the Status register and all of the eight interrupts can be masked at once by clearing the *IE* bit of the Status register.

Processing

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

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

Servicing

If the interrupt is caused by one of the two software-generated exceptions (SW1 or SW0), the interrupt condition is cleared by setting the corresponding Cause register bit to 0.

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

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

6.5 Exception Handling and Servicing Flowcharts

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

- General exceptions and their exception handler
- TLB/XTLB Miss exceptions and their exception handle
- The Cache Error exception and its handler
- Reset, Soft Reset, and NMI exceptions and a guideline to their handler

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

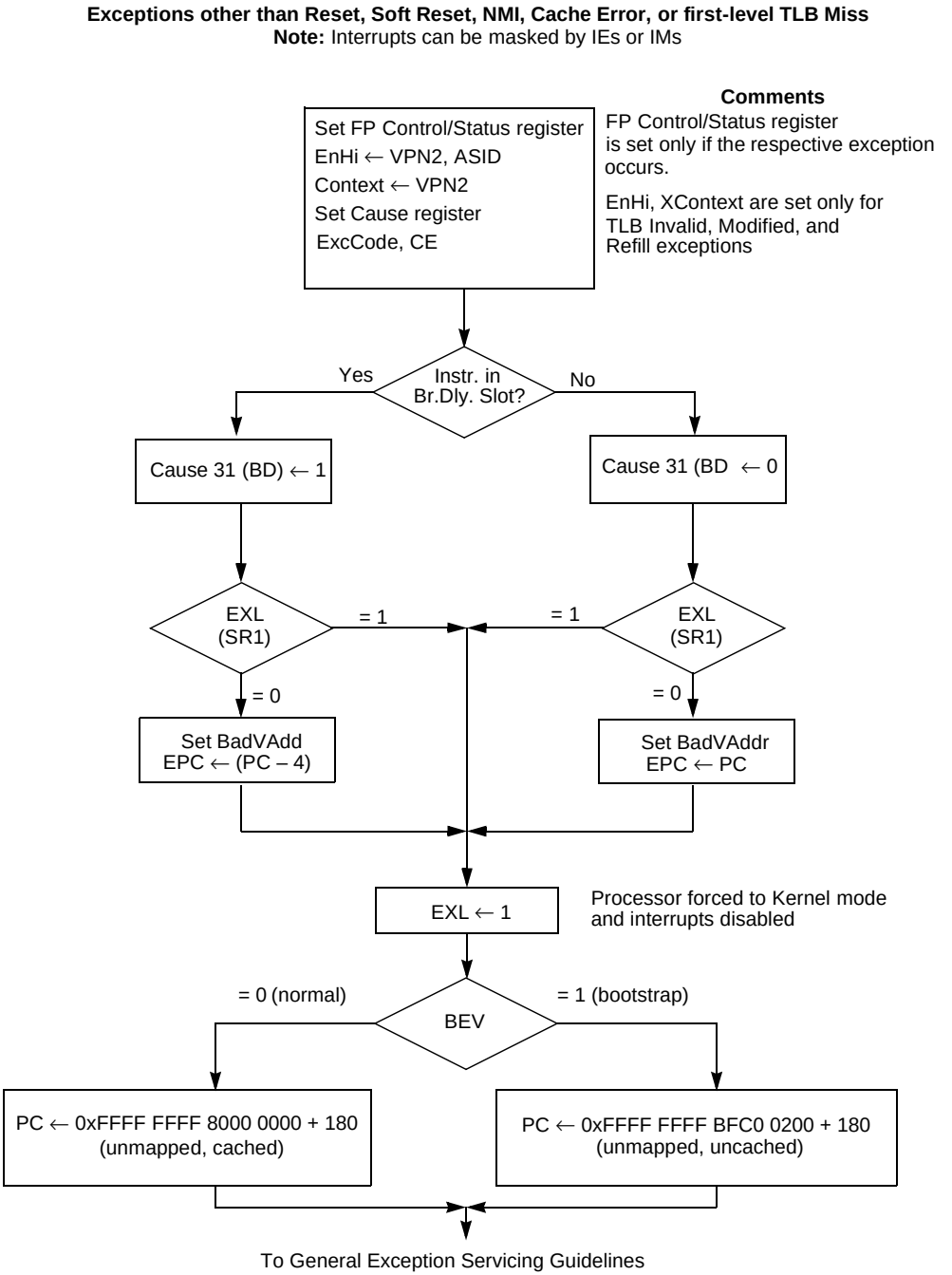


Figure 6-19 General Exception Handler (HW)

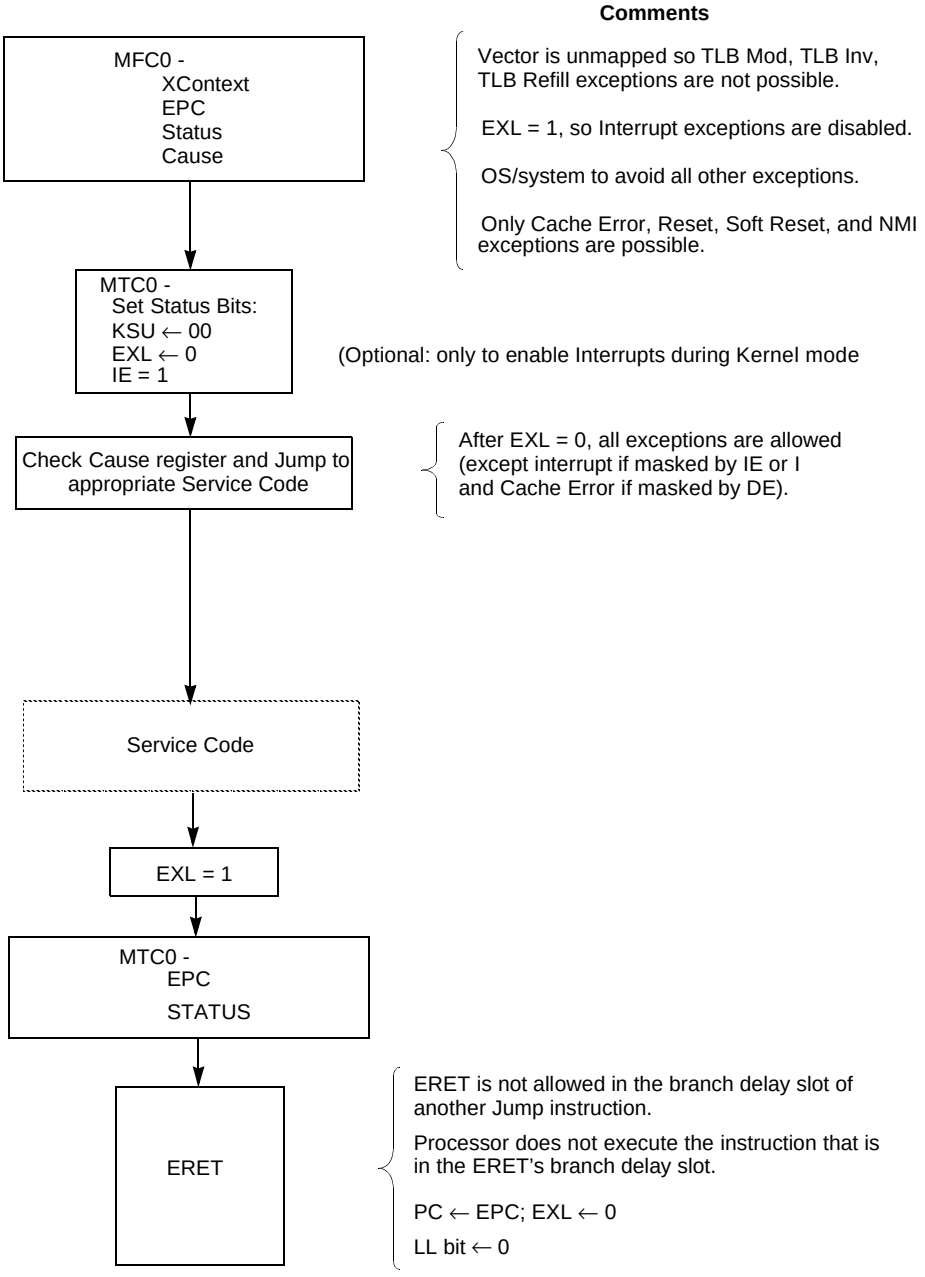


Figure 6-20 General Exception Servicing Guidelines (SW)

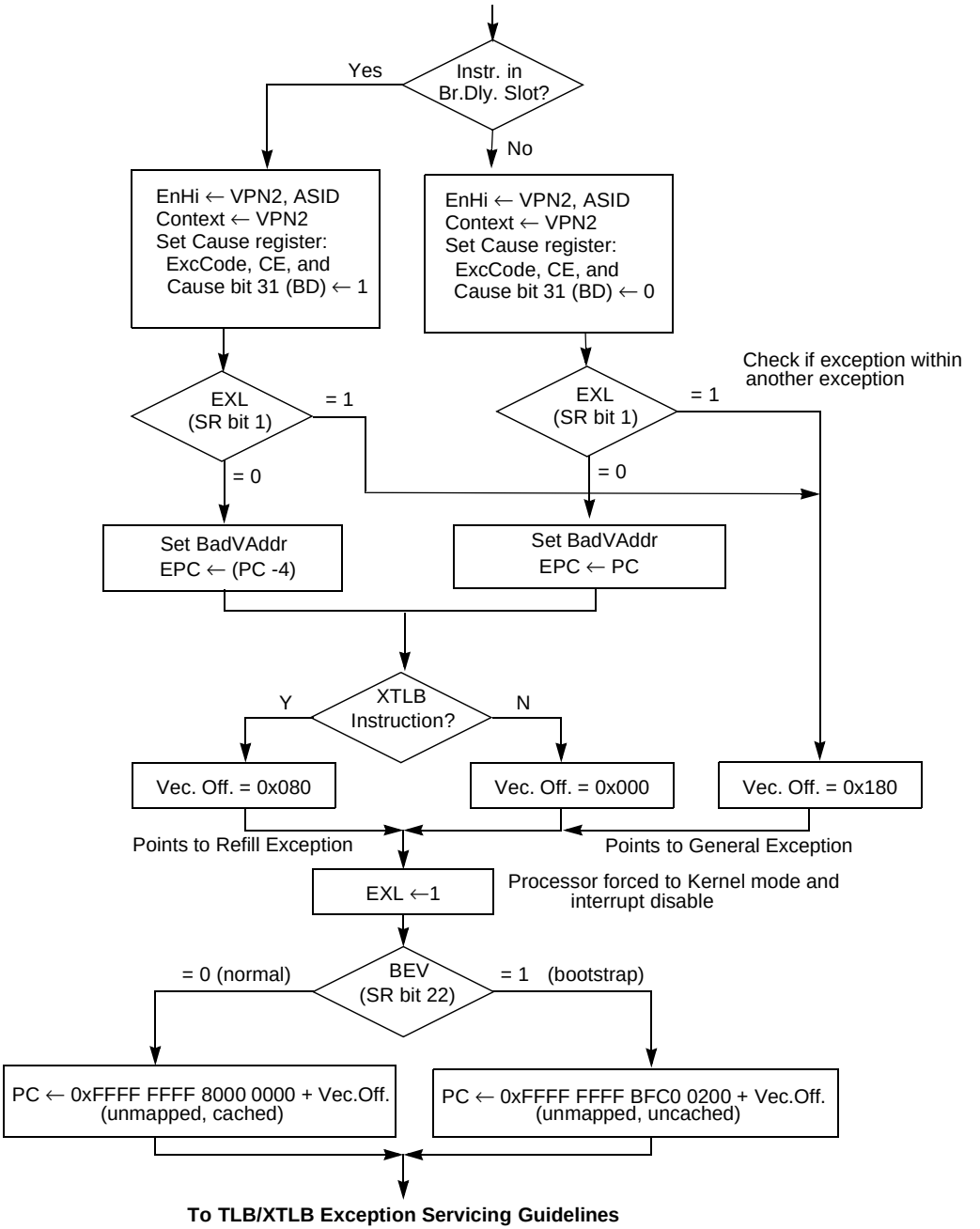


Figure 6-21 TLB/XTLB Miss Exception Handler (HW)

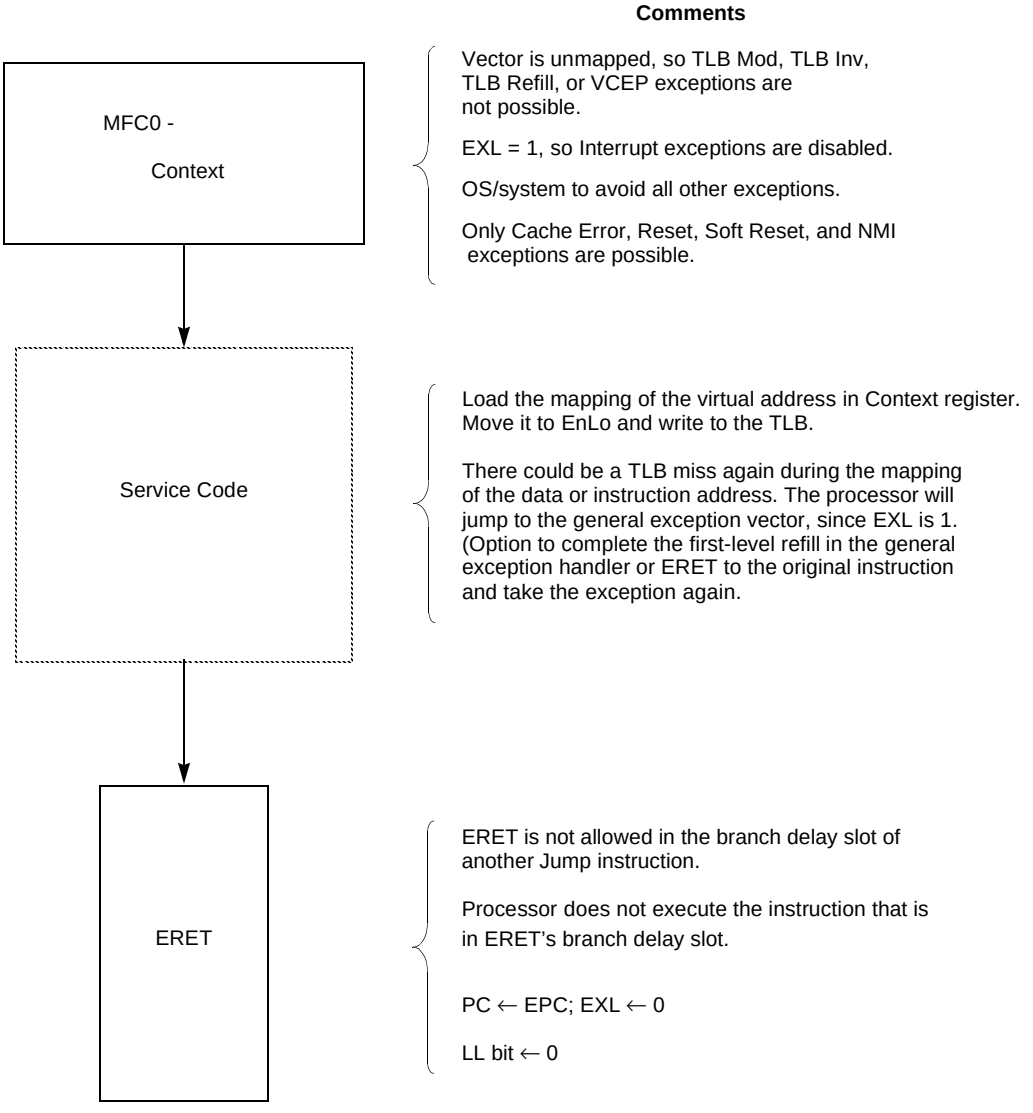


Figure 6-22 TLB/XTLB Exception Servicing Guidelines (SW)

Figure 6-23 Cache Error Exception Handling (HW) and Servicing Guidelines

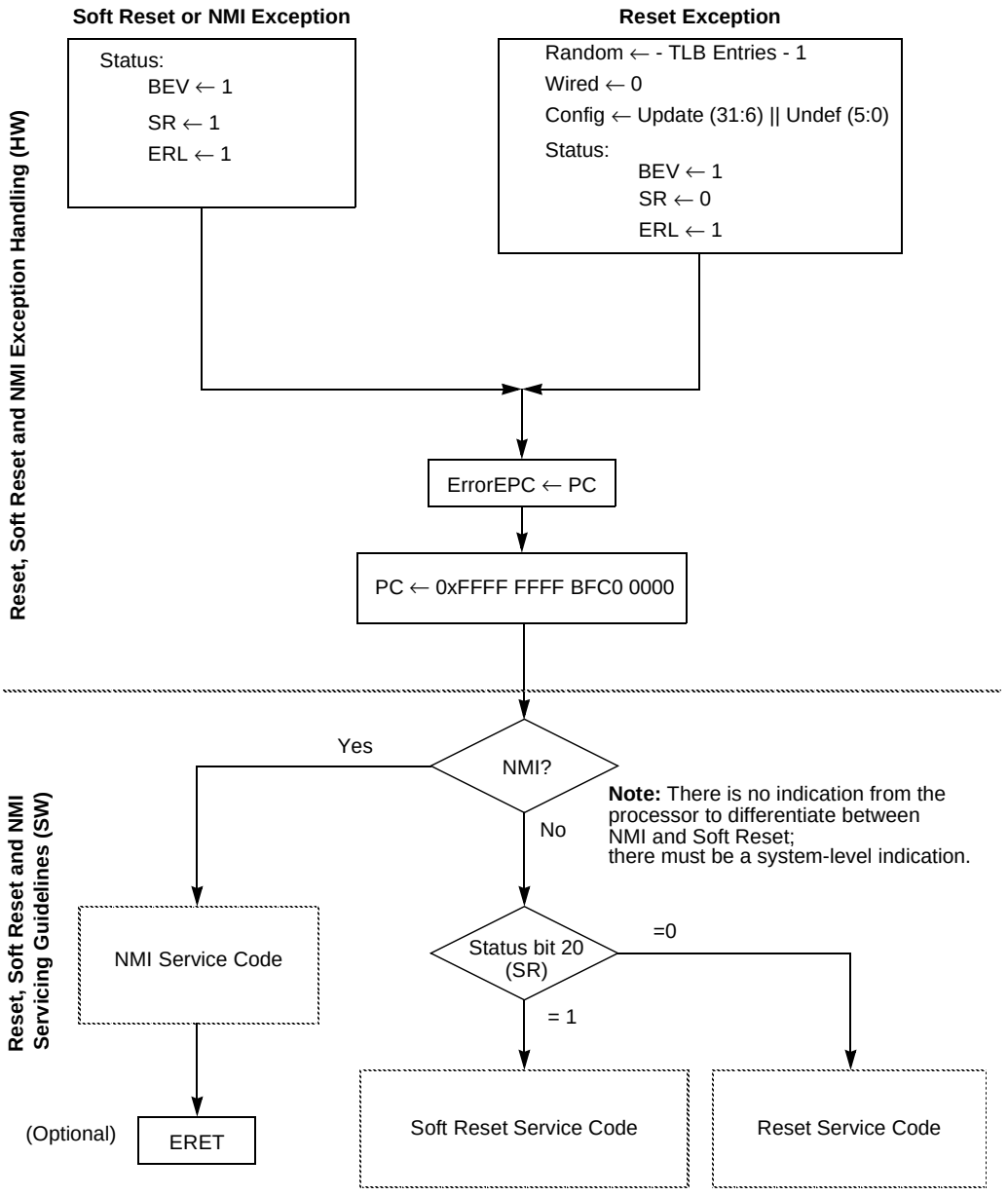


Figure 6-24 Reset, Soft Reset, and NMI Exception Handling

6.6 Interrupts

The VR5432 processor supports the following interrupts: five hardware interrupts, one internal timer interrupt, two software interrupts, and one nonmaskable interrupt. The processor takes an exception on any interrupt. Note that there is no priority mechanism specified for the processor among all these interrupts. This section describes the five hardware and single nonmaskable interrupts.

6.6.1 Hardware Interrupts

The five CPU hardware interrupts can be caused by an external write request to the VR5432 or through dedicated interrupt pins. These pins are latched into an internal register by the rising edge of SysClock.

6.6.2 Nonmaskable Interrupt (NMI)

The nonmaskable interrupt is caused by an external write request to the VR5432 or by the dedicated NMI* signal on the VR5432. This signal is latched into an internal register by the rising edge of SysClock.

6.6.3 Asserting Interrupts

External write requests to the CPU are accepted based on an internal address map of the processor. When *SysAD* (6:4) = 000_2, an external write request to an architecturally transparent register called the Interrupt register occurs. This register is available for external write cycles, but not for external read cycles.

During a data cycle, *SysAD* (22) and *SysAD* (20:16) are the write enable signals for the six individual Interrupt register bits and *SysAD* (6) with *SysAD* (4:0) are the values to be written into these bits. This allows any subset of the Interrupt register to be set or cleared with a single write request. Figure 6-25 shows the mechanics of an external write request to the Interrupt register.

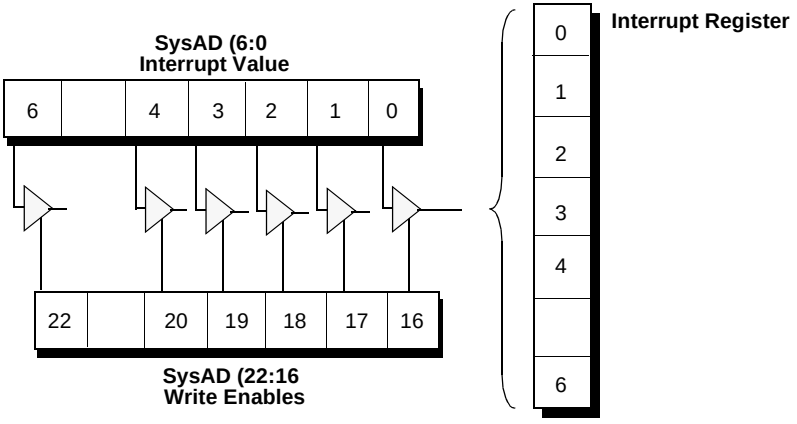


Figure 6-25 Interrupt Register Bits and Enables

Figure 6-26 shows how the VR5432 interrupts are readable through the Cause register.

- Bit 5 of the Interrupt register is tied to the Timer Interrupt signal. Th result is directly readable as bit 15 of the Cause register.
- Bits 4:0 of the Interrupt register are bitwise ORed with the current value of interrupt pins Int* (4:0). The result is directly readable as bits 14:10 of the Cause register.

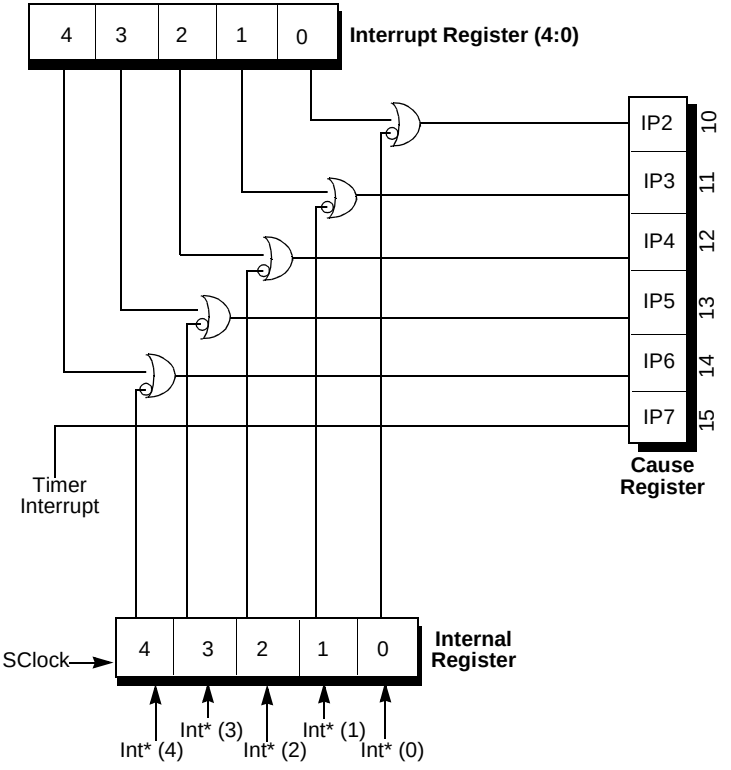


Figure 6-26 VR5432 Interrupt Signals

Figure 6-27 shows the internal derivation of the NMI* signal for the VR5432 processor.

The NMI* signal is latched by the rising edge of SysClock. Bit 6 of the Interrupt register is then ORed with the inverted value of NMI* to form the nonmaskable interrupt. Only the falling edge of the latched signal will cause the NMI.

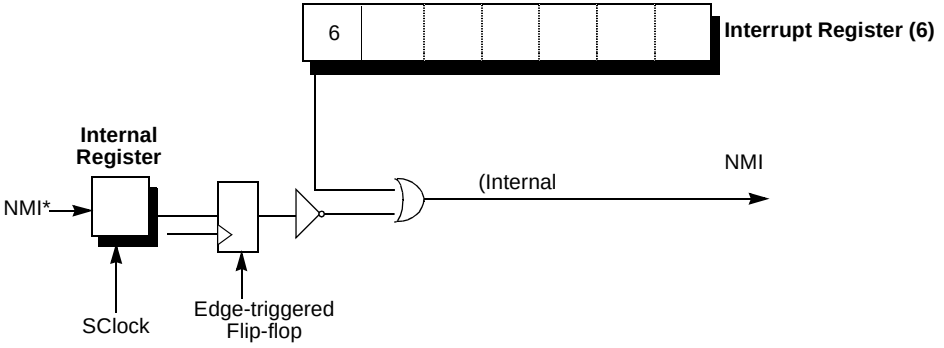


Figure 6-27 VR5432 Nonmaskable Interrupt Signal

Figure 6-28 shows the masking of the VR5432 interrupt signal.

- Cause register bits 15:8 (*IP7–IP0*) are AND-ORed with Status register interrupt mask bits 15:8 *IM7–IM0)* to mask individual interrupts.
- Status register bit 0 is a global Interrupt Enable (*IE*). It is ANDe with the output of the AND-OR logic to produce th VR5432 interrupt signal.

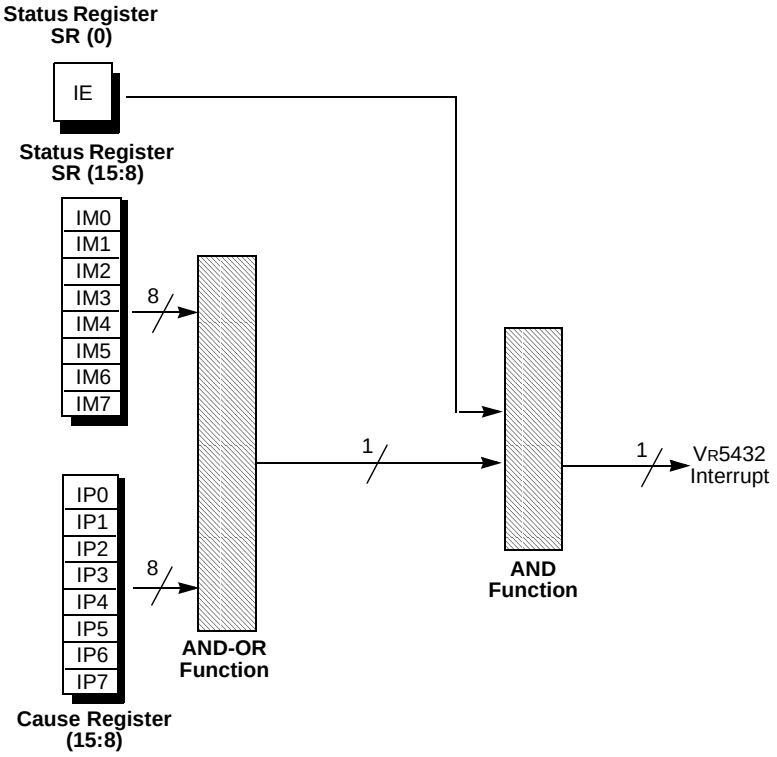


Figure 6-28 Masking of VR5432 Interrupts

Floating-Point Unit

7

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

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

7.1 Overview

Architecturally, the FPU is a coprocessor for the CPU (it is assigned coprocessor label CP1) and extends the CPU instruction set to perform arithmetic operations on floating-point values. However, it is implemented using the same data paths used to process integer instructions. This allows integer and floating-point instructions to be assigned to either data path, maximizing the efficiency of the dual-issue superscalar pipeline.

7.2 FPU Features

This section briefly describes the architectural model, the Load and Store instruction set, and the coprocessor interface to the FPU. A more detailed description is given in the sections that follow.

- **Full 64-bit operation**. When the *FR* bit in the CPU Status register equals 0, the FPU is in 32-bit mode and contains 32 32-bit registers that hold single- or, when used in pairs, double-precision values. When the *FR* bit in the CPU Status register equals 1, the FPU is in 64-bit mode and the registers are expanded to 64 bits wide. In this mode, each register can hold single- or double-precision values. The FPU also includes a 32-bit Control/Status register that provides access to all IEEE Standard exception handling capabilities.
- **Load and Store instruction set**. Like the CPU, the FPU uses a load and store-oriented instruction set, with single-cycle load and stor operations

7.3 FPU Programming Model

This section describes the FPU registers and their data organization. The FPU registers include Floating-point General-purpose registers (FGRs) and two control registers: Control/Status and Implementation/Revision.

7.4 Floating-Point General-Purpose Registers

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

- As 32 general-purpose registers (32 FGRs), each of which is 32 bits wide when the *FR* bit in the CPU Status register equals 0; or as 32 general-purpose registers (32 FGRs), each of which is 64 bits wide when *FR* equals 1. The CPU accesses these registers through Move, Load, and Store instructions.
- As 16 floating-point registers (see the next section for a description of FPRs), each of which is 64 bits wide, when the *FR* bit in the CPU Status register equals 0. The FPRs hold values in either single- o double-precision floating-point format. Each FPR corresponds t adjacently numbered FGRs, as shown in Figure 7-1. Th *FR* bit ca only be 0 when executing the MIPS I, II, or III instruction set Executing a MIPS IV instruction with the *FR* bit equal to 0 results in undefined behavior.
- As 32 floating-point registers (see the next section for a description of FPRs), each of which is 64 bits wide, when the *FR* bit in the CPU Status register equals 1. The FPRs hold values in either single- or double-precision floating-point format. Each FPR corresponds to an FGR, as shown in Figure 7-1.

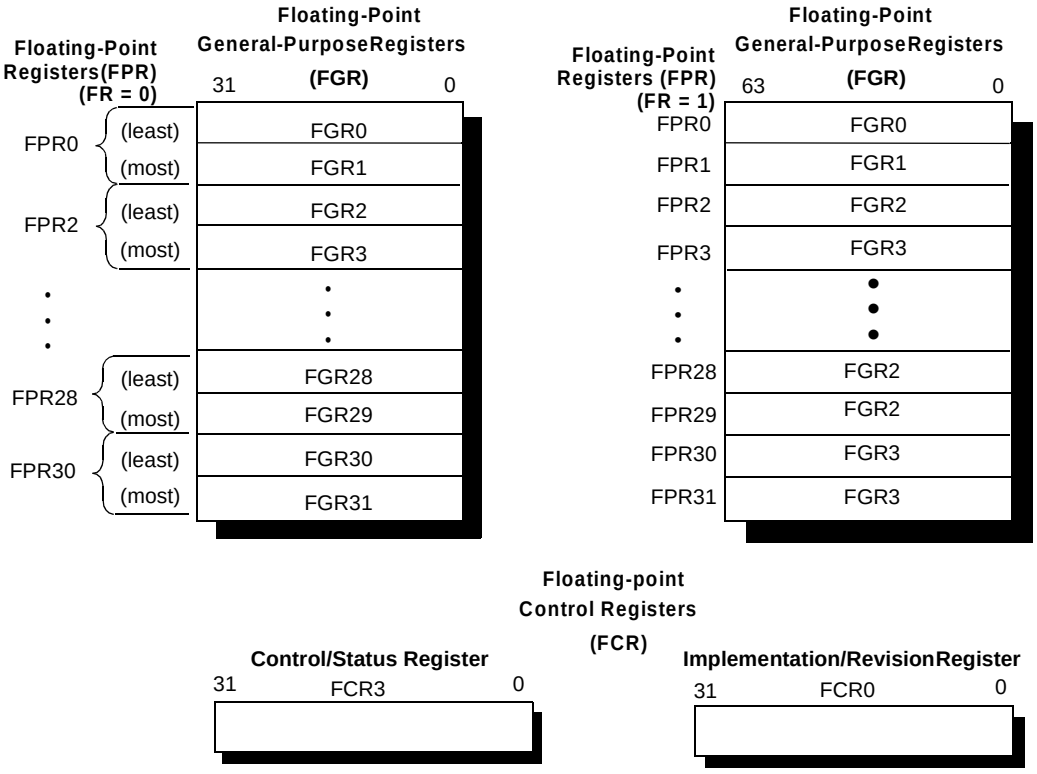


Figure 7-1 FPU Registers

7.5 Floating-Point Registers

The FPU provides:

- 16 floating-point registers (FPRs) when the *FR* bit in the Status register equals 0, or
- 32 floating-point registers (FPRs) when the *FR* bit in the Status register equals 1

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

The FPRs hold values in either single- or double-precision floating-point format. If the *FR* bit equals 0, only even numbers (the *least* register, as shown in Figure 7-1) can be used to address FPRs. When the *FR* bit is set to a 1, all FPR register numbers are valid.

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

7.6 Floating-Point Control Registers

The architecture reserves for the FPU 32 control registers (FCRs) that can only be accessed by move operations. Two such registers are implemented. The FCRs are described below:

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

Table 7-1 lists the assignments of the FCRs.

Table 7-1 Floating-Point Control Register Assignments

FCR Number	Use
FCR0	Coprocessor Implementation and Revision register
FCR1 to FCR30	Reserved
FCR31	*Rounding Mode*, *Cause*, *Trap*, *Enable*, and *Flag*

7.6.1 Implementation and Revision Register (FCR0)

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

Figure 7-2 shows the layout of the register; Table 7-2 describes the Implementation and Revision register (FCR0) fields.

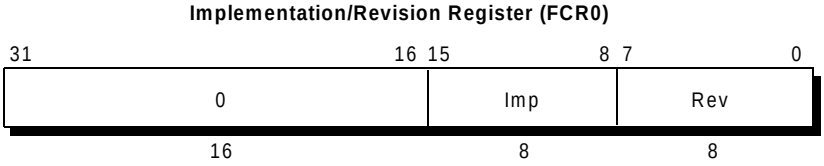


Figure 7-2 Implementation/Revision Register

Table 7-2 FCR0 Fields

Field	Description
Imp	Implementation number (0x54)
Rev	Revision number in the form of *y.x*
0	Reserved. Must be written as 0; returns 0 when read.

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

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

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

7.6.2 Control/Status Register (FCR31)

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

Figure 7-3 shows the format of the Control/Status register; Table 7-3 describes the Control/Status register fields. Figure 7-4 shows the Control/Status register *Cause*, *Flag*, and *Enable* fields.

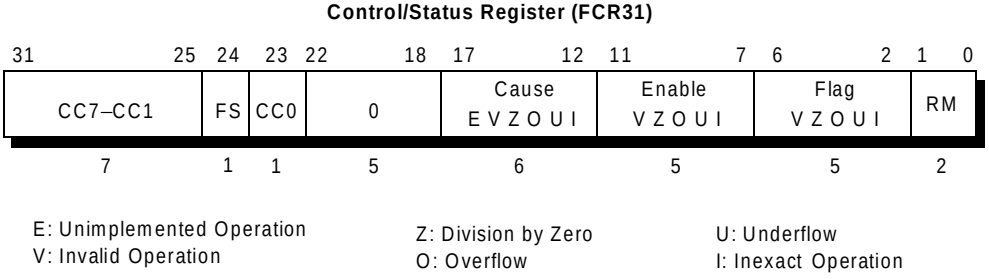


Figure 7-3 FP Control/Status Register Bit Assignments

Table 7-3 Control/Status Register Fields

Field	Description
CC7:CC1	*Condition* bits 7:1. See description of Control/Status register *Condition* bit.
FS	When set, denormalized results are flushed to 0 instead of causing an Unimplemented Operation exception. On the VR5432, even if the *FS* bit is set, if a MADD, MSUB, NMADD, or NMSUB instruction encounters a denormalized result during the multiply portion of the calculation, an Unimplemented Operation exception is always taken.
CC0	*Condition* bit 0. See description of Control/Status register *Condition* bit.
Cause	*Cause* bits. See description of Control/Status register *Cause*, *Flag*, and *Enable* bits.
Enable	*Enable* bits. See description of Control/Status register *Cause*, *Flag*, and *Enable* bits.
Flag	*Flag* bits. See description of Control/Status register *Cause*, *Flag*, and *Enable* bits.
RM	*Rounding Mode* bits. See description of Control/Status register *Rounding Mode Control* bits.

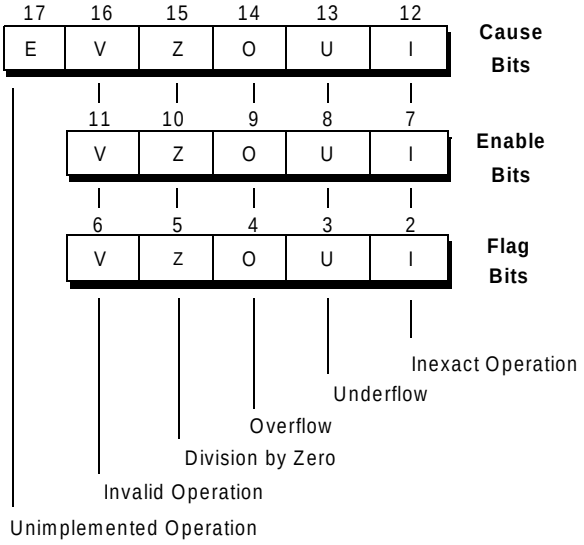


Figure 7-4 Control/Status Register Cause, Flag, and Enable Fields

7.6.2.1 Accessing the Control/Status register

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

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

7.6.2.2 IEEE Standard 754

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

7.6.2.3 Control/Status register FS bit

When the *FS* bit is set, denormalized results are flushed to 0 instead of causing an Unimplemented Operation exception.

However, for MADD.fmt, NMADD.fmt, MSUB.fmt, and NMSUB.fmt instructions, the VR5432 will always take an Unimplemented Operation exception if the intermediate multiply result is a denormalized value, regardless of the value of the *FS* bit.

7.6.2.4 Control/Status register Condition bit

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

7.6.2.5 Control/Status register Cause, Flag, and Enable fields

Figure 7-4 illustrates the *Cause*, *Flag*, and *Enable* fields of the Control/Status register.

Cause bits

Bits 17:12 in the Control/Status register contain *Cause* bits, as shown in Figure 7-4, that reflect the results of the most recently executed instruction. The *Cause* bits are a logical extension of the CP0 Cause register; they identify the exceptions raised by the last floating-point operation and raise an interrupt or exception if the corresponding *Enable* bit is set. If more than one exception occurs on a single instruction, each appropriate bit is set.

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

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

Enable bits

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

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

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

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

Flag bits

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

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

7.6.2.6 Control/Status register Rounding mode Control bits

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

As shown in Table 7-4, these bits specify the Rounding mode that the FPU uses for all floating-point operations.

Table 7-4 Rounding Mode Bit Decoding

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

7.7 Floating-Point Formats

The FPU performs both 32-bit (single-precision) and 64-bit (double-precision) IEEE-standard floating-point operations. The 32-bit single-precision format has a 24-bit signed-magnitude fraction field (*f* + *s*) and an 8-bit exponent (*e*), as shown in Figure 7-5.

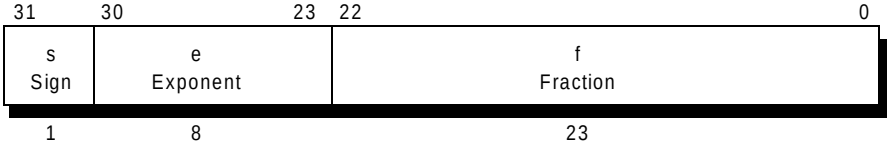


Figure 7-5 Single-Precision Floating-Point Format

The 64-bit double-precision format has a 53-bit signed-magnitude fraction field (*f* + *s)* and an 11-bit exponent, as shown in Figure 7-6.

Figure 7-6 Double-Precision Floating-Point Format

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

- Sign field, *s*
- Biased exponent, $e = E + bias$
- Fraction $f = .b_1b_2....b_{p-1}$

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

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

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

For single- and double-precision formats, the value of a number, *v*, is determined by the equations shown in Table 7-5.

Table 7-5 Calculating Values in Single- and Double-Precision Formats

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

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

Figure 7-6 defines the values for the format parameters; minimum and maximum floating-point values are given in Table 7-7.

Table 7-6 Floating-Point Format Parameter Values

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

Table 7-7 Minimum and Maximum Floating-Point Values

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

7.8 Binary Fixed-Point Format

Binary fixed-point values are held in two's-complement format. Unsigned fixed-point values are not directly provided by the Floating-Point instruction set. Figure 7-7 illustrates binary fixed-point format; Table 7-8 lists the binary fixed-point format fields.

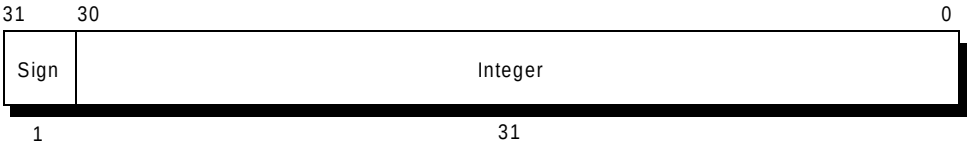


Figure 7-7 Binary Fixed-Point Format

Field assignments of the binary fixed-point format are:

Table 7-8 Binary Fixed-Point Format Fields

Field	Description
Sign	*Sign* bit
Integer	Integer value

7.9 Floating-Point Instruction Set Overview

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

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

In the instruction formats shown in Table 7-9 through Table 7-12, the *fmt* appended to the instruction opcode specifies the data format: *S* specifies single-precision binary floating-point, *D* specifies double-precision binary floating-point, *W* specifies 32-bit binary fixed-point, and *L* specifies 64-bit (long) binary fixed-point.

Table 7-9 FPU Instruction Summary: Load, Move, Store Instructions

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

Table 7-10 FPU Instruction Summary: Conversion Instructions

Opcode	Description
CVT.S.fmt	Floating-point Convert to Single FP
CVT.D.fmt	Floating-point Convert to Double FP
CVT.W.fmt	Floating-point Convert to Single Fixed-point
CVT.L.fmt	Floating-point Convert to Long Fixed-point
ROUND.W.fmt	Floating-point Round to Single Fixed-point
ROUND.L.fmt	Floating-point Round to Long Fixed-point
TRUNC.W.fmt	Floating-point Truncate to Single Fixed-point
TRUNC.L.fmt	Floating-point Truncate to Long Fixed-point
CEIL.W.fmt	Floating-point Ceiling to Single Fixed-point
CEIL.L.fmt	Floating-point Ceiling to Long Fixed-point
FLOOR.W.fmt	Floating-point Floor to Single Fixed-point
FLOOR.L.fmt	Floating-point Floor to Long Fixed-point

Table 7-11 FPU Instruction Summary: Computational Instructio n

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

Table 7-12 FPU Instruction Summary: Compare and Branch Instructions

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

7.9.1 Floating-Point Load, Store, and Move Instructions

This section discusses the manner in which the FPU uses the Load, Store, and Move instructions listed in Table 7-9.

7.9.1.1 Transfers between FPU and memory

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

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

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

7.9.1.2 Transfers between the FPU and CPU

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

- Move Word to FPU (Coprocessor 1 (MTC1)
- Move Word from FPU (Coprocessor 1 (MFC1)
- Doubleword Move to FPU (Coprocessor 1 (DMTC1)
- Doubleword Move from FPU (Coprocessor 1 (DMFC1)

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

7.9.1.3 Load delay and hardware interlocks

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

7.9.1.4 Data alignment

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

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

7.9.1.5 Byte-numbering order

Regardless of data byte-numbering order, the address specifies the byte that has the smallest byte address in the addressed field. For a big-endian system, it is the left-most byte; for a little-endian system, it is the right-most byte.

7.9.2 **Floating-Point Conversion Instructions**

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

7.9.2.1 Conversion from floating-point to 64-bit fixed-point or long integer

For operations relating to the conversion from floating-point to fixed-point or integer formats, the resulting range must be between -2^{52} to $2^{52} - 1$. This condition exists due to the 53-bit floating-point data path.

When the source value is not rounded to an integer outside the range of -2^{52} to $2^{52} - 1$ and the result cannot be expressed correctly, an Unimplemented Operation exception is taken. The result of the instruction is discarded.

The floating-point Unimplemented Operation exception condition applies to the following conversion instructions:

- CEIL.L.[S/D]
- CVT.L.[S/D]
- FLOOR.L.[S/D]
- ROUND.L.[S/D]
- TRUNC.L. [S/D]

7.9.3 Floating-Point Computational Instructions

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

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

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

7.9.3.1 Branch on FPU Condition instructions

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

7.9.3.2 Floating-point Compare operations

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

Table 7-13 lists the mnemonics for the Compare instruction conditions.

Table 7-13 Mnemonics and Definitions of Compare Instruction Conditions

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

7.10 FPU Instruction Pipeline Overview

The FPU provides an instruction pipeline that parallels the CPU instruction pipeline, utilizing the same five-stage architecture as the CPU.

7.10.1 Instruction Execution

FPU instructions execute using the CPU data path. FPU and CPU instructions share the same issue logic, data path, and pipeline stages. Single-cycle FPU instructions execute with the same pipeline sequence as single-cycle CPU instructions. A multicycle FPU instruction will iterate in the EX stage until it completes. Figure 7-8 illustrates a single-cycle and a multicycle FPU pipeline.

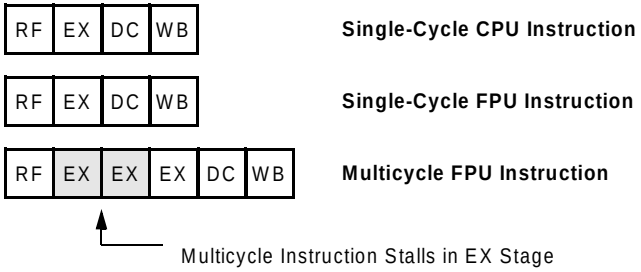


Figure 7-8 FPU Instruction Pipeline

7.10.2 Instruction Execution Cycle Time

Unlike the CPU, which executes almost all instructions in a single cycle, more time may be required to execute FPU instructions.

Table 7-14 gives the minimum latency, in processor pipeline cycles, of each floating-point operation for the currently implemented configurations. These latency calculations assume that the result of the operation is immediately used in a succeeding operation.

Table 7-14 Floating-Point Operation Latencies

Operation	Pipeline Cycles Latency/Repeat[1]		Operation	Pipeline Cycles Latency/Repeat[1]	
	S	D		S	D
LDC1	3/2[6]	3/2[6]	CEIL.L.fmt	6/5[4]	6/5[4]
LDXC1	3/2[6]	3/2[6]	CEIL.W.fmt	6/5	6/5
LWC1	3/2[6]	3/2[6]	CVT.D.fmt	2/1	**Note 2**
LWXC1	3/2[6]	3/2[6]	CVT.D.fmt (W/L)	6/5	6/5
PREF	3/2[6]	3/2[6]	CVT.L.fmt	6/5[3]	6/5[3]
PREFX	3/2[6]	3/2[6]	CVT.S.fmt	**Note 2**	3/2
SDC1	NA/1	NA/1	CVT.S.fmt (W/L)	6/5	6/5[3]
SDXC1	NA/2	NA/2	CVT.W.fmt	6/5	6/5
SWC1	NA/1	NA/1	FLOOR.L.fmt	6/5[4]	6/5[4]
SWXC1	NA/2	NA/2	FLOOR.W.fmt	6/5	6/5
CFC1	NA[5]	NA[5]	ROUND.L.fmt	6/5[4]	6/5[4]
CTC1	NA[5]	NA[5]	ROUND.W.fmt	6/5	6/5
DMFC1	2/1	2/1	TRUNC.L.fmt	6/5[4]	6/5[4]
DMTC1	2/1	2/1	TRUNC.W.fmt	6/5	6/5
MFC1	2/1	2/1	ABS.fmt	2/1	2/1
MOV.fmt	2/1	2/1	ADD.fmt	4/3	4/3
MOVF	2/1	2/1	DIV.fmt	31/30	59/58
MOVF.fmt	2/1	2/1	MADD.fmt	9/8	10/9
MOVN.fmt	2/1	2/1	MSUB.fmt	9/8	10/9
MOVT	2/1	2/1	MUL.fmt	5/4	6/5
MOVT.fmt	2/1	2/1	NEG.fmt	2/1	2/1
MOVZ.fmt	2/1	2/1	NMADD.fmt	9/8	10/9
MTC1	2/1	2/1	NMSUB.fmt	9/8	10/9
BC1F	NA/1	NA/1	RECIP.fmt	31/30	59/58
BC1FL	NA/1	NA/1	RSQRT.fmt	61/60	121/120
BC1T	NA/1	NA/1	SQRT.fmt	31/30	59/58

Table 7-14 Floating-Point Operation Latencies (continued)

Operation	Pipeline Cycles Latency/Repeat[1]		Operation	Pipeline Cycles Latency/Repeat[1]	
	S	D		S	D
BC1TL	NA/1	NA/1	SUB.fmt	4/3	4/3
C.cond.fmt	2/1	2/1			

Notes:

1. This is the throughput of a single data path. Except for memory operations, all instructions can be dual issued.
2. These operations are illegal.
3. Trap on greater than 52 bits of significance.
4. Trap on greater than 53 bits of significance.
5. Serializing instruction.
6. Average best case.

7.10.3 Instruction Issuing Constraints with Multicycle Instructions

While a multicycle instruction is issued and stalls in the EX stage, instruction issuing will continue for one to three more instructions, depending on instruction resource requirements. At that point, issue will stall, but all instructions issued will continue to execute. When the multicycle instruction reaches the DC stage, instruction issuing resumes. This behavior allows up to two multicycle instructions in a four-instruction window to be executing simultaneously, effectively hiding the execution latency of one instruction under the other.

Floating-Point Exceptions

8

This chapter describes floating-point exceptions, including FPU exception types, exception trap processing, exception flags, saving and restoring state when handling an exception, and trap handlers for IEEE Standard 754 exceptions.

A floating-point exception occurs whenever the FPU cannot handle either the operands or the results of a floating-point operation in its normal way. The FPU responds by generating an exception to initiate a software trap or by setting a status flag.

8.1 Exception Types

The FP Control/Status register described in Section 7.6.2 on page 155 contains an *Enable* bit for each exception type. Exception *Enable* bits determine whether an exception will cause the FPU to initiate a trap or set a status flag.

- If a trap is taken, the FPU remains in the state found at the beginnin of the operation and a software exception handling routine executes.
- If no trap is taken, an appropriate value is written into the FPU destination register and execution continues.

The FPU supports the five IEEE Standard 754 exceptions:

- Inexact Operation (
- Underflow (U)
- Overflow (O)
- Division by Zero (Z)
- Invalid Operation (V)

Cause, *Enable*, and *Flag* bits (status flags) are used.

The FPU adds a sixth exception type, Unimplemented Operation (E), to use when the FPU cannot implement the standard MIPS floating-point architecture, including cases in which the FPU cannot determine the correct exception behavior. This exception indicates the use of a software implementation. The Unimplemented Operation exception has no *Enable* or *Flag* bit; whenever this exception occurs, an Unimplemented Operation exception trap is taken (if the FPU interrupt input to the CPU is enabled).

Figure 8-1 illustrates the Control/Status register bits that support exceptions.

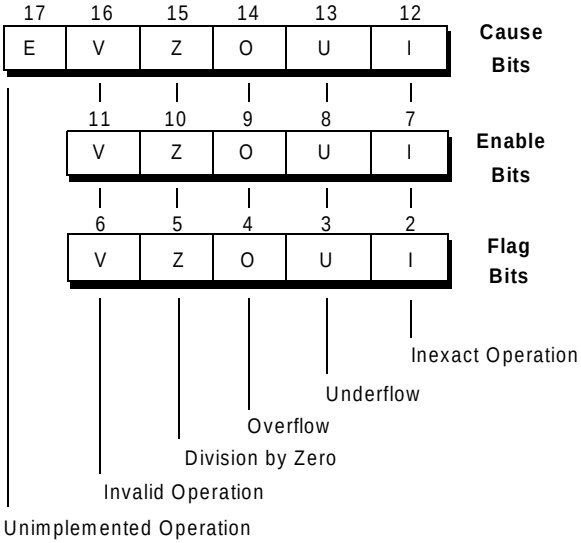


Figure 8-1 Control/Status Register Exception/Flag/Trap/Enable Bits

Each of the five IEEE Standard-754 exceptions (V, Z, O, U, I) is associated with a trap under user control and is enabled by setting one of the five *Enable* bits. When an exception occurs, the corresponding *Cause* bit is set. If the corresponding *Enable* bit is not set, the *Flag* bit is also set. If the corresponding *Enable* bit is set, the *Flag* bit is not set and the FPU generates an interrupt to the CPU. Subsequent exception processing allows a trap to be taken.

8.2 Exception Trap Processing

When a floating-point exception trap is taken, the Cause register indicates that the floating-point coprocessor is the cause of the exception trap. The *Floating-Point Exception (FPE)* code is used, and the Cause bits of the Floating-Point Control/ Status register indicate the reason for the Floating-Point exception. These bits are, in effect, an extension of the system coprocessor Cause register.

8.3 Flags

A *Flag* bit is provided for each IEEE exception. This *Flag* bit is set to a 1 on the assertion of its corresponding exception, with no corresponding exception trap signaled.

The *Flag* bit is reset by writing a new value into the Status register. Flags can be saved and restored by software individually or as a group.

When no exception trap is signaled, the floating-point coprocessor takes a default action, providing a substitute value for the exception-causing result of the floating-point operation. The particular default action taken depends upon the type of exception. Table 8-1 lists the default action taken by the FPU for each of the IEEE exceptions.

Table 8-1 Default FPU Exception Acti o n

Field	Description	Rounding Mode	Default Action
I	Inexact Operation exception	Any	Supplies a rounded result
U	Underflow exception	RN	Modifies underflow values to 0 with the sign of the intermediate result
		RZ	Modifies underflow values to 0 with the sign of the intermediate result
		RP	Modifies positive underflows to the format's smallest positive finite number; modifies negative underflows to 0
		RM	Modifies negative underflows to the format's smallest negative finite number; modifies positive underflows to 0
O	Overflow exception	RN	Modifies overflow values to ∞ with the sign of the intermediate result
		RZ	Modifies overflow values to the format's largest finite number with the sign of the intermediate result
		RP	Modifies negative overflows to the format's most negative finite number; modifies positive overflows to + ∞
		RM	Modifies positive overflows to the format's largest finite number; modifies negative overflows to – ∞
Z	Division by Zero exception	Any	Supplies a properly signed ∞
V	Invalid Operation exception	Any	Supplies a quiet Not a Number (NaN)

Table 8-2 lists the exception-causing situations and contrasts the behavior of the FPU with the requirements of IEEE Standard 754.

Table 8-2 FPU Exception Conditions

FPA Internal Result	IEEE Standard 754	Trap Enable	Trap Disable	Notes
Inexact operation	I	I	I	Loss of accuracy
Exponent overflow	O, I[1]	O, I	O, I	Normalized exponent > E_{max}
Division by zero	Z	Z	Z	Zero is (exponent = E_{min} – 1, mantissa = 0)
Overflow on convert	V	E	E	Source out of integer range
Signaling NaN source	V	V	V	
Invalid operation	V	V	V	0/0, etc.
Exponent underflow	U	E	E	Normalized exponent < E_{min}
Denormalized or QNaN	None	E	E	Denormalized is (exponent = E_{min} – 1 and mantissa <> 0)

Note:

1. IEEE Standard 754 specifies an Inexact Operation exception on overflow only if the overflow trap is disabled.

8.4 FPU Exceptions

The following sections describe the conditions that cause the FPU to generate each of its exceptions and details the FPU response to each exception condition.

8.4.1 Inexact Operation Exception (I)

The FPU generates the Inexact Operation exception if one of the following occurs:

- The rounded result of an operation is not exact
- The rounded result of an operation overflows
- The rounded result of an operation underflows and both th *Underflo* and *Inexact Enable* bits are not set and th *FS* bit is set

The FPU usually examines the operands of floating-point operations before execution actually begins, to determine (based on the exponent values of the operands) if the operation can *possibly* cause an exception. If there is a possibility of an instruction causing an exception trap, the FPU uses a coprocessor stall to execute the instruction.

It is impossible, however, for the FPU to predetermine if an instruction will produce an inexact result. If Inexact Operation exception traps are enabled, the FPU uses the coprocessor stall mechanism to execute all floating-point operations that require more than one cycle. Since this mode of execution can affect performance, Inexact Operation exception traps should be enabled only when necessary.

Trap-enabled results. If Inexact Operation exception traps are enabled, the result register is not modified and the source registers are preserved.

Trap-disabled results. The rounded or overflowed result is delivered to the destination register if no other software trap occurs.

8.4.2 Invalid Operation Exception (V)

The Invalid Operation exception is signaled if one or both of the operands are invalid for an implemented operation. When the exception occurs without a trap, the MIPS ISA defines the result as a quiet Not a Number (NaN). The invalid operations are:

- Addition or subtraction: magnitude subtraction of infinities, such as: $(+\infty) + (-\infty)$ or $(-\infty) - (-\infty)$
- Multiplication: $\times \infty$, with any signs
- Division: 0/0, or ∞/∞, with any signs
- Comparison of predicates involving < or > without **?**, when the operands are unordered
- Comparison or a Convert from Floating-Point Operation on a signaling Na
- Any arithmetic operation on a signaling NaN. A move (MOV) operation is not considered an arithmetic operation, but absolut value (ABS) and negate (NEG) are considered arithmetic operations and cause this exception if one or both operands is a signaling NaN.
- Square root: $\sqrt{x}$, where x is less than zer

Software can simulate the Invalid Operation exception for other operations that are invalid for the given source operands. Examples of these operations include IEEE Standard 754-specified functions implemented in software, such as Remainder: x REM y, where y is 0 or x is infinite; conversion of a floating-point number to a decimal format whose value causes an overflow, is infinity, or is NaN; and transcendental functions, such as ln (–5) or cos – 1(3).

Trap-enabled results. The original operand values are undisturbed.

Trap-disabled results. A quiet NaN is delivered to the destination register if no other software trap occurs.

8.4.3 Division by Zero Exception (Z)

The Division by Zero exception is signaled on an implemented divide operation if the divisor is zero and the dividend is a finite nonzero number. Software can simulate this exception for other operations that produce a signed infinity, such as ln(0), sec(π/2), csc(0), or $0^{-1.}$

Trap-enabled results. The result register is not modified, and the source registers are preserved.

Trap-disabled results. The result, when no trap occurs, is a correctly signed infinity.

8.4.4 Overflow Exception (O)

The Overflow exception is signaled when the magnitude of the rounded floating-point result, with an unbounded exponent range, is larger than the largest finite number of the destination format. (This exception also sets the Inexact Operation exception and *Flag* bits.)

Trap-enabled results. The result register is not modified, and the source registers are preserved.

Trap-disabled results. The result, when no trap occurs, is determined by the rounding mode and the sign of the intermediate result (as listed in Table 8-1).

8.4.5 Underflow Exception (U)

Two related events contribute to the Underflow exception:

- Creation of a small nonzero result between $\pm 2^{Emi}$, which can cause some later exception because it is so smal
- Extraordinary loss of accuracy during the approximation of such small numbers by a denormalized number

IEEE Standard 754 allows a variety of ways to detect these events, but requires that they be detected the same way for all operations.

These types of exceptions can be detected by one of the following methods:

- After rounding (when a nonzero result, computed as though th exponent range were unbounded, would lie strictly between $\pm 2^{Emin}$)
- Before rounding (when a nonzero result, computed as though the exponent range and the precision were unbounded, would lie strictly between $\pm 2^{Emi}$)

The MIPS architecture requires that these types of exceptions be detected after rounding.

Loss of accuracy can be detected by one of the following methods:

- Denormalization loss (when the delivered result differs from what would have been computed if the exponent range were unbounded)
- Inexact result (when the delivered result differs from what would have been computed if the exponent range and precision were bot unbounded)

The MIPS architecture requires that loss of accuracy be detected as an inexact result.

Trap-enabled results. If Underflow or Inexact traps are enabled, or if the *FS* bit is not set, then an Unimplemented Operation exception (E) is generated, and the result register is not modified.

Trap-disabled results. If Underflow and Inexact traps are not enabled and the *FS* bit is set, the result is determined by the rounding mode and the sign of the intermediate result (as listed in Table 8-1).

8.4.6 Unimplemented Operation Exception (E)

Any attempt to execute an instruction with an operation code or format code that has been reserved for future definition sets the *Unimplemented* bit in the *Cause* field in the FPU Control/Status register and traps. The operand and destination registers remain undisturbed and the instruction can be emulated in software. Any of the IEEE Standard 754 exceptions can arise from the emulated operation, and these exceptions in turn can be simulated.

The Unimplemented exception can also be signaled when unusual operands or result conditions are detected that the implemented hardware cannot handle properly. These include:

- Denormalized operand, except for a Compare instructi
- Quiet Not a Number (NaN) operand, except for a Compare instructi
- Denormalized result or Underflow, when either Underflow or Inexact *Enabl* bits are set or th *FS* bit is not set
- Reserved opcodes
- Unimplemented formats
- Operations that are invalid for their format (for instance, CVT.S.S
- Floating-point-to-integer conversions when the result overflows
- Result underflows when the Underflow trap and Flush-to-Zero mode are both disabled
- Floating-point-to-long integer conversion operations when th required result is wider than the shifter can support -2^{52} to $2^{52} - 1$)
- SQRT in MIPS II mode

Note: Denormalized and NaN operands are only trapped if the instruction is a conversion or computational operation. Moves do not trap if their operands are either denormalized or NaNs. The processor will raise an Unimplemented Operation exception for a CVT[s.d] instruction only when bits 63:55 of the 64-bit integer are not all zeros or ones.

The use of this exception for such conditions is optional; most of these conditions are newly developed and are not expected to be widely used in early implementations. Loopholes are provided in the architecture so that these conditions can be implemented with assistance provided by software, maintaining full compatibility with IEEE Standard 754.

Trap-enabled results. The original operand values are undisturbed.
Trap-disabled results. This trap cannot be disabled.

8.5 Saving and Restoring State

Sixteen or 32 doubleword coprocessor load or store operations save or restore the coprocessor Floating-point register state in memory. The remainder of the control and status information can be saved or restored through Move to/from Coprocessor Control Register instructions and by saving and restoring the processor registers. Normally, the Control/Status register is saved first and restored last.

When the coprocessor Control/Status register (FCR31) is read and the coprocessor is executing one or more floating-point instructions, the instruction(s) in progress are either completed or reported as exceptions. The architecture requires that no more than one of these pending instructions can cause an exception. If the pending instruction cannot be completed, this instruction is placed in the Exception register, if present. Information indicating the type of exception is placed in the Control/Status register. When state is restored, state information in the status word indicates that exceptions are pending.

Writing a zero value to the *Cause* field of the Control/Status register clears all pending exceptions, permitting normal processing to restart after the Floating-Point register state is restored.

The *Cause* field of the Control/Status register holds the results of only one instruction; the FPU examines source operands before an operation is initiated to determine if this instruction can possibly cause an exception. If an exception is possible, the FPU executes the instruction in Stall mode to ensure that no more than one instruction (that might cause an exception) is executed at a time.

8.6 Trap Handlers for IEEE Standard 754 Exceptions

IEEE Standard 754 strongly recommends that users be allowed to specify a trap handler for any of the five standard exceptions that can compute; the trap handler can either compute or specify a substitute result to be placed in the destination register of the operation.

By retrieving an instruction using the processor Exception Program Counter (EPC) register, the trap handler determines:

- Exceptions occurring during the operatio
- The operation being performed
- The destination format

On Overflow or Underflow exceptions (except for conversions), and on Inexact Operation exceptions, the trap handler gains access to the correctly rounded result by examining source registers and simulating the operation in software.

On Overflow or Underflow exceptions encountered on floating-point conversions, and on Invalid Operation and Division by Zero exceptions, the trap handler gains access to the operand values by examining the source registers of the instruction.

The IEEE Standard 754 recommends that, if enabled, the Overflow and Underflow traps take precedence over a separate Inexact trap. This prioritization is accomplished in software; hardware sets both bits.

Bus Interface

9

The system interface allows the processor to access the external resources needed to satisfy cache misses and uncached operations, while permitting an external agent access to some of the processor's internal resources. The bus supports multiple outstanding reads and split response on read transactions to increase system bus efficiency.

The clock portion of the VR5432 system interface has been simplified and many of the external clock signals have been deleted from the system interface of the VR4000 device.

There is also a VR4300 compatibility mode in which the system interface protocol emulates that of the VR4300 series devices. This mode is selected by driving the OptionR43K* input signal low during a cold reset. Multiple-split reads and parity are not supported in this mode.

This chapter describes the system interface of both the processor and the external agent.

9.1 Interface Buses In Native Mode

Figure 9-1 shows the primary communication paths for the system interface in Native mode. These paths consists of a 32-bit multiplexed address and data bus, SysAD (31:0), and a 9-bit command bus, SysCmd (8:0). The SysAD and SysCmd buses are bidirectional.

A request through the system interface consists of:

- An address
- A system interface command that specifies the precise nature of the request
- A series of data elements if the request is for a writ
- A read response for a read reques

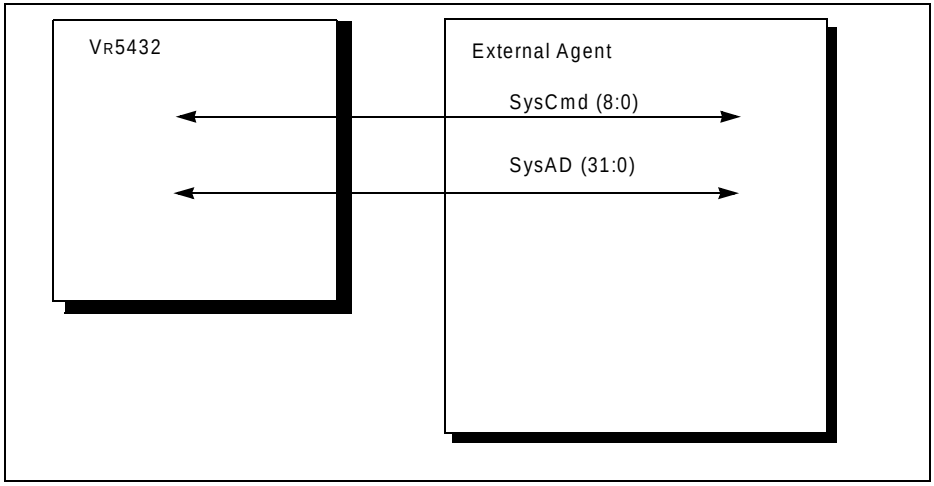


Figure 9-1 System Interface Buses in Native Mode

9.2 Interface Buses in R43K Mode

Figure 9-2 shows the primary communication paths for the system interface in R43K mode. A 32-bit multiplexed address/data bus, SysAD (31:0), transfers addresses/data from the processor to the external agent and vice versa. A 5-bit command bus, SysCmd (4:0), indicates processor/external agent commands. The SysAD and SysCmd buses are bidirectional.

A request through the system interface consists of:

- An address
- A system interface command that specifies the precise nature of the request
- A series of data elements if the request is for a writ
- A read response for a read reques

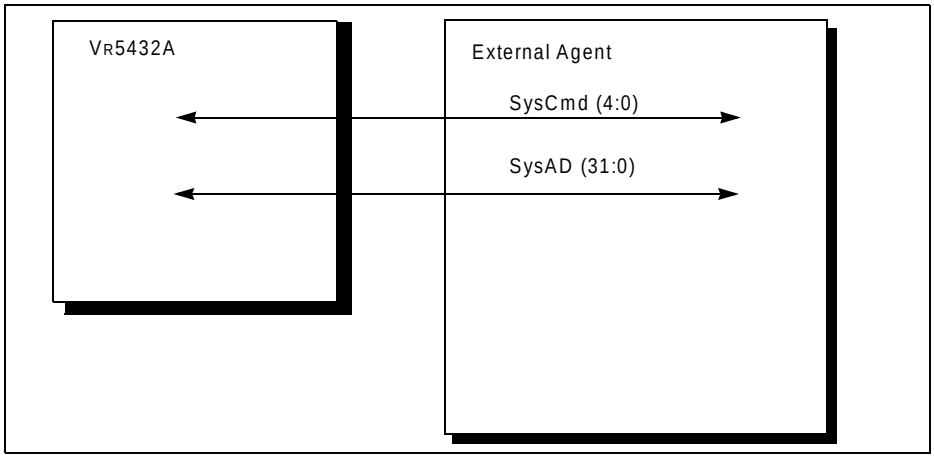


Figure 9-2 System Interface Buses in R43K Mode

System Interface Transactions (Native Mode)

10

This section describes processor and external requests in the native system interface protocol of the VR5432 processor (i.e., the OptionR43K* input signal sampled high during cold reset). The transactions used in R43K (VR4300 compatibility) mode are described in Chapter 12.

All system interface transactions using the VR5432 processor are noncoherent. There is no hardware cache coherency support provided. Requests fall into three categories:

- Cached
- Uncached
- Uncached accelerated

10.1 Terminology

The following terms are used in Chapter 10 through Chapter 13:

- An **external agent** is any device connected to the processor, over the system interface, that processes requests issued by the processor.
- A **system event** is an event that occurs within the processor and requires access to external resources. System events include: an instruction fetch that misses in the instruction cache; a Load/Store instruction that misses in the data cache; an uncached Load or Stor instruction; an execution of a CACHE instruction
- **Sequen** refers to the series of requests that a processor generates t service a system event.
- **Protocol** refers to the cycle-by-cycle signal transitions that occur o the system interface pins, through which external requests are issued.
- **Syntax** refers to the definition of the bit patterns that appear on encoded buses, such as the command bus.

10.2 Processor Requests

When a system event occurs, the processor issues either a single request or a series of requests—called processor requests—through the system interface, to access an external resource and service the event. For this to work, the processor system interface must be connected to an external agent that is compatible with the system interface protocol and can coordinate access to system resources.

An external agent requesting access to a processor internal resource generates an *external request*. This access request passes through the system interface. System events and request cycles are shown in Figure 10-1.

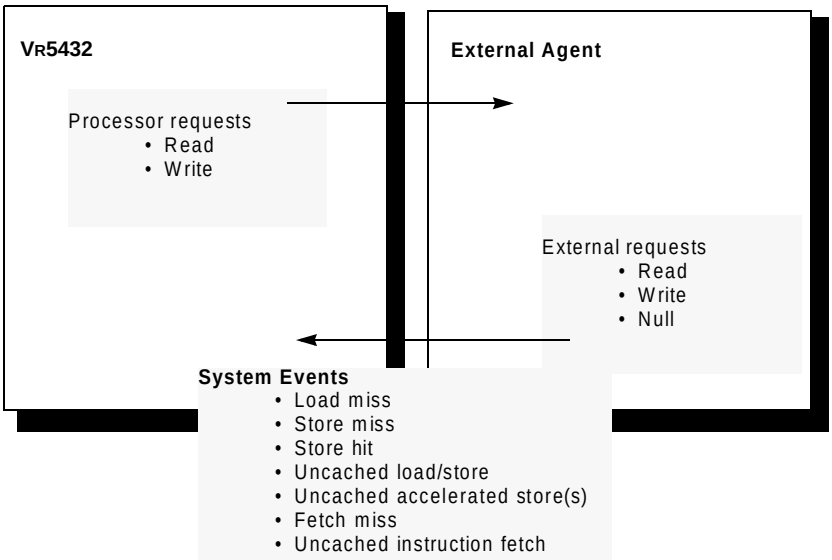


Figure 10-1 Requests and System Events

10.2.1 Rules for Processor Requests

A processor request is a request or a series of requests, through the system interface, to access some external resource. As shown in Figure 10-2, processor requests include read and write.

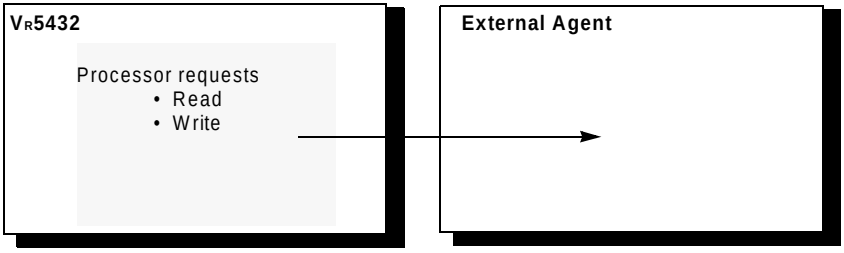


Figure 10-2 Processor Requests to External Agent

The read request asks for a block, doubleword, partial doubleword, word, or partial word of data, either from main memory or from another system resource.

The write request provides a block, doubleword, partial doubleword, word, or partial word of data to be written either to main memory or to another system resource.

The VR5432 processor provides the option of supporting a single outstanding read request like the R4x00/R5x00 processors, or supporting multiple outstanding reads while continuing to allow write cycles during the time the outstanding reads are being processed. The option is programmable in the Configuration register and is called Multiple-Split Read mode.

In the R4x00/R5x00 single transaction scenario, the processor issues a read request and waits for a read response before issuing any subsequent requests. Also, in the single transaction case, the processor submits a write request only if there are no read requests pending.

In the multiple-split read case, the VR5432 supports multiple outstanding read requests and split transactions. This means the VR5432 allows multiple outstanding read transactions, and allows write transactions to be interleaved between the read request and the read response. These additional capabilities are supported for processor requests, but are not supported for external requests.

The processor has the input signals RdRdy* and WrRdy* to allow an external agent to manage the flow of processor requests. RdRdy* controls the flow of processor read requests, while WrRdy* controls the flow of processor write requests. The processor request cycle sequence is shown in Figure 10-3.

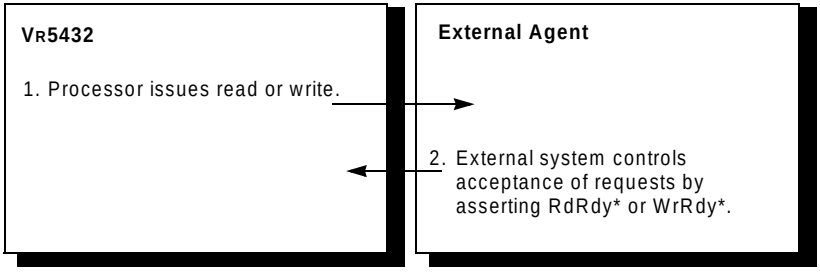


Figure 10-3 Processor Request Flow Control

10.2.2 Processor Read Request

When a processor issues a read request, the external agent must access the specified resource and return the requested data.

A processor read request can be split from the external agent's return of the requested data; in other words, the external agent can initiate an unrelated external request before it returns the response data for a processor read. A processor read request is completed after the last word of response data has been received from the external agent.

Note that the data identifier associated with the response data can signal that the returned data is erroneous, causing the processor to make a bus error.

Processor read requests that have been issued, but for which data has not yet been returned, are said to be *pending*. A read remains pending until the requested read data is returned. For multiple outstanding read requests, the data must be returned in the same order in which the read transactions were issued.

The addition of multiple-split-reads has also necessitated the addition of a read address timing mode similar to the write timing options that were previously a part of the R4x00/R5x00 processors. Therefore, the VR5432 includes support for reissue timing for consecutive transactions (read or write) in the Multiple-Split-Read mode.

In Single Transaction mode, the external agent must be capable of accepting a processor read request any time the following two conditions are met:

- There is no processor read request pending.
- The RdRdy* signal was asserted two cycles before the issue cycle.

In Multiple-Split-Read mode, the external agent must be capable of accepting a processor read request any time the following condition is met:

- The RdRdy* signal was asserted two cycles before the issue cycl and conforms to the requirements of reissue timing

10.2.3 Processor Write Request

When a processor issues a write request, the specified resource is accessed and the data is written to it. A processor write request is complete after the last word of data has been transmitted to the external agent. The VR5432 processor supports R4000-compatible, write reissue, and pipelined write operations, as defined in Section 11.5. However, in Multiple-Split-Read mode, only reissue timing is supported. This is fully explained in Section 11.5.4.

If the VR5432 is *not* in Multiple-Split-Read mode, the external agent must be capable of accepting a processor write request any time the following two conditions are met:

- No processor read request is pending
- The WrRdy* signal was asserted two cycles before the issue cycle, and conforms to the timing mode programmed into the Configuration register.

In Multiple-Split-Read mode, the external agent must be capable of accepting a processor write request any time the following condition is met:

- The WrRdy* signal was asserted two cycles before the issue cycl and conforms to the requirements of reissue timing. Reissue timing is detailed in Section 11.5

10.3 External Requests

External requests include read, write, and null requests, as shown in Figure 10-4. This section also includes a description of a read response, a special case of an external request.

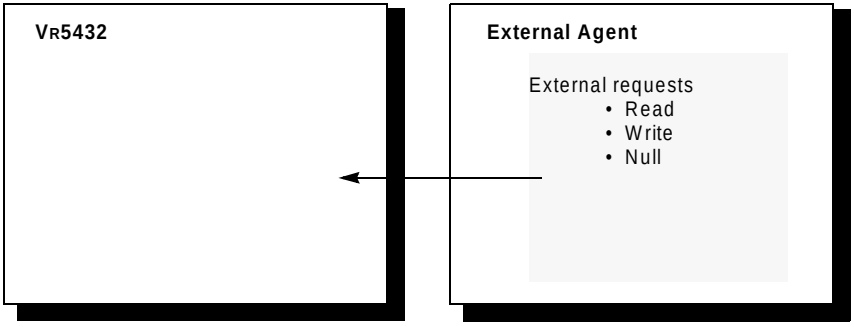


Figure 10-4 External Requests to Processor

The **read** request asks for a word of data from the processor's internal resource.

The **write** request provides a word of data to be written to the processor's internal resource.

The **null** request requires no action by the processor; it provides a mechanism for the external agent to return the system interface to the master state without affecting the processor.

The processor controls the flow of external requests through the arbitration signals ExtRqst*, Release*, and PReq*, as shown in Figure 10-5. The external agent must acquire mastership of the system interface before it is allowed to issue an external request; the external agent arbitrates for mastership of the system interface by asserting ExtRqst* and then waits for the processor to assert Release* for one cycle. After the processor asserts Release*, the external agent masters the bus until it issues a cycle to the processor.

However, the processor can assert PReq* to indicate that it has additional transactions that it would like to perform. This signal is useful when the processor builds up a number of transactions after releasing the bus for an external request. The external agent can optionally return mastership to the processor by issuing a null request. This action can optimize bus activity.

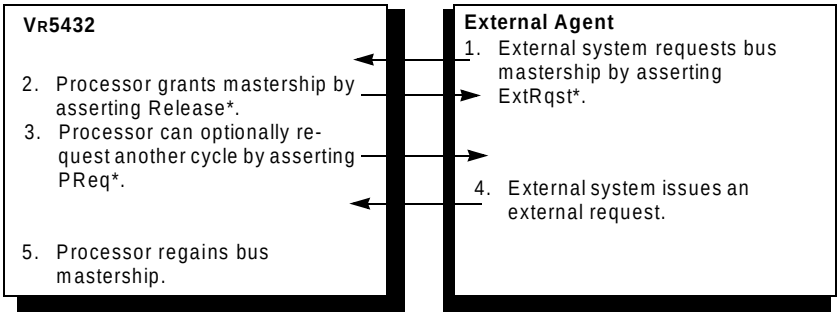


Figure 10-5 External Request Arbitration

Mastership of the system interface always returns to the processor after an external request is issued with ValidIn asserted. The processor does not accept a subsequent external request until it has completed the current request.

The external agent asserts ExtRqst* to indicate that it wishes to begin an external request. The external agent then waits for the processor to signal that it is ready to accept this request by asserting Release*. The processor signals that it is ready to accept an external request based on the following criteria:

- The processor completes the set of requests in progress
- The processor detects ExtRqst*

10.3.1 External Read Request

In contrast to a processor read request, data is returned directly in response to an external read request; no other processor requests can be issued until the processor returns the requested data. An external read request is complete after the processor returns the requested word of data.

Note: The processor does not contain any resources that are readable by an external read request; in response to an external read request, the processor returns undefined data and a data identifier with its *Erroneous Data* bit, *SysCmd* (5), set.

10.3.2 External Write Request

When an external agent issues a write request, the specified resource is accessed and the data is written to it. An external write request is complete after the word of data has been transmitted to the processor.

The only processor resource available to an external write request is the Interrupt register.

10.3.3 Read Response

A read response returns data in response to a processor read request, as shown in Figure 10-6. While a read response is technically an external request, it has one characteristic that differentiates it from all other external requests—it does not perform system interface arbitration. For this reason, read responses are handled separately from all other external requests, and are simply called read responses.

When multiple read requests are outstanding from the processor, responses for all outstanding reads must be asserted before mastership of the system interface returns to the processor. The only exception to this is if an external request other than a read response is asserted. The processor will take mastership of the bus after an external read, write, or null cycle.

The processor can assert PReq* to indicate that it has additional transactions to perform. The external agent can return mastership of the system interface to the processor by asserting a null request cycle.

The data identifier associated with the response data can signal that the returned data is erroneous, causing the processor to make a bus error.

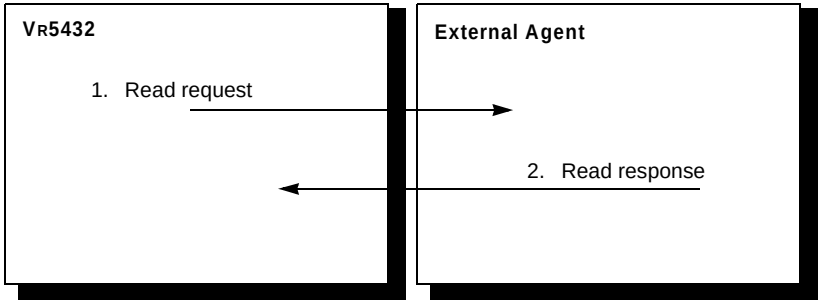


Figure 10-6 External Agent Read Response to Processor Request

10.4 Handling Requests

This section details the sequence of both processor and external requests. The following system events are discussed:

- Load miss
- Store miss
- Store hit
- Uncached loads/stores
- Uncached accelerated stores
- Uncached instruction fetch
- Fetch miss

10.4.1 Load Miss

When a processor load misses in the cache, before the processor can proceed it must obtain the cache line that contains the data element to be loaded from the external agent.

If the new cache line replaces a current dirty cache line, the current cache line must be written back before the new line can be loaded in the cache.

Table 10-1 shows the actions taken on a load miss.

Table 10-1 Load Miss Actions

Page Attribute	State of Data Cache Line Being Replaced	
	Clean/Invalid	Dirty (D = 1)
Cached	BR	BR/BW

BR: Processor block read request
BR/BW: Processor block read request followed by processor block write request

The processor takes the following steps:

1. The processor issues a block read request for the cache line that contains the data element to be loaded.
2. The processor waits for an external agent to provide the read response.
3. The processor continues to execute instructions, unless data dependency is detected between load data and a subsequent instruction. In this case, the pipeline is stalled until the dependency can be resolved. The pipeline is also stalled if a Load/Store instruction tries to access the same data cache line that is under miss servicing.

If the current cache line must be written back, the processor issues a block write request to save the dirty cache line in memory. This write may be done between the read request and the read response if multiple split reads are enabled in the Configuration register.

10.4.2 Store Miss

When a processor store misses in the cache, the processor will request, from the external agent, the cache line that contains the target location of the store.

The processor then executes the following actions:

- The processor issues a block read request for the cache line that contains the data element to be loaded.
- The processor then waits for an external agent to provide the read response.
- The processor continues to execute instructions, unless another Load instruction attempts to access the same cache line that is bein refilled or another Store instruction is executed. In either case, th pipeline is stalled until cache refill is finished
- The cache line is loaded into the cache and the store data is merged into the appropriate location
- If the page attribute is write-through, a nonblock write request is issued

Table 10-2 shows the actions taken on a store miss to the primary cache.

Table 10-2 Store Miss Actions

Page Attribute	State of Data Cache Line Being Replaced	
	Clean/Invalid	Dirty (W = 1)
Write-back	BR	BR/BW
Write-through	BR/W	BR/BW/W

BR: Processor block read request for missed cache line
BW: Processor block write request for replaced dirty data
W: Processor nonblock write request for write-through data

If the page attribute is write-back or write-through, the processor issues a block read request for the cache line that contains the data element to be loaded. If a cache line must be written back due to replacement, the processor issues a write request for that cache line. If the page attribute is write-through, the processor

issues a nonblock write request for the new data. Writes may be done between a read request and a read response if multiple-split-reads are enabled in the Configuration register.

10.4.3 Store Hit

The action on the system bus is determined by whether the line is write-back or write-through. Write-back store hits cause no bus transactions. For lines with a write-through policy, the store generates a processor nonblock write request for the store data.

10.4.4 Uncached Loads or Stores

When the processor performs an uncached load, it issues a doubleword, partial doubleword, word, or partial word read request. When the processor performs an uncached store, it issues a doubleword, partial doubleword, word, or partial word write request. All writes by the processor are buffered from the system interface by a four-deep transaction buffer. Since this buffer behaves as a FIFO, previous write requests in the buffer are completed before a following read request is serviced.

10.4.5 Uncached Accelerated Stores

Uncached accelerated operations are uncached operations to a page with an uncached accelerated cache algorithm. When the processor performs an uncached accelerated store, it can perform a block write, or it can perform one or more doubleword, partial doubleword, word, or partial word write requests. All writes by the processor are buffered from the system interface by a four-deep transaction buffer. Since this buffer behaves as a FIFO, previous write requests in the buffer are completed before a following read request is serviced.

Uncached accelerated operations allow the user to combine several sequential uncached word or doubleword operations into a single 32-byte block of write data and only generate a single external SysAD bus transaction. In order for the programmer to optimize these transactions, special attention should be paid to the uncached accelerated data alignment and gathering rules.

Uncached accelerated writes pass through the transaction buffer in FIFO order, the same as all other types of transactions. However, successive uncached accelerated transactions are assembled into up to four doubleword FIFO entries if the uncached accelerated write gathering rules are followed. These rules are as follows:

- The initial uncached accelerated transaction must be aligned on mod 32-byte boundary
- All uncached accelerated transactions must be word or doublewor sized.
- Word and doubleword transactions must be naturally aligned
- Word writes must happen in pairs in order to form a naturally aligned doubleword.
- Addresses must increase sequentially.

Uncached accelerated write transactions that do not follow the rules stated above will be treated as nonaccelerated uncached transactions. In addition, if a gathering sequence is disrupted by any operation other than those making up the uncached accelerated gather, the portion of the data that is gathered will be sent out as uncached word or doubleword operations.

The uncached accelerated operation will also be disrupted if the processor enters Debug mode. When Debug mode is entered, the transaction buffer is emptied. In addition, it is likely that the gather operation will be disrupted if an exception is taken.

10.4.6 Uncached Instruction Fetch

The processor issues word reads for instruction fetches to uncached addresses. Word reads must conform to the byte alignment as indicated by the size encoded on *SysCmd* (2:0). Thus, any system ROM address space accessed during a processor boot restart must support properly aligned 32-bit reads.

10.4.7 Fetch Miss

When the processor misses the instruction cache during an instruction fetch, it issues a read request for cache line acquisition. An external agent returns data as a read response.

System Interface Protocols (Native Mode)

11

The following sections contain a cycle-by-cycle description of the system interface protocols for each type of processor and external request in the native protocol of the VR5432. For the protocols followed in R43K (VR4300 compatibility) mode, see Chapter 13.

11.1 Address and Data Cycles

Cycles in which the SysAD bus contains a valid address are called address cycles. Cycles in which the SysAD bus contains valid data are called data cycles. Validity of addresses and data from the processor is determined by the state of the ValidOut* signal. Validity of the address and data from the external agent is determined by the state of the ValidIn* signal.

The SysCmd bus identifies the contents of the SysAD bus during any cycle in which it is valid from the processor or the external agent. The most-significant bit of the SysCmd bus is always used to indicate whether the current cycle is an address cycle or a data cycle.

- During address cycles *SysCmd* (8) = 0. The remainder of the SysCmd bus *SysCmd* (7:0), contains the encoded system interface command.
- During data cycles, *SysCmd* (8) = 1. The remainder of the SysCmd bus, *SysCmd* (7:0), contains an encoded data identifier.

11.2 Issue Cycles

There are two types of processor issue cycles:

- Processor read reques
- Processor write reques

The processor samples the signal RdRdy* to determine the issue cycle for a processor read; the processor samples the signal WrRdy* to determine the issue cycle of a processor write request.

As shown in Figure 11-1, RdRdy* must be asserted two cycles prior to the first address cycle of the processor read request in order to define the address cycle as the issue cycle.

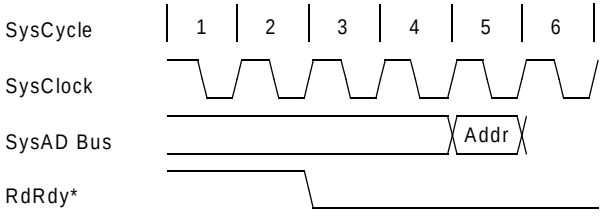


Figure 11-1 State of RdRdy Signal for Read Requests*

As shown in Figure 11-2, WrRdy* must be asserted two cycles prior to the first address cycle of the processor write request to define the address cycle as the issue cycle.

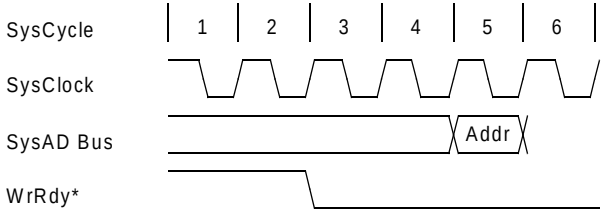


Figure 11-2 State of WrRdy Signal for Write Requests*

The processor repeats the address cycle for the request until the conditions for a valid issue cycle are met. After the issue cycle, if the processor request requires data to be sent, the data transmission begins. There is only one issue cycle for any processor request.

The processor accepts external requests, even while attempting to issue a processor request, by releasing the system interface to slave state in response to an assertion of ExtRqst* by the external agent.

Note that the rules governing the issue cycle of a processor request are strictly applied to determine which action the processor takes. The processor can either:

- Complete the issuance of the processor request in its entirety befor the external request is accepted, o
- Release the system interface to slave state without completing th issuance of the processor reques

In the latter case, the processor issues the processor request (if the processor request is still necessary) after the external request is complete. The rules governing an issue cycle again apply to the processor request.

11.3 Handshake Signals

The VR5432 processor manages the flow of requests through the following six control signals:

- RdRdy* and WrRdy* are used by the external agent to indicate whe it can accept a new read (RdRdy*) or write (WrRdy*) transaction.
- ExtRqst* and Release* are used to transfer control of the SysAD and SysCmd buses. ExtRqst* is used by an external agent to indicate a need to control the interface. Release* is asserted by the processor when it transfers control of the system interface to the external agent.
- The VR5432 processor uses ValidOut* and the external agent uses ValidIn* to indicate valid commands and data on the SysCmd and SysAD buses.

11.4 System Interface Operation

Figure 11-3 shows how the system interface operates from register to register. That is, processor outputs come directly from output registers and begin to change with the rising edge of SysClock.

Processor inputs are fed directly to input registers that latch these input signals with the rising edge of SysClock. This allows the system interface to run at the highest possible clock frequency.

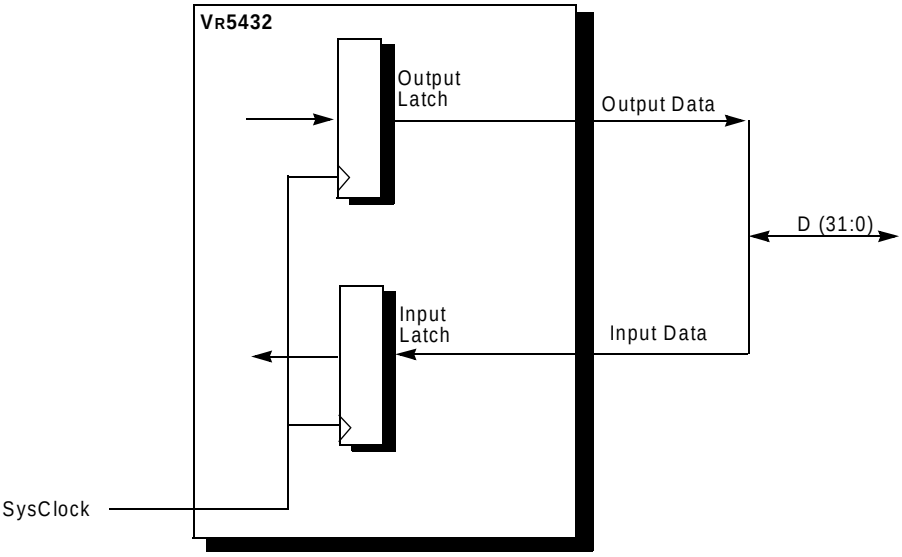


Figure 11-3 System Interface Register-to-Register Operation

11.4.1 Master and Slave States

When the VR5432 processor is driving the SysAD and SysCmd buses, the system interface is in master state. When the external agent is driving the SysAD and SysCmd buses, the system interface is in slave state.

In master state, the processor asserts the ValidOut* signal whenever the SysAD and SysCmd buses are valid.

In slave state, the external agent asserts the ValidIn* signal whenever the SysA and SysCmd buses are valid.

The system interface remains in master state unless one of the following events occurs:

- The external agent requests and is granted the system interface (external arbitration).
- The processor issues a read request and asserts the Release* signal as an uncompelled change to slave state.

11.4.2 External Arbitration

External arbitration is performed by the processor. The system interface must be in slave state for the external agent to issue an external request through the system interface. The transition from master state to slave state is arbitrated by the master processor, using the system interface handshake signals ExtRqst* and Release*. This transition is described by the following procedure:

1. An external agent signals that it wishes to issue an external request by asserting ExtRqst*.
2. When the processor is ready to accept an external request, it releases the system interface from master to slave state by asserting Release* for one cycle.
3. The system interface returns to master state as soon as the external request is complete.

11.4.3 Uncompelled Change to Slave State

An uncompelled change to slave state is the transition of the system interface from master state to slave state, initiated by the processor when a processor read request is pending. Release* is asserted automatically after a read request; an uncompelled change to slave state then occurs. This transition to slave state allows the external agent to return read response data without arbitrating for bus ownership. In Multiple-Split-Read mode, the uncompelled change to slave state may not occur immediately after the read request occurs. In this case, it will occur depending on the internal state of the processor and the state of the RdRdy*, WrRdy*, and ExtRqst* signals.

After an uncompelled change to slave state, the processor returns to master state at the end of the next external request. This can be a read response or some other type of external request. In Multiple-Split Read mode, the processor will return to master state if an external request occurs, or after a read response has been issued for all outstanding reads. If the external agent issues an external request or there is another pending read request, the processor performs another uncompelled change to slave state as it requires by asserting Release* for one cycle. The processor release is asserted depending on the internal state of the processor.

An external agent must note that the processor has performed an uncompelled change to slave state and begin driving the SysAD bus along with the SysCmd bus. As long as the system interface is in slave state, the external agent can begin an external request without arbitrating for the system interface; that is, without asserting ExtRqst*.

Table 11-1 lists the abbreviations and definitions for each of the buses that are used in the timing diagrams that follow.

Table 11-1 System Interface Requests

Scope	Abbreviation	Meaning
Global	Unsd	Unused
SysAD bus	Addr	Physical address
	Data<n>	Data element number n of a block of data
SysCmd bus	Cmd	An unspecified system interface command
	Read	A processor or external read request command
	Write	A processor or external write request command
	SINull	A system interface release external null request command
	NData	A data identifier for a data element other than the last data element
	NEOD	A data identifier for the last data element

11.5 Processor Request Protocols

This section describes the read and write processor request protocols.

Note: In the timing diagrams, the two closely spaced, wavy vertical lines, such as those shown in Figure 11-4, indicate one or more identical cycles that are not illustrated due to space constraints.

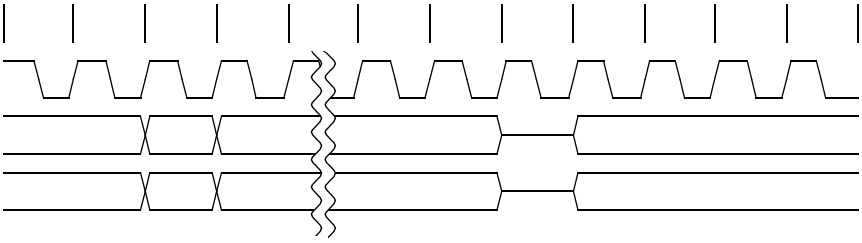


Figure 11-4 Symbol for Undocumented Cycles

11.5.1 Processor Read Request Protocol

The following sequence describes the protocol for doubleword, partial doubleword, word, and partial word processor read requests. The following steps correspond to Figure 11-5.

1. RdRdy* is asserted low, indicating that the external agent is ready to accept a read request.
2. With the system interface in master state, a processor read request is issued by driving a read command on the SysCmd bus and a read address on the SysAD bus. The physical address is driven onto SysAD (31:0). All other bits are driven to zero.
3. At the same time, the processor asserts ValidOut* for one cycle, indicating that valid data is present on the SysCmd and the SysAD buses.
4. The processor makes an uncompelled change to slave state. The external agent must not assert the ExtRqst* signal for the purpose of returning a read response, but rather must wait for the uncompelled change to slave state. The signal ExtRqst* can be asserted before or during a read response to perform an external request other than a read response.
5. The processor releases the SysCmd and the SysAD buses one SysClock after the assertion of Release*.
6. The external agent drives the SysCmd and the SysAD buses within two cycles after the assertion of Release*.

Once in slave state, the external agent can return the requested data through a read response. The read response can return the requested data or, if the requested data could not be successfully retrieved, an indication that the returned data is erroneous. If the returned data is erroneous, the processor takes a Bus Error exception.

Figure 11-5 illustrates a processor read request, coupled with an uncompelled change to slave state that occurs as the read request is issued.

Note: Timing for the SysADC bus is the same as for the SysAD bus.

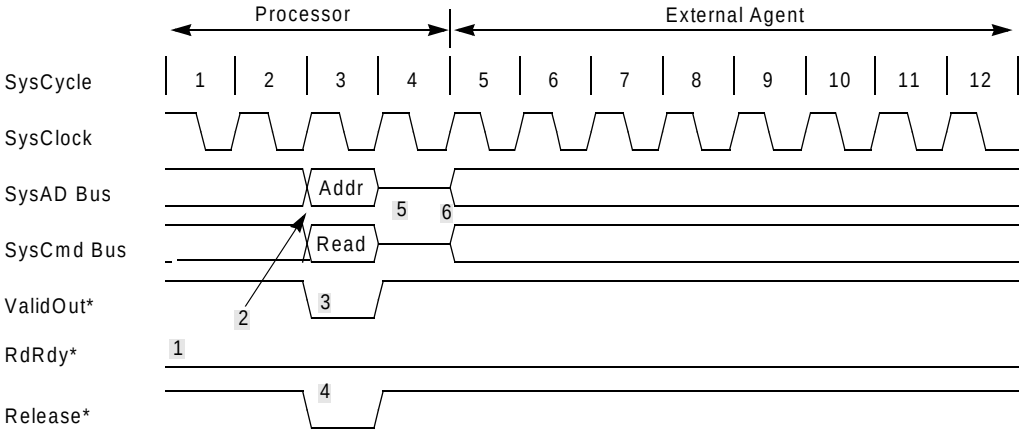


Figure 11-5 Processor Read Request Protocol

The processor can accept either a read response, if a read is pending, or another type of external request in response to a Release* assertion. It is up to the external agent to maintain consistent behavior on the ExtRqst* signal.

If the processor releases the bus and an external request is not issued, it will not release again unless ExtRqst* is asserted. In addition, if ExtRqst* is detected by the processor, it can release the bus at any time, and the external agent must be able to respond with an appropriate cycle.

However, if a read request is pending and an external request other than a read response is issued, the processor will assert Release* for the pending read response after processing the external request. When the processor becomes the master of the bus due to an external request, it can assert additional transactions as allowed by the current bus timing mode before releasing the bus for a read response or an external request.

11.5.2 Processor Write Request Protocol

Processor write requests are issued using one of the two following protocols.

- Word or partial word writes using a nonblock write request protoco
- Cached block writes and uncached, accelerated writes using a bloc write request protocol

Processor nonblock write requests are issued with the system interface in master state, as described in the steps below. Figure 11-6 shows a processor nonblock write request cycle.

1. WrRdy* is asserted low, indicating that the external agent is ready to accept a write request.
2. A processor single nonblock write request is issued by driving a write command on the SysCmd bus and a write address on the SysAD bus. The physical address is driven onto SysAD (31:0). All other bits are driven to zero.
3. The processor asserts ValidOut*.
4. The processor drives a data identifier on the SysCmd bus and data on the SysAD bus.
5. The data identifier associated with the data cycle must contain a last-data-cycle indication. At the end of the cycle, ValidOut* is deasserted.

   Note: Timing for the SysADC bus is the same as for the SysAD bus.

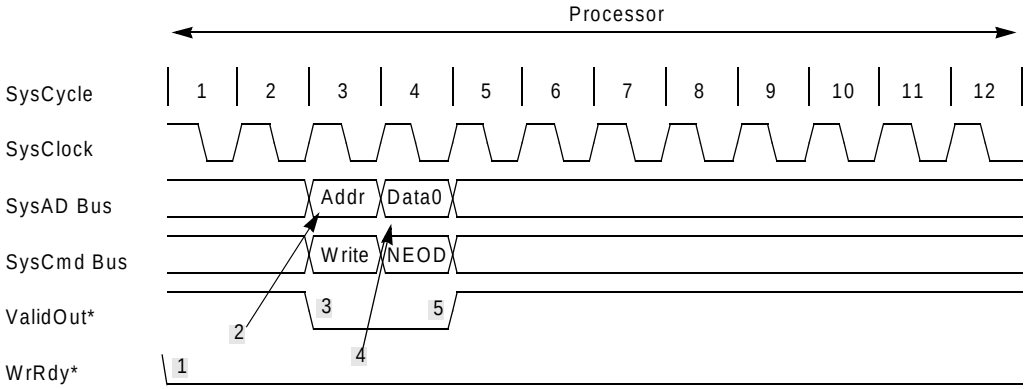


Figure 11-6 Processor Nonblock Write Request Protocol

Figure 11-7 illustrates a cache block write request. The steps for the block write request are the same as those indicated for the nonblock write request.

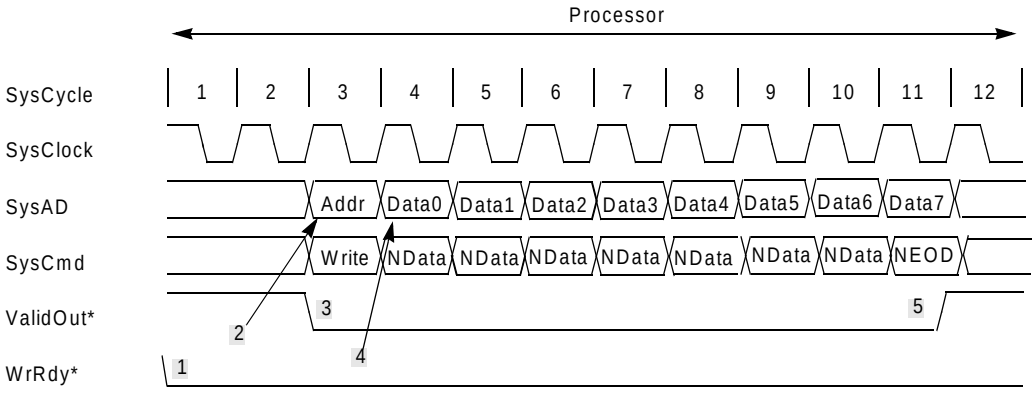


Figure 11-7 Processor Block Write Request

11.5.3 Processor Request Flow Control

The external agent uses RdRdy* to control the flow of processor read requests.

Figure 11-8 illustrates this flow control, as described in the steps below.

1. The processor samples the RdRdy* signal to determine if the external agent is capable of accepting a read request.
2. A read request is issued to the external agent.
3. The external agent deasserts RdRdy*, indicating that it cannot accept additional read requests.
4. The read request issue is stalled because RdRdy* was negated two cycles earlier.
5. A read request is again issued to the external agent.

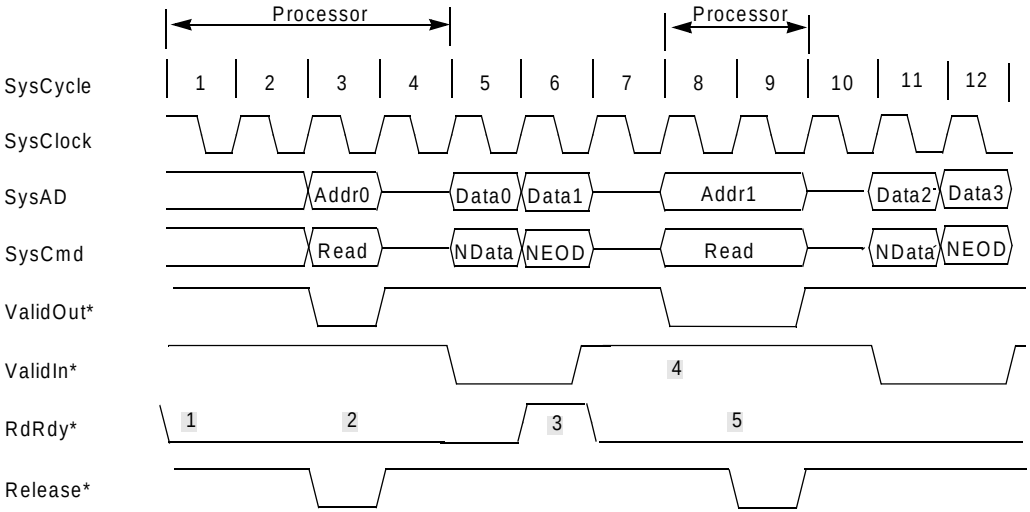


Figure 11-8 Processor Request Flow Control

Figure 11-9 illustrates two processor write requests in which the issue of the second is delayed for the assertion of WrRdy*.

1. WrRdy* is asserted low, indicating the external agent is ready to accept a write request.
2. The processor asserts ValidOut*, a write command on the SysCmd bus, and a write address on the SysAD bus.
3. The second write request is delayed until the WrRdy* signal is again asserted.
4. The processor does not complete the issue of a write request until it issues an address cycle in response to the write request for which the signal WrRdy* was asserted two cycles earlier.

   Note: Timing for the SysADC bus is the same as for the SysAD bus.

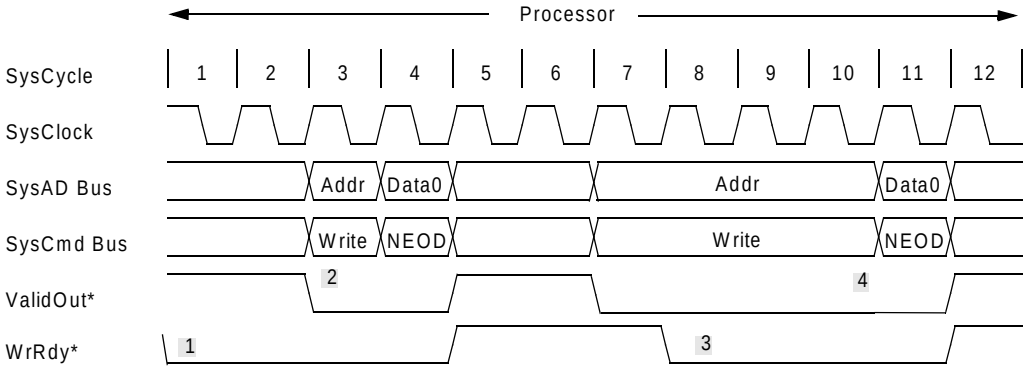


Figure 11-9 Two Processor Write Requests with Second Write Delayed

11.5.4 Processor Request Timing Modes

The VR5432 processor supports timing modes that allow it to be compatible with the R4x00/R5x00 processors and provides new, more efficient timing modes that are unique to the VR5432. The R4x00/R5x00 timing modes include R4000 compatible, write reissue, and pipelined writes. These modes are ignored in R43K mode, as described in Chapter 13. The VR5432 processor also performs multiple-split-reads.

The VR5432 processor interface requires that WrRdy* be asserted two system cycles prior to the issue of a write cycle. An external agent that negates WrRdy* immediately upon receiving the write that fills its buffer will suspend any subsequent writes for four system cycles in R4000 nonblock write-compatible mode. The processor always inserts at least two unused system cycles after a write address/data pair in order to give the external agent time to suspend the next write.

Figure 11-10 shows back-to-back write cycles in R4000-compatible mode.

1. WrRdy* is asserted, indicating that the processor can issue a write request.
2. WrRdy* remains asserted, indicating that the external agent can accept another write request.
3. WrRdy* deasserts, indicating that the external agent cannot accept another write request, stalling the issue of the next write request.

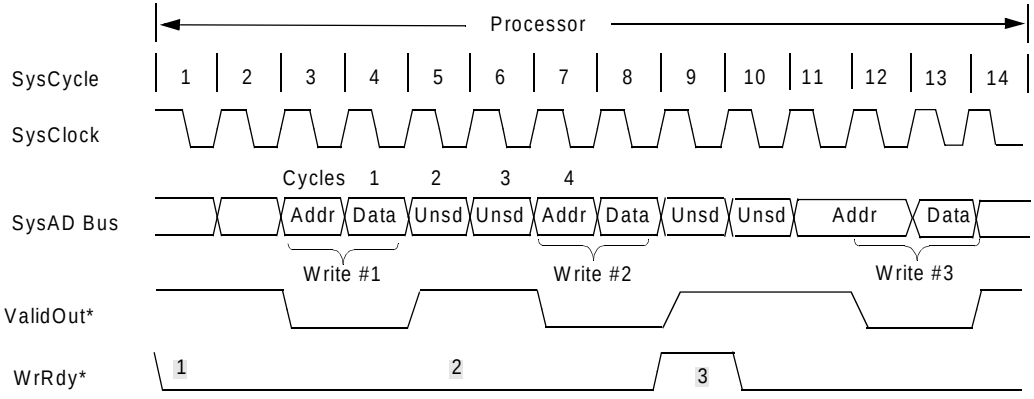


Figure 11-10 R4000-Compatible Back-to-Back Write Cycle Timing

A read address that follows a write transaction will follow the unused cycle pattern indicated by the R4000-compatible mode of one address every four cycles. This is illustrated in Figure 11-11.

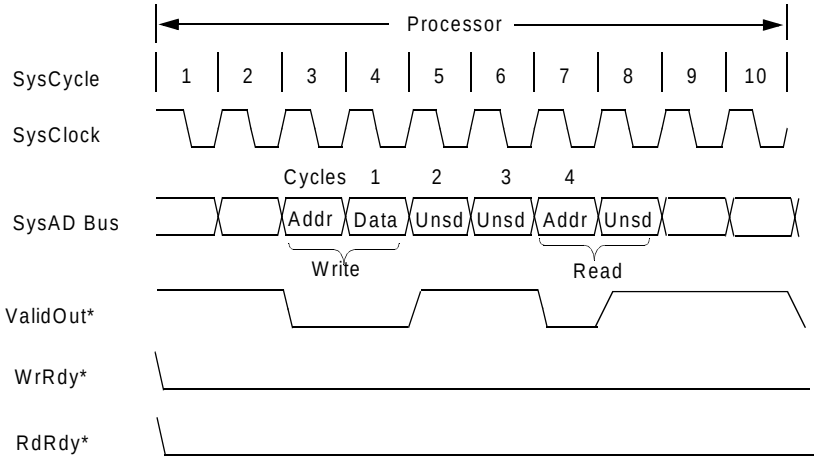


Figure 11-11 Read Cycle Following a Nonblock Write Cycle

An address/data pair every four system cycles is not sufficiently high performance for all applications. For this reason, the VR5432 processor provides two protocol options that modify the R4000 back-to-back write protocol to allow an address/data pair every two system cycles. It also enhances the read protocol to allow multiple outstanding read requests to exist, and to allow requests of any kind to be issued at a rate of one address every two cycles. These protocols are as follows.

Write reissue allows WrRdy* to be negated during the address cycle and forces the write cycle to be reissued.

Pipelined writes leave the sample point of WrRdy* unchanged, two cycles before the issue cycle, and requires that the external agent accept one more write than dictated by the R4000 protocol.

Multiple-split read timing allows RdRdy* to be negated during the address cycle and forces the read cycle to be reissued. It also allows WrRdy* to be negated during the address cycle and forces the write cycle to be reissued. This is the same timing as a write reissue, but it is extended to apply to both read and write addresses. In addition, it allows subsequent read or write addresses to be issued every two cycles. The unique characteristic of multiple-split-read behavior is that it will continue to issue read or write transactions without asserting Release* until one of the following conditions exists:

- The processor does not have any transactions to issue and a read request is outstanding
- The resources required for the transaction are not available, as indicated by RdRdy* or WrRdy*, and a read request is outstanding or ExtRqst* is asserted.
- ExtRqst* is asserted, and the processor is at a transaction set boundary.
- There are read requests outstanding and a set of write transactions completes.
- The processor limit of four outstanding read requests is reached. Not that the processor does not have an internal limit on write requests and can issue write requests after the limit of four outstanding reads is reached. However, the bus will be released before any additional read requests are issued.

Read data is required to be returned in the order in which the read request was issued. Also note that the write timing for Multiple-Split-Read mode is always consistent with write reissue timing and does not use pipelined write timing.

The write reissue protocol is shown in Figure 11-12. Writes issue when WrRdy* is asserted both two cycles prior to the address cycle and during the address cycle.

1. WrRdy* is asserted, indicating that the external agent can accept a write request.
2. WrRdy* remains asserted as the write is issued, and the external agent is ready to accept another write request.
3. WrRdy* deasserts during the address cycle. This write request is aborted and reissued.
4. WrRdy* is asserted, indicating that the external agent can accept a write request.
5. WrRdy* remains asserted as the write is issued, and the external agent is able to accept another write request.

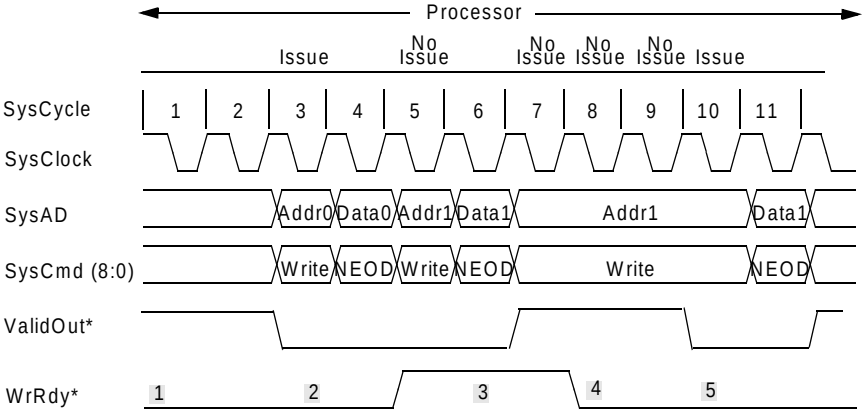


Figure 11-12 Write Reissue Protocol

The pipelined write protocol is shown in Figure 11-13. Writes issue when WrRdy* is asserted two cycles before the address cycle and the external agent is required to accept one more write after WrRdy* is negated.

1. WrRdy* is asserted, indicating that the external agent can accept a write request.
2. WrRdy* remains asserted as the write is issued, and the external agent is able to accept another write request.
3. WrRdy* is deasserted, indicating that the external agent cannot accept another write request; it does, however, accept this write.
4. WrRdy* is asserted, indicating that the external agent can accept a write request.

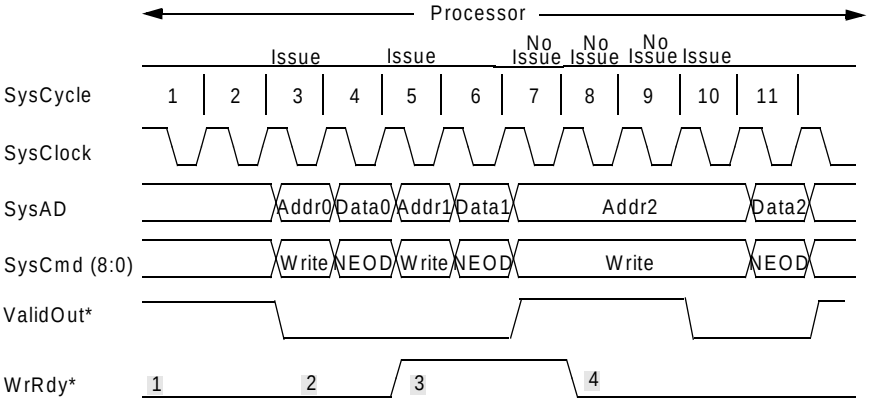


Figure 11-13 Pipelined Write Cycles

Multiple-split-read protocol timing scenario #1 is shown in Figure 11-14. Read cycles issue only when RdRdy* is asserted both two cycles prior to the address cycle and during the address cycle. Also, write cycles issue when WrRdy* is asserted both two cycles prior to the address cycle and during the address cycle.

1. RdRdy* is asserted, indicating that the external agent can accept a read request.
2. RdRdy* remains asserted as the read is issued, indicating that the read request is accepted.
3. RdRdy* deasserts during the address cycle. This read request is aborted and will be reissued.
4. Release* is asserted when the resources are not available for the processor to do the second read operation.

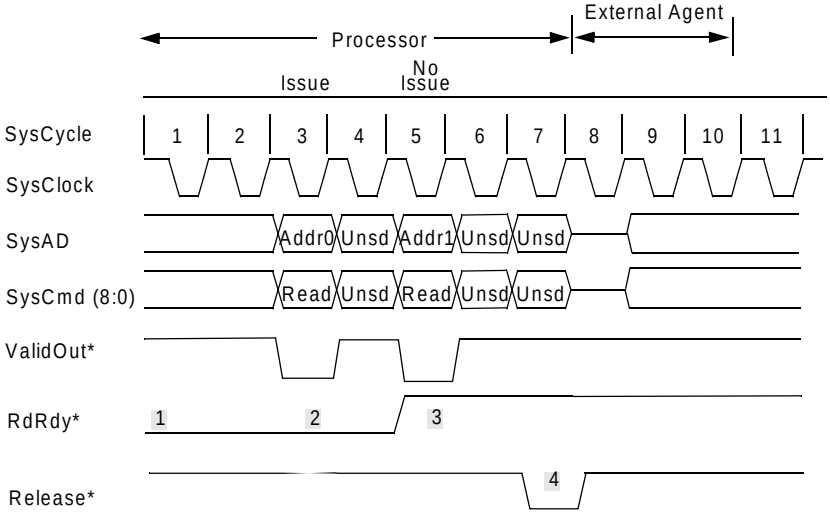


Figure 11-14 Multiple-Split-Read Cycle Scenario #1

Multiple-split-read protocol timing scenario #2 is shown in Figure 11-15.

1. RdRdy* is asserted, indicating that the external agent can accept a read request.
2. RdRdy* remains asserted as the first read is issued, indicating that the read request is accepted.
3. RdRdy* remains asserted during the second read address cycle, making this a valid read request cycle.
4. Release* is asserted when the processor has asserted all the operations it has queued.

   Note: Release* can be asserted in step 4 of this scenario, even if RdRdy* is deasserted in the same cycle. In this case, only the first request is successfully issued. In Multiple-Split-Read mode, it is possible for the processor to assert Release* when the external agent has deasserted RdRdy* in what could have been an issue cycle. If this occurs, and there are no other outstanding read requests, it is the responsibility of the external agent to issue a null or other external request to return bus control to the processor. The external agent should always follow the Single Transaction mode rule and accept a read request from the processor if there are no read requests pending and RdRdy* is asserted two cycles prior to an issue cycle.

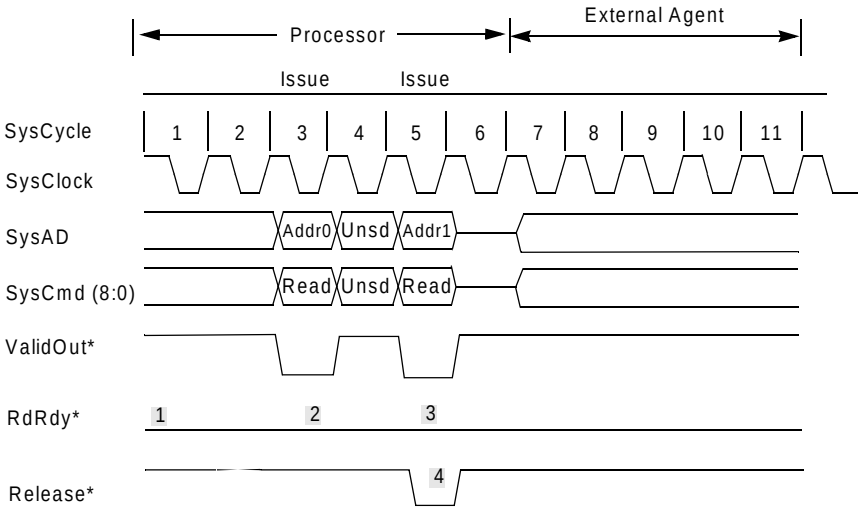


Figure 11-15 Multiple-Split-Read Cycle Scenario #2

Figure 11-16 shows the processor issuing two read cycles back-to-back, followed by a block write cycle in Multiple-Split-Read mode. This shows one case where multiple transactions can be issued, but the order and type of cycles is mixed. The control for transaction issue is from RdRdy*, WrRdy*, ExtRqst*, and the processor's transaction queue. The case of a read followed by a write is a likely case for a cache miss where data is being replaced in the cache.

1. RdRdy* is asserted, indicating that the external agent can accept a read request.
2. RdRdy* remains asserted as the read is issued, indicating that the first read request is accepted.
3. RdRdy* remains asserted as the second read is issued, indicating that the second read request is accepted.
4. WrRdy* is asserted, indicating that the external agent can accept a write request.
5. WrRdy* remains asserted as the write is issued, indicating that the write request is accepted.
6. Release* is asserted when the processor has asserted all the operations it has queued.

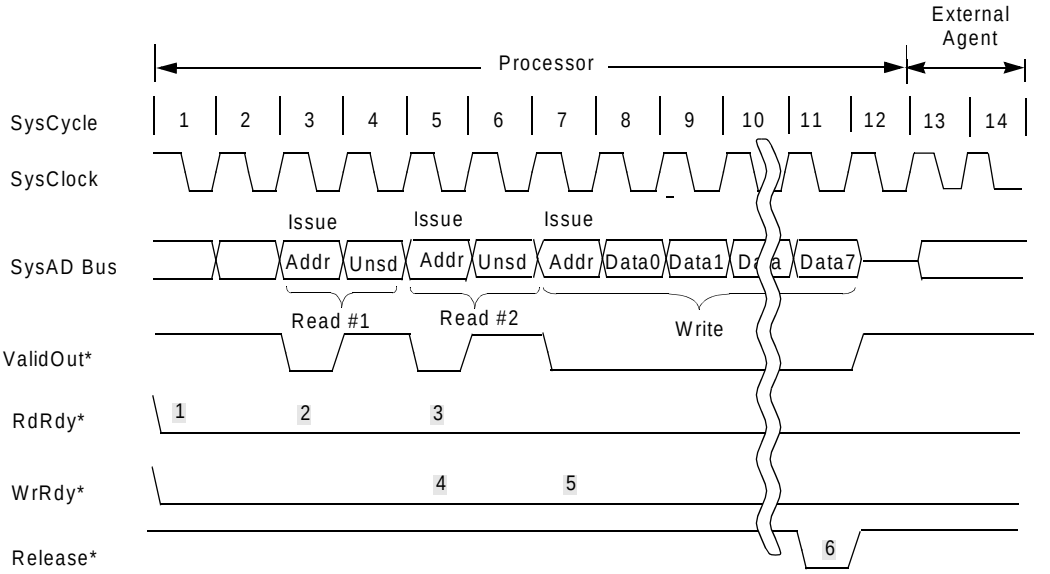


Figure 11-16 Multiple-Split-Read Cycles with Two Reads Split by One Write

An additional case presented by multiple-split-read cycles involves the PReq* signal. PReq* allows the processor to convey to the external agent that it has additional transactions to issue. To make the best use of the system interface bandwidth, it is beneficial to allow the processor to use bus cycles that would otherwise be wasted waiting for read data to be returned. The following sequence is illustrated in Figure 11-17:

1. RdRdy* is asserted low, indicating that the external agent is ready to accept a read request.
2. The processor makes an uncompelled change to slave state as the read request is issued.
3. The processor asserts PReq* to indicate that it has additional bus transactions pending.
4. The external agent issues a null request in order to allow the processor to regain bus control. Null requests are explained in Section 11.6.3.
5. The processor resumes control of the bus.

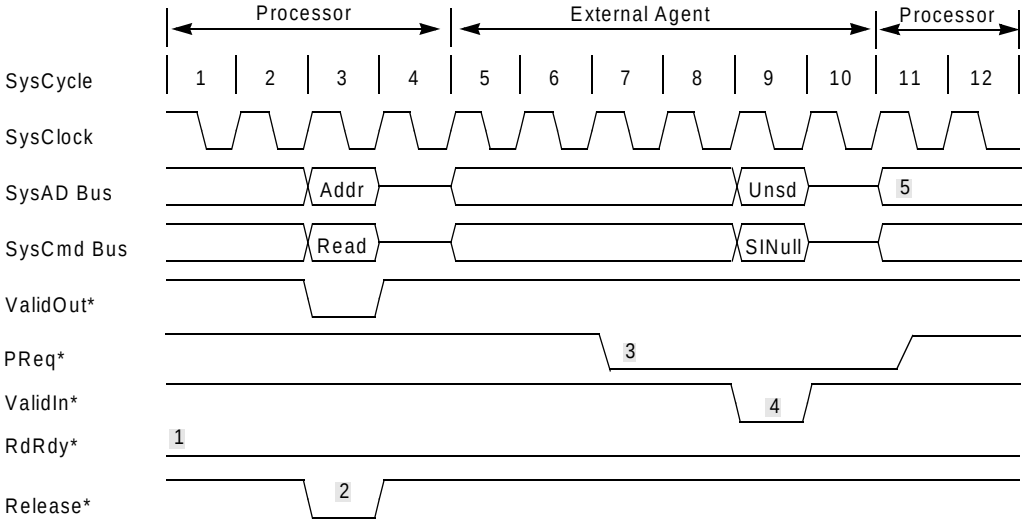


Figure 11-17 PReq After a Processor Read Cycle*

11.6 External Request Protocols

External requests can only be issued with the system interface in slave state. An external agent asserts ExtRqst* to arbitrate for the system interface, then waits for the processor to release the system interface to slave state by asserting Release* before the external agent issues an external request. If the system interface is already in slave state—that is, the processor has previously performed an uncompelled change to slave state—the external agent can begin an external request immediately.

After issuing an external request, the external agent must return the system interface to master state. If the external agent does not have any additional external requests to perform, ExtRqst* must be deasserted two cycles after the cycle in which Release* was asserted. For a string of external requests, the ExtRqst* signal is asserted until the last request cycle, at which point it is deasserted two cycles after the cycle in which Release* was asserted.

The processor continues to handle external requests as long as ExtRqst* is asserted; however, the processor cannot release the system interface to slave state for a subsequent external request until it has completed the current request. As long as ExtRqst* is asserted, the string of external requests is not interrupted by a processor request.

External write requests intended for the processor must have SysAD (6:4) = 000_2 in the address cycle. If the address cycle does not assert the appropriate signals, VR5432 will ignore the cycle. The processor will retake control of the system interface.

This section describes the following external request protocols:

- Read
- Null
- Writ
- Read response

11.6.1 External Arbitration Protocol

System interface arbitration uses the signals ExtRqst* and Release*, as described previously. Figure 11-18 is a timing diagram of the arbitration protocol, in which slave and master states are shown.

The arbitration cycle consists of the following steps:

1. The external agent asserts ExtRqst* when it wishes to submit an external request.
2. The processor waits until it is ready to handle an external request, at which point it asserts Release* for one cycle.
3. The processor puts the SysAD and SysCmd buses in tristate mode.
4. The external agent must wait at least two cycles after the assertion of Release* before it drives the SysAD and SysCmd buses.
5. The external agent negates ExtRqst* two cycles after the assertion of Release*, unless the external agent wishes to perform an additional external request.
6. The external agent sets the SysAD and the SysCmd buses to tristate at the completion of an external request.

The processor can start issuing a processor request one cycle after the external agent sets the bus to tristate.

Note: Timing for the SysADC bus is the same as for the SysAD bus.

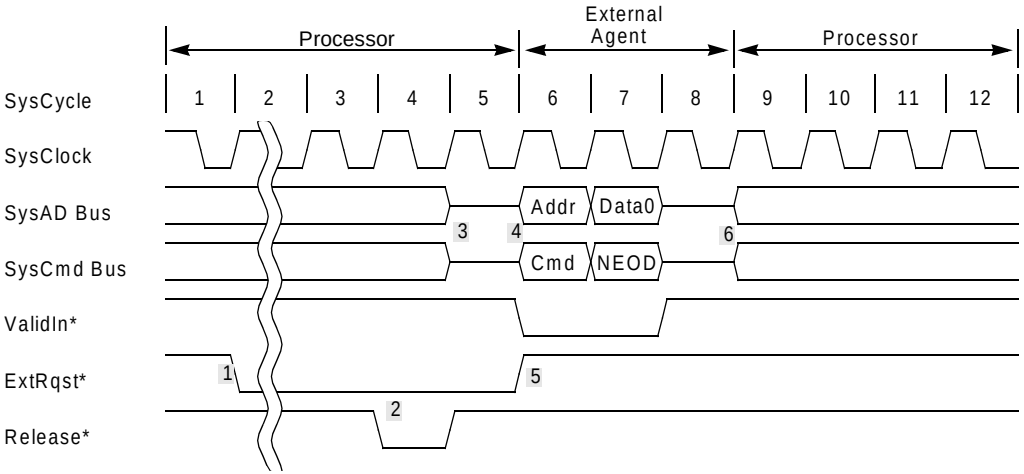


Figure 11-18 Arbitration Protocol for External Requests

11.6.2 External Read Request Protocol

External reads are requests for a word of data from a processor internal resource, such as a register. External read requests cannot be split—that is, no other request can occur between the external read request and its read response.

Figure 11-19 shows a timing diagram of an external read request, which consists of the following steps:

1. An external agent asserts ExtRqst* to arbitrate for the system interface.
2. The processor releases the system interface to slave state by asserting Release* for one cycle and then deasserting Release*.
3. After Release* is deasserted, the SysAD and SysCmd buses are put in tristate mode for one cycle.
4. The external agent drives a read request command on the SysCmd bus and a read request address on the SysAD bus and asserts ValidIn* for one cycle.
5. After the address and command are sent, the external agent releases the SysCmd and SysAD buses by setting them to tristate and allowing the processor to drive them. The processor, having accessed the data that is the target of the read, returns this data to the external agent. The processor accomplishes this by driving a data identifier on the SysCmd bus, the response data on the SysAD bus, and asserting ValidOut* for one cycle. The data identifier indicates that this is last-data-cycle response data.
6. The system interface is in master state. The processor continues driving the SysCmd and SysAD buses after the read response is returned.

   Note: Timing for the SysADC bus is the same as for the SysAD bus.

External read requests are only allowed to read a word of data from the processor. The processor response to external read requests for any data element other than a word is undefined.

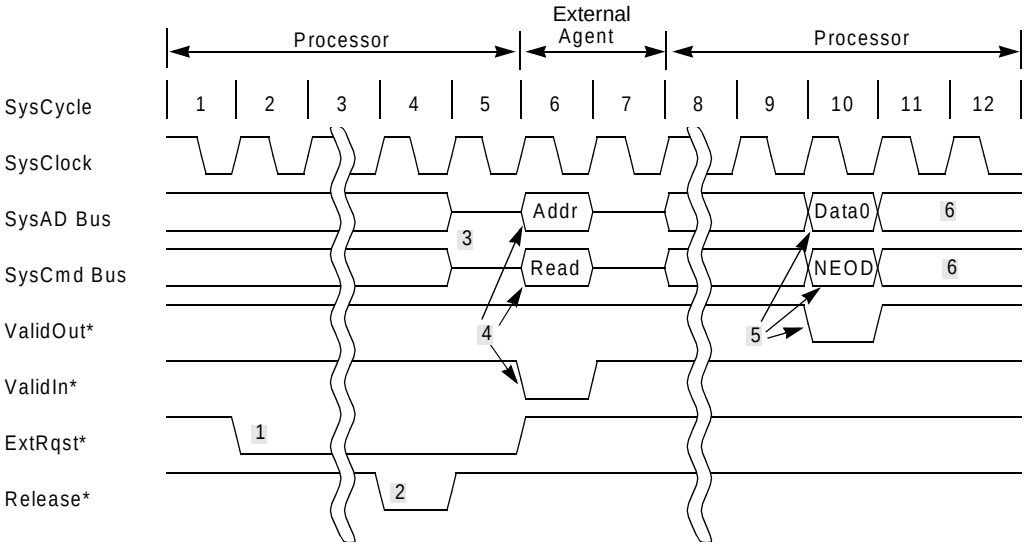


Figure 11-19 External Read Request, System Interface in Master State

Note: The processor does not contain any resources that are readable by an external read request; in response to an external read request, the processor returns undefined data and a data identifier with its *Erroneous Data* bit, *SysCmd* (5), set.

11.6.3 External Null Request Protocol

The processor supports an external null request, which returns the processor system interface to master state from slave state without otherwise affecting th processor.

External null requests require no action from the processor other than to return the system interface to master state.

Figure 11-20 shows a timing diagram of an external null request, which consists of the following steps:

1. The external agent drives an external null request command on the SysCmd bus, and asserts ValidIn* for one cycle to return system interface ownership to the processor.
2. The SysAD bus is unused (does not contain valid data) during the address cycle associated with the external null request.
3. After the address cycle is issued, the null request is complete.

For an external null request, the external agent releases the SysCmd and SysAD buses; the processor should then return to the master state.

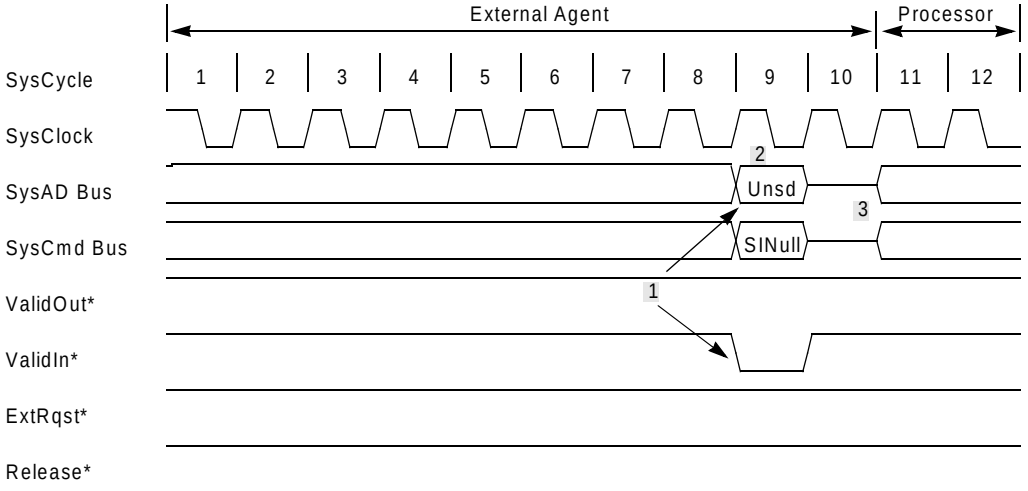


Figure 11-20 System Interface Release External Null Request

11.6.4 External Write Request Protocol

External write requests use a protocol identical to the processor single-word write protocol, except that the ValidIn* signal is asserted instead of ValidOut*. Figure 11-21 shows a timing diagram of an external write request, which consists of the following steps:

1. The external agent asserts ExtRqst* to arbitrate for the system interface.
2. The processor releases the system interface to slave state by asserting Release*.
3. The external agent drives a write command on the SysCmd bus, a write address on the SysAD bus, and asserts ValidIn*.
4. The external agent drives a data identifier on the SysCmd bus, data on the SysAD bus, and asserts ValidIn*.
5. The data identifier associated with the data cycle must contain a last-data-cycle indication.
6. After the data cycle is issued, the write request is complete and the external agent sets the SysCmd and SysAD buses to a tristate, allowing the system interface to return to master state. Timing for the SysADC bus is the same as for the SysAD bus.

External write requests are only allowed to write a word of data to the processor. Processor behavior in response to an external write request for any data element other than a word is undefined.

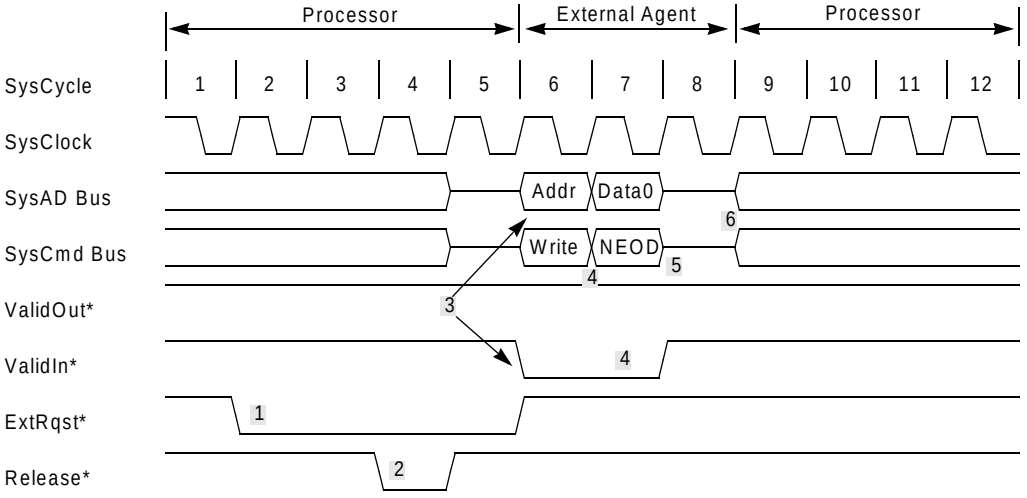


Figure 11-21 External Write Request with System Interface Initially a Bus Master

11.6.5 Read Response Protocol

An external agent must return data to the processor in response to a processor read request by using a read response protocol. A read response protocol consists of the following steps:

1. The external agent waits for the processor to perform an uncompelled change to slave state.
2. The external agent returns the data through a single data cycle or a series of data cycles.
3. After the last data cycle is issued, the read response is complete and the external agent sets the SysCmd and SysAD buses to tristate.
4. The system interface returns to master state.

   Note: The processor always performs an uncompelled change to slave state after issuing a read request. However, the uncompelled change to slave state may happen after more than one transaction in the Multiple-Split-Read timing mode.

5. The data identifier for data cycles must indicate the fact that this data is response data. The data identifier associated with the last data cycle must contain a last-data-cycle indication.

Data must always be returned in the order it was requested for multiple-split-read transactions.

The data identifier associated with a data cycle can indicate that the data transmitted during that cycle is erroneous; however, an external agent must return a data block of the correct size regardless of whether the data may be in error.

Read response data must only be delivered to the processor when a processor read request is pending. The behavior of the processor is undefined when a read response is presented to it and there is no processor read pending.

Figure 11-22 illustrates a processor word read request followed by a word read response. Figure 11-23 illustrates a read response for a processor block read with the system interface already in slave state.

Note: Timing for the SysADC bus is the same as for the SysAD bus.

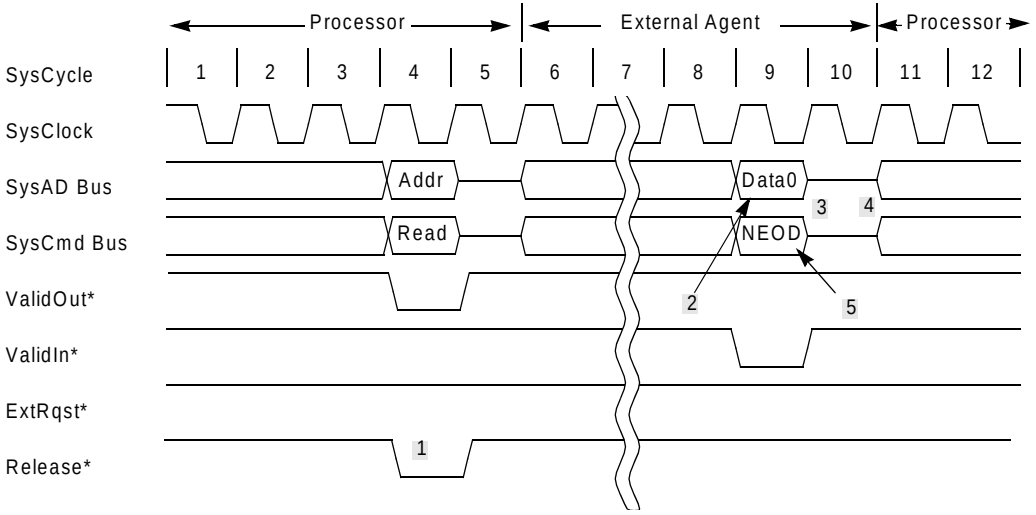


Figure 11-22 Processor Word Read Request Followed by a Word Read Response

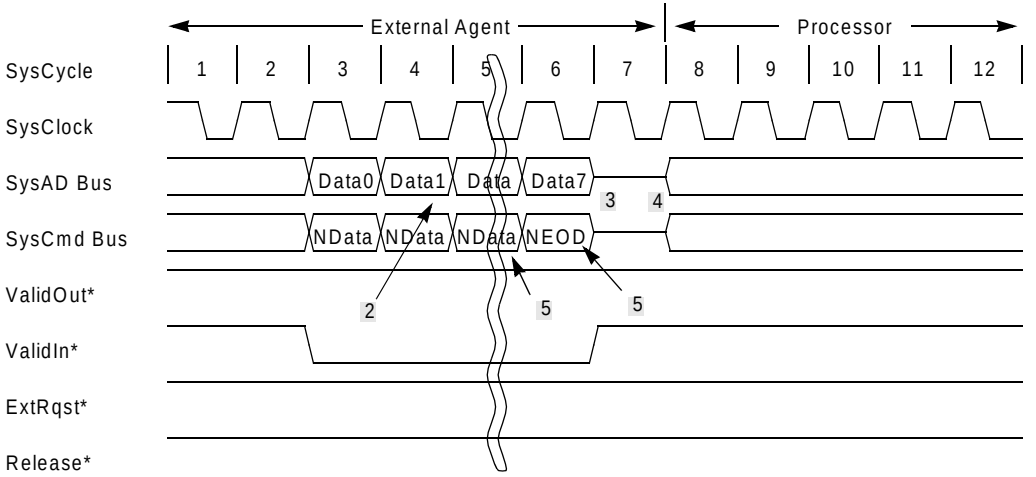


Figure 11-23 Block Read Response, System Interface Already in Slave State

11.6.5.1 Read response in Multiple-Split-Read mode

Multiple-Split-Read timing mode implies that there can be more than one read request issued before an uncompelled change to slave state occurs. Therefore, the VR5432 processor allows the memory subsystem to assert more than one read response cycle before the processor returns to master state. When an uncompelled change to slave state occurs in Multiple-Split-Read mode, the processor will only return to master state when one of the following occurs:

- Data for all outstanding reads is returned.
- An external read, write, or null cycle is issued.

If the read sequence shown in Figure 11-15 occurs, where the first read issued is a block read and the second read issued is a nonblock read, the read response sequence shown in Figure 11-24 can occur. Refer to the steps listed in the previous section.

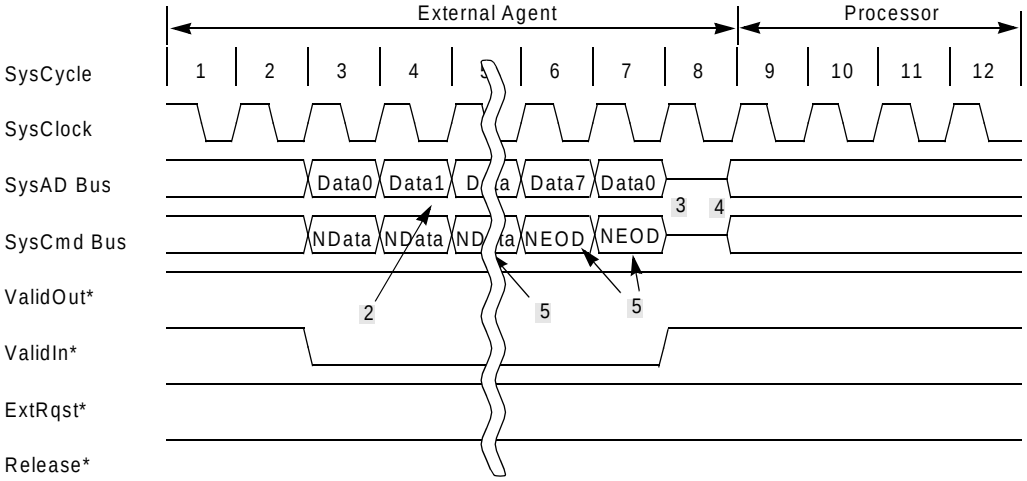


Figure 11-24 Two Consecutive Read Responses in Multiple-Split-Read Mode

Note: ValidIn* controls the flow of data for multiple responses.

If an error occurs during a read response and additional read transactions are outstanding, read responses are still required for all outstanding transactions.

11.7 SysADC (3:0) Protocol

The following rules apply to the use of SysADC (7:0) during a block read response.

- The following actions are taken for each doubleword of the transfer. If data is erroneous, i.e., *SysCmd* (5) = 1, or if parity checking is enabled, i.e., *SysCmd* (4) = 0, and a parity error is detected, the primary cache lines are invalidated. A Bus Error exception is generated when *SysCmd* (5) = 1. A Cache Error exception is generated with the *EE* bit asserted in the CacheErr register when a parity error is detected.
- If a memory error occurs during a block read operation, the SysADC bits should be forced to bad parity for all bytes affected by the memory error during the read response. Forcing bad parity will caus the appropriate cache line to be invalidated

11.8 Data Rate Control

The system interface supports a maximum data rate of one word per cycle. The rate at which data is delivered to the processor can be determined by the external agent—for example, the external agent can drive data and assert ValidIn* every *n* cycles, instead of every cycle. An external agent can deliver data at any rate possible.

The processor only accepts cycles as valid when ValidIn* is asserted and the SysCmd bus contains a data identifier; thereafter, the processor continues to accept data until it receives the data word tagged as the last one. The behavior of the processor is undefined if data is delivered in any pattern other than one valid cycle for nonblock data or two or eight valid cycles for block data.

11.9 Data Transfer Patterns

A data pattern is a sequence of letters indicating the data and unused cycles that repeat to provide the appropriate data rate. For example, the data pattern WWxx specifies a repeatable data rate of two words every four SysClock cycles, with the last two cycles unused. Table 11-2 lists the processor data rate for each of the possible block write modes that may be specified at boot time by setting the *EP* bit of the Config register.

Table 11-2 Transmit Data Rates and Patterns

Maximum Data Rate	Data Pattern
1 Word/1 SysClock Cycle	WWWWWWWW
2 Words/3 SysClock Cycles	WWxWWxWWxWWx
2 Words/4 SysClock Cycles	WWxxWWxxWWxxWWxx
1 Word/2 SysClock Cycles	WxWxWxWxWxWxWxWx
2 Words/5 SysClock Cycles	WWxxxWWxxxWWxxxWWxxx
2 Words/6 SysClock Cycles	WWxxxxWWxxxxWWxxxxWWxxxx
1 Word/3 SysClock Cycles	WxxWxxWxxWxxWxxWxxWxxWxx
2 Words/8 SysClock Cycles	WWxxxxxxWWxxxxxxWWxxxxxxWWxxxxxx
1 Word/4 SysClock Cycles	WxxxWxxxWxxxWxxxWxxxWxxxWxxxWxxx

Note:
W:Data cycle
x :Unused cycle where the data is held on the SysAD bus.

Figure 11-25 shows a read response in which data is provided to the processor at a rate of two words every three cycles, using the data pattern WWx.

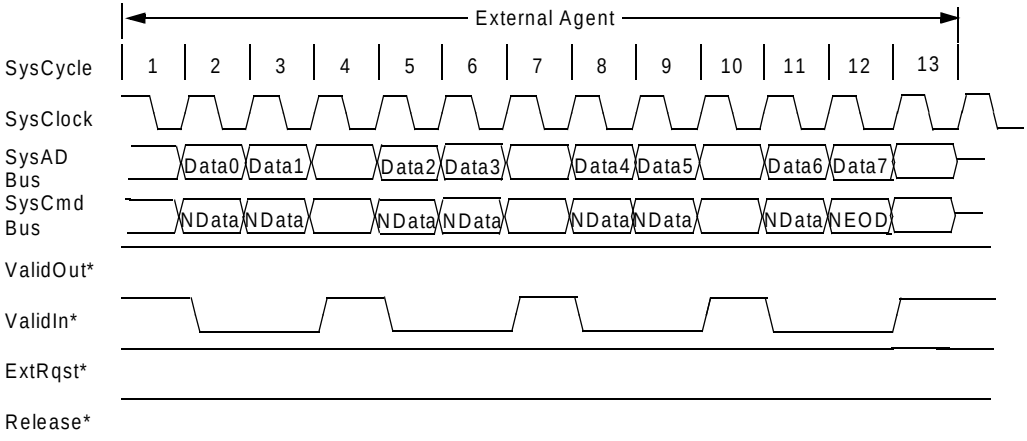


Figure 11-25 Read Response, Reduced Data Rate, System Interface in Slave State

11.10 Word Transfer Ordering

The VR5432 bus transfers a 32-bit address in an address cycle and 32-bit data in a data cycle. When it transfers a block, the VR5432 bus protocol takes two clock cycles for each doubleword transfer. Within the two clocks, the transfer order follows these rules:

- The lower 4 bytes (the low-order word), bits 31:0, of the doublewor are transferred in the first bus clock in Little-Endian mode and in the second bus clock in Big-Endian mode
- The higher 4 bytes (the high-order word), bits 63:32, of the doubleword are transferred in the first bus clock in Big-Endian mod and in the second bus clock in Little-Endian mode

A word transfer or a partial word transfer requires only one bus clock on the VR5432 bus. Table 11-3 compares the block, doubleword, partial doubleword, word, and partial word write schemes in Little-Endian and Big-Endian modes.

Table 11-3 Processor Data Write Orders

Transfer Types	VR5432 (Little Endian)	VR5432 (Big Endian)
Block (D_n[31:0] is the low-order word and D_n[63:32] is the high-order word of the doubleword D_n[63:0]) (A is address)	1. A [31:0], D_0[31:0] 2. D_0[63:32] 3. D_1[31:0] 4. D_1[63:32] 5. D_2[31:0] 6. D_2[63:32] 7. D_3[31:0] 8. D_3[63:32]	1. A [31:0], D_0[63:32] 2. D_0[31:0] 3. D_1[63:32] 4. D_1[31:0] 5. D_2[63:32] 6. D_2[31:0] 7. D_3[63:32] 8. D_3[31:0]
Doubleword	1. A [31:0], D[31:0] 2. D[63:32]	1. A [31:0], D[63:32] 2. D[31:0]
Word (or Partial Word)	1. A [31:0], W[31:0]	1. A [31:0], W[31:0]

When the VR5432 retrieves a cache line from an external agent, the doubleword read order follows the subblock order rule listed in Appendix A. With a doubleword, the two-word transfers follow the rules above. When reading a doubleword, a word, or a partial word, the transfer rules are the same as those for writes. Table 11-4 compares the block, doubleword, word, and partial word read order for VR5432 in Little-Endian and Big-Endian modes.

Table 11-4 Processor Data Read Orders

Transfer Types	VR5432 (Little Endian)	VR5432 (Big Endian)
Block (A[4:3] = 00) (A[2:0] = 000) (D_n[31:0] is the low-order word and D_n[63:32] is the high-order word of the doubleword D_n[63:0])	1. D_0[31:0] 2. D_0[63:32] 3. D_1[31:0] 4. D_1[63:32] 5. D_2[31:0] 6. D_2[63:32] 7. D_3[31:0] 8. D_3[63:32]	1. D_0[63:32] 2. D_0[31:0] 3. D_1[63:32] 4. D_1[31:0] 5. D_2[63:32] 6. D_2[31:0] 7. D_3[63:32] 8. D_3[31:0]
Block (A[4:3] = 01) (A[2:0] = 000)	1. D_1[31:0] 2. D_1[63:32] 3. D_0[31:0] 4. D_0[63:32] 5. D_3[31:0] 6. D_3[63:32] 7. D_2[31:0] 8. D_2[63:32]	1. D_1[63:32] 2. D_1[31:0] 3. D_0[63:32] 4. D_0[31:0] 5. D_3[63:32] 6. D_3[31:0] 7. D_2[63:32] 8. D_2[31:0]
Block (A[4:3] = 10) (A[2:0] = 000)	1. D_2[31:0] 2. D_2[63:32] 3. D_3[31:0] 4. D_3[63:32] 5. D_0[31:0] 6. D_0[63:32] 7. D_1[31:0] 8. D_1[63:32]	1. D_2[63:32] 2. D_2[31:0] 3. D_3[63:32] 4. D_3[31:0] 5. D_0[63:32] 6. D_0[31:0] 7. D_1[63:32] 8. D_1[31:0]

Table 11-4 Processor Data Read Orders (continued)

Transfer Types	VR5432 (Little Endian)	VR5432 (Big Endian)
Block (A[4:3] = 11) (A[2:0] = 000)	1. D_3[31:0] 2. D_3[63:32] 3. D_2[31:0] 4. D_2[63:32] 5. D_1[31:0] 6. D_1[63:32] 7. D_0[31:0] 8. D_0[63:32]	1. D_3[63:32] 2. D_3[31:0] 3. D_2[63:32] 4. D_2[31:0] 5. D_1[63:32] 6. D_1[31:0] 7. D_0[63:32] 8. D_0[31:0]
Doubleword	1. D[31:0] 2. D[63:32]	1. D[63:32] 2. D[31:0]
Word (or Partial Word)	1. W[31:0]	1. W[31:0]

The external agent can only write one word to the VR5432 at a time (see Figure 11-21), so it takes one bus clock to transfer the word from the external agent to the VR5432.

11.11 Independent Transmissions on the SysAD Bus

In most applications, the SysAD bus is a point-to-point connection, running from the processor to a bidirectional registered transceiver residing in an external agent. For these applications, the SysAD bus has only two possible drivers, the processor or the external agent.

Certain applications may require connection of additional drivers and receivers to the SysAD bus, to allow transmissions over the SysAD bus with which the processor is not involved. These are called independent transmissions. To effect an independent transmission, the external agent must coordinate control of the SysAD bus by using arbitration handshake signals and external null requests.

An independent transmission on the SysAD bus must follow this procedure:

1. The external agent requests control of the SysAD bus, to issue an external request.
2. The processor releases the system interface to slave state.
3. The external agent then allows the independent transmission to take place on the SysAD bus, making sure that ValidIn* is not asserted while the transmission is occurring.
4. When the transmission is complete, the external agent must issue a system interface release external null request to return the system interface to master state.

11.12 System Interface Cycle Time

The processor specifies latency for various processor transactions and for the processor response time to external requests as follows. Processor requests themselves are constrained by the system interface request protocol, and request cycle counts can be determined by examining the protocol. The following system interface interactions can also occur in a VR5432 system:

- Waiting period for the processor to release the system interface to slave state in response to an external request (release latency)
- Response time for an external request that requires a respons (external response latency)

Release latency depends on the state of the system interface during the time of the external request. The system interface will be released to the external agent following the set of bus cycles in which the external request was detected. The external request is detected two system bus cycles after it is asserted.

External response latency will be minimal for the VR5432. Write data will be taken immediately, and read requests will be given an error indication on the *SysCmd* (5) bit.

11.13 System Interface Commands/Data Identifiers

System interface commands specify the nature and attributes of any system interface request; this specification is made during the address cycle for the request. System interface data identifiers specify the attributes of data transmitted during a system interface data cycle.

The following sections describe the syntax (the bitwise encoding) of system interface commands and data identifiers.

For system interface commands and data identifiers associated with processor requests, reserved bits and reserved fields in the command and data identifiers are undefined.

11.13.1 Command and Data Identifier Syntax

System interface commands and data identifiers are encoded in nine bits and are transmitted on the SysCmd bus from the processor to an external agent, or from an external agent to the processor, during address and data cycles. Bit 8 (the most-significant bit) of the SysCmd bus determines whether the current content of the SysCmd bus is a command or a data identifier and, therefore, whether the current cycle is an address cycle or a data cycle. For system interface commands, *SysCmd* (8) must be set to 0. For system interface data identifiers, *SysCmd* (8) must be set to 1.

11.13.2 System Interface Command Syntax

This section describes the SysCmd bus encoding for system interface commands. Figure 11-26 shows a common encoding used for all system interface commands.

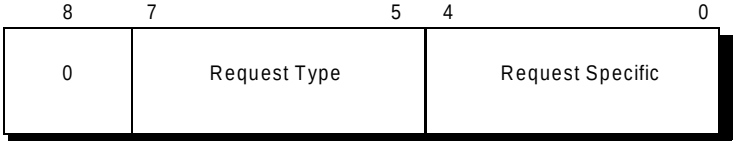


Figure 11-26 System Interface Command Syntax Bit Definition

SysCmd (8) must be driven as 0 for all system interface commands.

SysCmd (7:5) specify the system interface request type, which may be read, write, or null. Table 11-5 shows the types of requests encoded by the *SysCmd* (7:5) bits.

Table 11-5 Encoding of SysCmd (7:5) for System Interface Commands

SysCmd (7:5)	Command
0	Read request
1	Reserved
2	Write request
3	Null request
4–7	Reserved

SysCmd (4:0) are specific to each type of request and are defined in each of the following sections.

11.13.3 Read Requests

Figure 11-27 shows the format of a SysCmd read request.

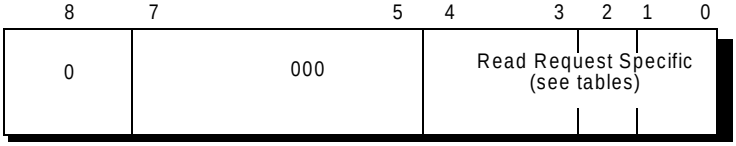


Figure 11-27 Read Request SysCmd Bus Bit Definition

Table 11-6 through Table 11-8 list the encodings of *SysCmd* (4:0) for read requests.

Table 11-6 Encoding of SysCmd (4:3) for Read Requests

SysCmd (4:3)	Read Attributes
0, 1	Reserved
2	Block read
3	Word or partial word

Table 11-7 Encoding of SysCmd (2:0) for Block Read Request

SysCmd (2)	Reserved
SysCmd (1:0)	**Read Block Size**
0	2 words
1	8 words
2, 3	Reserved

Table 11-8 Read Request Data Size Encoding of SysCmd (2:0)

SysCmd (2)	Reserved
SysCmd (1:0)	**Read Block Size**
0	1 byte valid (Byte)
1	2 bytes valid (Halfword)
2	3 bytes valid (Tribyte)
3	4 bytes valid (Word)

11.13.3.1 Write requests

Figure 11-28 shows the format of a SysCmd write request.

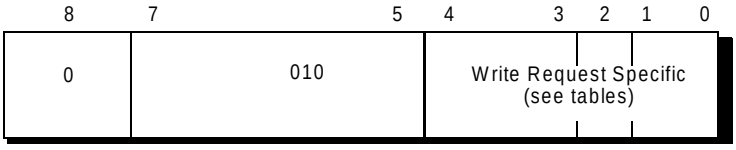


Figure 11-28 Write Request SysCmd Bus Bit Definition

Table 11-9 lists the write attributes encoded in bits *SysCmd* (4:3).

Table 11-9 Write Request Encoding of SysCmd (4:3)

SysCmd (4:3)	Write Attributes
0	Reserved
1	Reserved
2	Block write
3	Word or partial word

Table 11-10 lists the block write replacement attributes encoded in bits *SysCmd* (2:0).

Table 11-10 Block Write Request Encoding of SysCmd (2:0)

SysCmd (2)	Cache Line Replacement Attributes (8 words only)
0	Cache line replaced
1	Cache line retained
SysCmd (1:0)	**Write Block Size**
0	2 words; *SysCmd* (2) is reserved for this encoding
1	8 words
2, 3	Reserved

Table 11-11 lists the write request bit encodings in *SysCmd* (2:0).

Table 11-11 Write Request Data Size Encoding of SysCmd (2:0)

SysCmd (2)	Reserved
SysCmd (1:0)	**Write Data Size**
0	1 byte valid (byte)
1	2 bytes valid (halfword)
2	3 bytes valid (tribyte)
3	4 bytes valid (word)

11.13.3.2 Null requests

Figure 11-29 shows the format of a SysCmd null request.

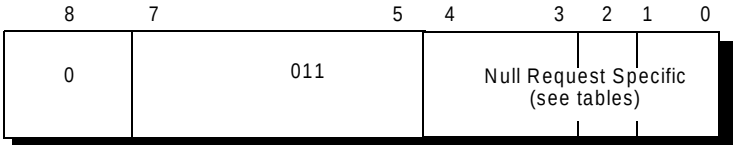


Figure 11-29 Null Request SysCmd Bus Bit Definition

System interface release external null requests use the null request command. Table 11-12 lists the encodings of *SysCmd* (4:3) for external null requests. *SysCmd* (2:0) are reserved for null requests.

Table 11-12 External Null Request Encoding of SysCmd (4:3)

SysCmd (4:3)	Null Attributes
0	System interface release
1–3	Reserved

11.13.4 System Interface Data Identifier Syntax

This section defines the encoding of the SysCmd bus for system interface data identifiers. Figure 11-30 shows encoding typically used for all system interface data identifiers.

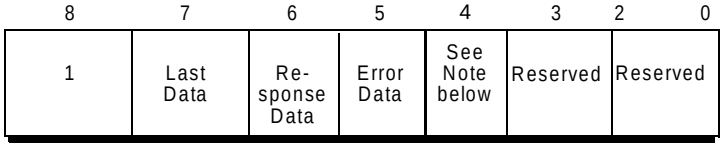


Figure 11-30 Data Identifier SysCmd Bus Bit Definition

SysCmd (8) must be set to 1 for all system interface data identifiers.

Note: *SysCmd* (4) is reserved for the processor data identifier. In an external data identifier, *SysCmd* (4) indicates whether or not to check the data and check bits for errors.

11.13.4.1 Data identifier bit definitions

SysCmd (7) marks the last data element and *SysCmd* (6) indicates whether the data is response data, for both processor and external data identifiers. Response data is data returned in response to a read request.

SysCmd (5) indicates whether the data element is error free. Erroneous data contains an error and is returned to the processor, forcing a bus error. In the case of a block response, the entire line must be delivered to the processor, no matter how minimal the error. The processor will not indicate erroneous data. The external agent should ignore this bit.

SysCmd (4) indicates to the processor whether to check the data and check bits for this data element.

SysCmd (3) is reserved for external data identifiers.

SysCmd (4:3) are reserved for processor data identifiers.

SysCmd (2:0) are reserved for data identifiers.

Table 11-13 lists the encodings of *SysCmd* (7:0) for processor data identifiers.

Table 11-13 Processor Data Identifier Encoding of SysCmd (7:0)

SysCmd (7)	Last Data Element Indication
0	Last data element
1	Not the last data element
SysCmd (6)	**Response Data Indication**
0	Data is response data
1	Data is not response data
SysCmd (5)	**Good Data Indication**
0	Data is error free
1	Data is erroneous
SysCmd (4:0)	**Reserved**

Table 11-14 lists the encodings of *SysCmd* (7:0) for external data identifiers.

Table 11-14 External Data Identifier Encoding of SysCmd (7:0)

SysCmd (7)	Last Data Element Indication
0	Last data element
1	Not the last data element
SysCmd (6)	**Response Data Indication**
0	Data is response data
1	Data is not response data
SysCmd (5)	**Good Data Indication**
0	Data is error free
1	Data is erroneous
SysCmd (4)	**Data Checking Enable**
0	Check the data and check bits
1	Do not check the data and check bits
SysCmd (3:0)	**Reserved**

11.14 System Interface Addresses

System interface addresses are full 32-bit physical addresses presented on the 32 bits of the SysAD bus during address cycles. All bits of the SysAD bus are used during address cycles.

11.14.1 Addressing Conventions

Addresses associated with doubleword, partial doubleword, word, or partial word transactions and update requests are aligned for the size of the data element. The system uses the following address conventions:

- Addresses associated with block requests are aligned to doubleword boundaries; that is, the low-order 3 bits of the address are 0
- Doubleword requests set the low-order 3 bits of the address to 0.
- Word requests set the low-order 2 bits of the address to 0.
- Halfword requests set the low-order bit of the address to 0
- Byte and tribyte requests use the byte address

11.14.2 Subblock Ordering

The order in which data is returned in response to a processor block read request is called “subblock ordering.” In subblock ordering, the processor delivers the address of the requested doubleword within the block. An external agent must return the block of data using subblock ordering, starting with the addressed doubleword.

For block write requests, the processor always delivers the address of the doubleword at the beginning of the block; the processor delivers data beginning with the doubleword at the beginning of the block and progresses sequentially through the doublewords that form the block. See Appendix A for more detail.

During data cycles, the valid byte lines depend upon the position of the data with respect to the aligned doubleword (this may be a byte, halfword, tribyte, or quadbyte/word). For example, in Little-Endian mode, on a byte request where the address module 8 is 0, *SysAD* (7:0) are valid during the data cycles. Table 11-15 lists the byte lanes used for partial word transfers for both big- and little-endian formats.

Table 11-15 Partial Word Transfer Byte Lane Usage

# Bytes SysCmd(1:0)	Address Mod 4	SysAD Byte Lanes Used							
		Little Endian				Big Endian			
		31:24	23:16	15:8	7:0	31:24	23:16	15:8	7:0
1 (00)	0				X	X			
	1			X			X		
	2		X					X	
	3	X							X
2 (01)	0			X	X	X	X		
	2	X	X					X	X
3 (10)	0		X	X	X	X	X	X	
	1	X	X	X			X	X	X
4 (11)	0	X	X	X	X	X	X	X	X

11.14.3 Processor Internal Address Map

External reads and writes provide access to processor internal resources that may be useful to an external agent. The processor decodes *SysAD* (6:4) of the address associated with an external read or write request to determine which processor internal resource is the target. However, the processor does not contain any resources that are readable through an external read request. Therefore, in response to an external read request, the processor returns undefined data and a data identifier with its *Erroneous Data* bit, *SysCmd* (5), set. The Interrupt register is the only processor internal resource available for write access by an external request. The Interrupt register is accessed by an external write request with an address of 000_2 for the processor.

System Interface Transactions (R43K Mode)

12

This section describes processor and external requests as they occur in R43K (VR4300 compatibility) mode (the OptionR43K* input signal sampled low during cold reset). The transactions used in the native system interface mode are described in Chapter 10.

All system interface transactions using the VR5432 processor are noncoherent; there is no hardware cache coherency support provided. Requests fall into three categories:

- Cached
- Uncached
- Uncached accelerated

12.1 Processor Requests

When a system event occurs, the processor issues either a single request or a series of requests (called processor requests) through the system interface, to access an external resource and service the event. For this to work, the processor system interface must be connected to an external agent that is compatible with the system interface protocol and can coordinate access to system resources.

An external agent requesting access to a processor internal resource generates an external request. This access request passes through the system interface. System events and request cycles are shown in Figure 12-1.

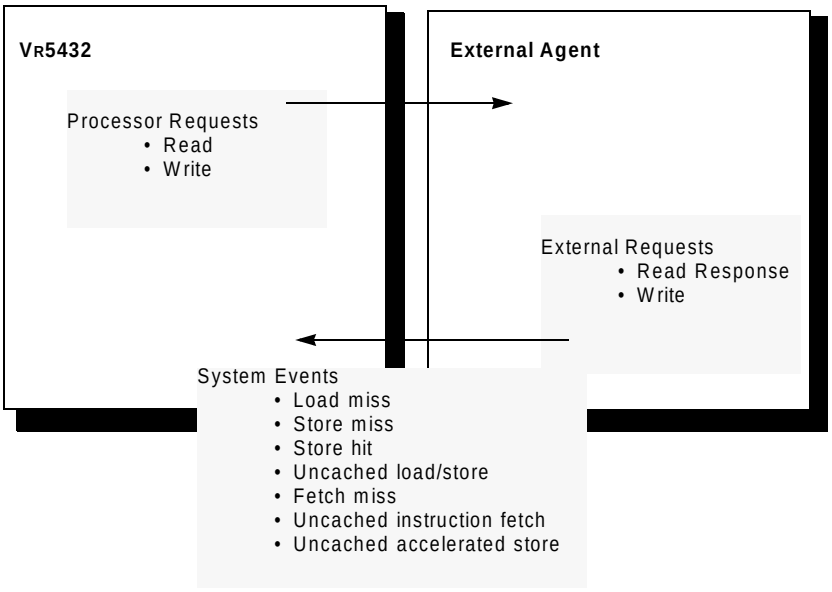


Figure 12-1 Requests and System Events

12.1.1 Rules for Processor Requests

A processor request is a request, through the system interface, to access some external resource. As shown in Figure 12-2, processor requests include read and write.

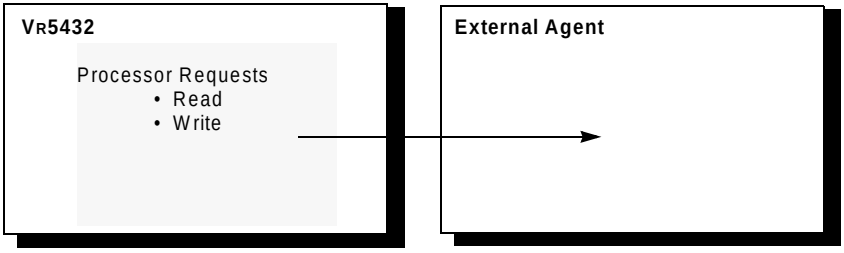


Figure 12-2 Processor Requests to External Agent

A read request asks for a block, doubleword, partial doubleword, word, or partial word of data either from main memory or from another system resource. A read request supplies an address to an external agent.

A write request provides a block, doubleword, partial doubleword, word, or partial word of data to be written either to main memory or to another system resource. A write request supplies an address and a block, doubleword, partial doubleword, word, or partial word of data to an external agent.

Processor read requests that have been issued, but for which data has not yet been returned, are said to be pending. The processor will not issue another request while a read is already pending. A processor read request is said to be complete after the last transfer of response data has been received from an external agent. A processor write request is said to be complete after the last word of data has been transmitted.

The processor input signal EOK* allows an external agent to manage the flow of processor requests. EOK* controls the flow of processor read and write requests.

The processor request cycle sequence is shown in Figure 12-3.

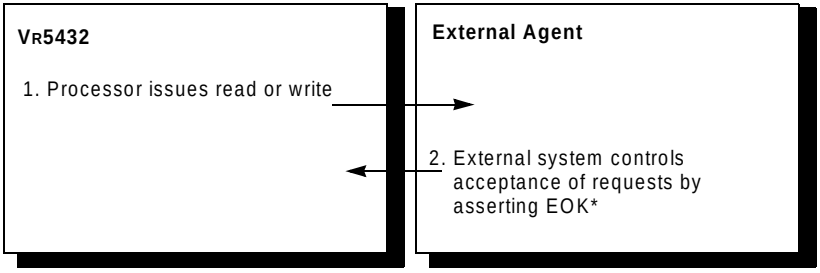


Figure 12-3 Processor Request Flow Control

12.1.2 Processor Read Request

When the VR5432 processor issues a read request, the external agent must access the specified resource and return the requested data.

A processor read request can be split from the external agent's return of the requested data. In other words, the external agent can initiate an unrelated external request before it returns the response data for a processor read. A processor read request is completed after the last word of response data has been received from the external agent.

Processor read requests that have been issued, but for which data has not yet been returned, are said to be *pending*. A read remains pending until the requested read data is returned.

Note that the data identifier associated with the response data can indicate that the response data is erroneous, causing the processor to generate a Bus Error exception.

The external agent must be capable of accepting a new processor read request at any time when the following two conditions are met:

- There is no processor read request pending.
- The EOK* signal has been asserted for two or more cycles

12.1.3 Processor Write Request

When a processor issues a write request, the specified resource is accessed and the data is written to it. A processor write request is complete after the last word of data has been transmitted to the external agent.

The external agent must be capable of accepting a processor write request any time the following two conditions are met:

- No processor read request is pending
- The EOK* signal has been asserted for two or more cycles

12.2 External Requests

External requests include read response and write requests, as shown in Figure 12-4.

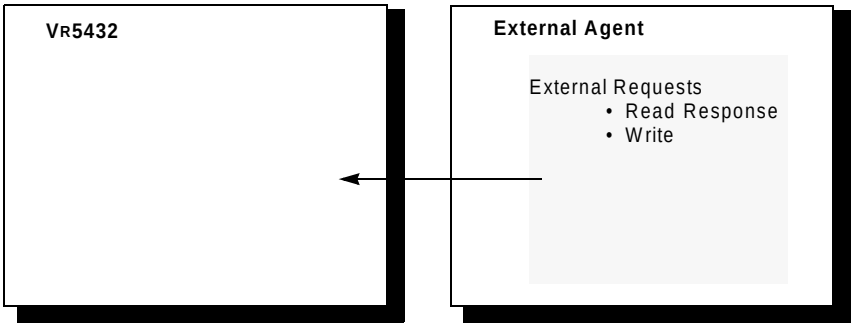


Figure 12-4 External Requests to Processor

A read response returns data in response to a processor read request.

A write request provides a word of data to be written to the processor's internal resource.

The processor controls the flow of external requests through the arbitration signals EReq* and PMaster*, as shown in Figure 12-5. The external agent must acquire mastership of the system interface before it is allowed to issue an external request. The external agent arbitrates for mastership of the system interface by asserting EReq* and then waits for the processor to deassert the PMaster* signal.

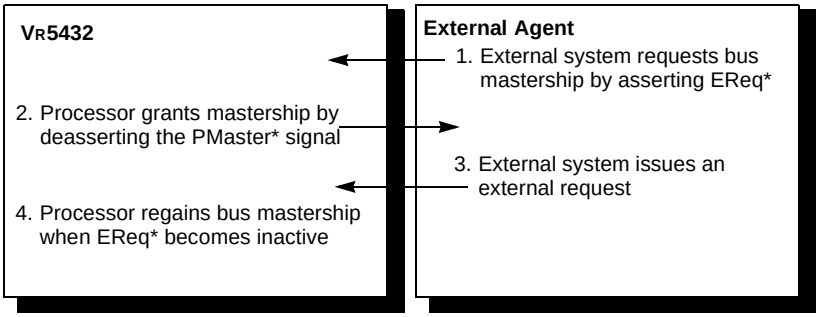


Figure 12-5 External Request Arbitration

Mastership of the system interface always returns to the processor when the EReq* signal becomes inactive after an external request is issued. The processor does not accept a subsequent external request until it has completed the current request.

If there are no processor requests pending, the processor determines, based on its internal state, whether to accept the external request or issue a new processor request. The processor can issue a new processor request even if the external agent is requesting access to the system interface.

The external agent asserts the EReq* signal to indicate that it wishes to begin an external request. The processor releases mastership of the system interface by deasserting the PMaster* signal. An external request can be accepted based on the following criteria:

- The processor completes any processor request in execution
- While the processor is waiting for the assertion of the EOK* signal to issue a processor read/write request, the EReq* signal is input to th processor one or more cycles before the EOK* signal is asserted.
- If the processor is waiting for the response to a read request after the processor has made an uncompelled change to a slave state (th external agent can issue an external request before providing the rea response data).

12.2.1 External Write Request

When an external agent issues a write request, the specified resource is accessed and the data is written to it. An external write request is complete after the word of data has been transmitted to the processor.

The only processor resource available to an external write request is the Interrupt register.

12.2.2 Read Response

A read response returns data in response to a processor read request, as shown in Figure 12-6. While a read response is technically an external request, it has one characteristic that differentiates it from all other external requests: it does not perform system interface arbitration (requesting mastership of the system interface using EReq*). For this reason, read responses are handled separately from all other external requests, and are simply called read responses.

The data identifier associated with the response data can signal that the returned data is erroneous, causing the processor to make a bus error.

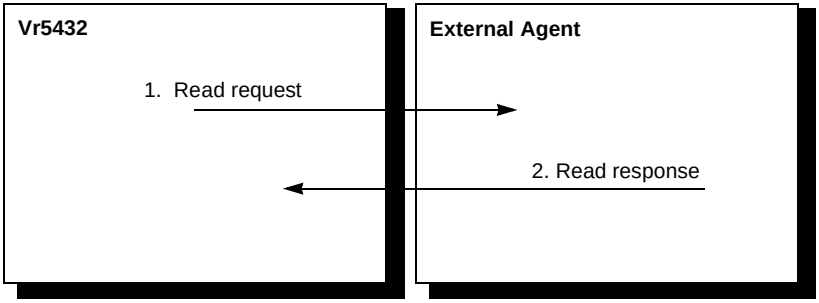


Figure 12-6 External Agent Read Response to Processor

12.3 Handling Requests

This section details the sequence of both processor and external requests. The following system events are discussed:

- Fetch miss
- Load miss
- Store miss
- Store hit
- Uncached loads/stores
- Uncached accelerated stores
- Uncached instruction fetch

12.3.1 Fetch Miss

When the processor misses in the instruction cache during an instruction fetch, it issues a read request for cache line acquisition. An external agent returns data as a read response.

12.3.2 Load Miss

When a processor load misses in the primary cache, before the processor can proceed it must obtain the cache line that contains the data element to be loaded from the external agent.

If the new cache line replaces a current dirty cache line, the current cache line must be written back before the new line can be loaded in the primary cache.

Table 12-1 shows the actions taken on a load miss to the primary cache.

Table 12-1 Load Miss to Primary Cache

Page Attribute	State of Data Cache Line Being Replaced	
	Clean/Invalid	Dirty (D = 1)
Cached	BR	BR/BW

BR: Processor block read request
BR/BW: Processor block read request followed by processor block write request

The processor takes the following steps:

1. The processor issues a block read request for the cache line that contains the data element to be loaded.
2. The processor waits for an external agent to provide the read response.
3. The processor completes the load after the first doubleword of the data cache miss is received. The remaining three doublewords are placed in the cache as they arrive.

If the cache data which the incoming line will replace contains valid dirty data, this data is written to memory. The read completes before the write of the dirty cast-out data.

12.3.3 Store Miss

When a processor store misses in the primary cache, the processor will request, from the external agent, the cache line that contains the target location of the store.

The processor then executes one of the following actions:

1. The processor issues a block read request for the cache line that contains the data element to be loaded.
2. The processor then waits for an external agent to provide the read response.
3. The cache line is loaded into the cache, and the store data is merged into the appropriate location.
4. If the page attribute is write-through, a nonblock write request is issued.

Table 12-2 shows the actions taken on a store miss to the primary cache.

Table 12-2 Store Miss to Primary Cache

Page Attribute	State of Data Cache Line Being Replaced	
	Clean/Invalid	Dirty (W = 1)
Write-back	BR	BR/BW
Write-through	BR/W	BR/BW/W

BR: Processor block read request for missed cache line
BW: Processor block write request for replaced dirty data
W: Processor nonblock write request for write-through data

If the page attribute is write-back or write-through, the processor issues a block read request for the cache line that contains the data element to be loaded. If a cache line must be written back due to replacement, the processor issues a write request for that cache line. If the page attribute is write-through, the processor issues a nonblock write request for the new data.

12.3.4 Store Hit

The action on the system bus is determined by whether the line is write-back or write-through. Write-back store hits cause no bus transactions. For lines with a write-through policy, the store generates a processor nonblock write request for the store data.

12.3.5 Uncached Loads or Stores

When the processor performs an uncached load, it issues a doubleword, partial doubleword, word, or partial word read request. When the processor performs an uncached store, it issues a doubleword, partial doubleword, word, or partial word write request. All writes by the processor are buffered from the system interface by a four-deep transaction buffer. Since this buffer behaves as a FIFO, previous write requests in the buffer are completed before a following read request is serviced.

12.3.6 Uncached Accelerated Stores

Uncached accelerated operations are uncached operations to a page with an uncached accelerated cache algorithm. When the processor performs an uncached accelerated store, it can perform a block write, or it can perform one or more doubleword, partial doubleword, word, or partial word write requests. All writes by the processor are buffered from the system interface by a four-deep transaction buffer. Since this buffer behaves as a FIFO, previous write requests in the buffer are completed before a following read request is serviced.

Uncached, accelerated operations allow the user to combine several sequential uncached word or doubleword operations into a single 32-byte block of write data and only generate a single external SysAD bus transaction. In order for the programmer to optimize these transactions, special attention should be paid to the uncached accelerated data alignment and gathering rules.

Uncached accelerated writes pass through the transaction buffer in FIFO order, the same as all other types of transactions. However, successive uncached accelerated transactions are assembled into up to four doubleword FIFO entries if the uncached accelerated write gathering rules are followed. The rules are:

- The initial uncached accelerated transaction must be aligned on mod 32-byte boundary
- All uncached accelerated transactions must be word or doublewor sized.
- Word and doubleword transactions must be naturally aligned
- Word writes must happen in pairs in order to form a naturally aligned doubleword.
- Addresses must increase sequentially.

Uncached accelerated write transactions that do not follow the rules stated above will be treated as nonaccelerated uncached transactions. In addition, if a gathering sequence is disrupted by any operation other than those making up the uncached accelerated gather, the portion of the data that is gathered will be sent out as uncached word or doubleword operations.

The uncached accelerated operation will also be disrupted if the processor enters Debug mode. When Debug mode is entered, the transaction buffer is emptied. In addition, it is likely that the gather operation will be disrupted if an exception is taken.

12.3.7 Uncached Instruction Fetch

The processor issues word reads for instruction fetches to uncached addresses. Word reads must conform to the byte alignment as indicated by the size encoded on *SysCmd* (1:0). Therefore, any system ROM address space accessed during a processor boot restart must support properly aligned 32-bit reads.

System Interface Protocols (R43K Mode)

13

The following sections contain a cycle-by-cycle description of the system interface protocols for each type of processor and external request in R43K (VR4300 compatibility) mode of the VR5432 processor. For the protocols followed in the native mode of the VR5432 processor, see Chapter 11.

13.1 Address and Data Cycles

Cycles in which the SysAD bus contains a valid address are called address cycles. Cycles in which the SysAD bus contains valid data are called data cycles. Validity of addresses and data from the processor is determined by the state of the PValid* signal. Validity of the address and data from the external agent is determined by the state of the EValid* signal.

The SysCmd bus identifies the contents of the SysAD bus during any cycle in which it is valid from the processor or the external agent. The most-significant bit of the SysCmd bus is always used to indicate whether the current cycle is an address cycle or a data cycle.

- During address cycles, i.e., *SysCmd* (4) = 0, the remainder of the SysCmd bus *SysCmd* (3:0), contains the encoded system interface command.
- During data cycles, i.e., *SysCmd* (4) = 1, the remainder of the SysCmd bus *SysCmd* (3:0), contains an encoded data identifier.

When the processor is driving the SysAD (31:0) and SysCmd (4:0) buses, the system interface is in *master state*. When the external agent is driving these buses, the system interface is in *slave state*.

- When the processor is in master state and the SysAD (31:0) and SysCmd (4:0) buses are valid, the processor asserts the PValid* signal.
- When the processor is in slave state and the SysAD (31:0) and SysCmd (4:0) buses are valid, an external agent asserts the EValid* signal.

13.2 Issue Cycles

There are two types of processor issue cycles:

- Processor read reques
- Processor write reques

The issue cycle of the processor read/write request is determined by the state of the EOK* signal. The issue cycle is a cycle that becomes valid in the address cycle of each processor request. Only one issue cycle exists for each processor request.

To define the issue cycle of the address cycle, assert the EOK* signal active at the external agent side one cycle before the address cycle of the processor read/write request, as shown in Figure 13-1.

To define the address cycle as the issuance cycle, do not deassert the EOK* signal inactive until the address cycle is started.

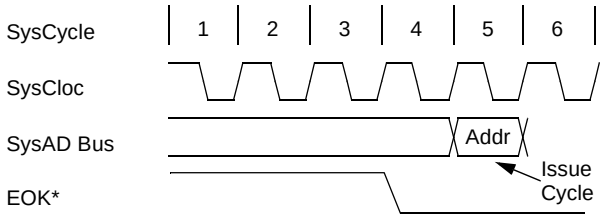


Figure 13-1 EOK Signal Status of Processor Request*

The processor repeats the address cycle for the request until the conditions for a valid issue cycle are met. Figure 13-2 illustrates how the address cycle is extended by the EOK* signal. There is only one issue cycle for any processor request.

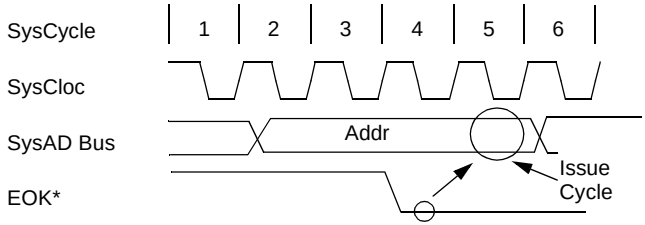


Figure 13-2 Address Cycle Extended by EOK Signal*

The processor accepts external requests, even while attempting to issue a processor request, by releasing the system interface to slave state in response to an assertion of EReq* by the external agent.

The rules governing the issue cycle of a processor request are strictly applied to determine which action the processor takes. The processor can either:

- Complete the issuance of the processor request in its entirety befor the external request is accepted, o
- Release the system interface to slave state without completing th issuance of the processor request

In the latter case, the processor issues the processor request (provided the processor request is still necessary) after the external request is completed. The rules governing an issue cycle again apply to the processor request.

13.3 Handshake Signals

The VR5432 processor manages the flow of requests through the following six control signals:

- EOK*—This signal is used by the external agent to indicate whethe it can accept a new read or write transaction.
- EReq*, PMaster*, PReq*—These signals are used to transfer control of the SysAD and SysCmd buses. EReq* is used by an external agent to indicate a need to control the interface. PMaster* is deasserted by the processor when it transfers the mastership of the system interfac to the external agent. PReq* is used by the processor to request th external agent to release control of the system interface.
- PValid*, EValid*—The VR5432 processor uses PValid* and th external agent uses EValid* to indicate valid commands/data on the SysCmd and SysAD buses.

13.4 System Interface Operation

Figure 13-3 shows how the system interface operates from register to register. Processor outputs come directly from output registers and begin to change on the rising edge of SysClock.

Processor inputs are fed directly to input registers that latch these input signals on the rising edge of SysClock. This option allows the system interface to run at the highest possible clock frequency.

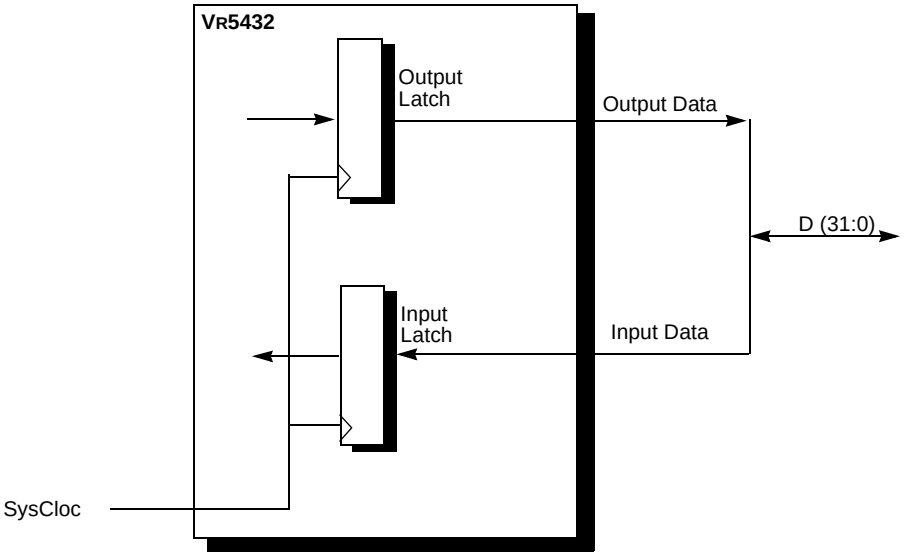


Figure 13-3 System Interface Register-to-Register Operation

13.4.1 Master and Slave States

When the VR5432 processor is driving the SysAD and SysCmd buses, the system interface is in master state. When the external agent is driving the SysAD and SysCmd buses, the system interface is in slave state. In master state, the processor asserts PValid* whenever the SysAD and SysCmd buses are valid. In slave state, the external agent asserts EValid* whenever the SysAD and SysCmd buses are valid.

The system interface remains in master state unless:

- The external agent requests and is granted the system interface (external arbitration).
- The processor issues a read request and deasserts PMaster* as an uncompelled change to slave state

13.4.2 External Arbitration

External arbitration is performed by the processor. The system interface must be in slave state for the external agent to issue an external request through the system interface. The transition from master state to slave state is arbitrated by the master processor using the system interface handshake signals EReq* and PMaster*. This transition is described by the following sequence:

1. An external agent signals the need to issue an external request by asserting EReq*.
2. When the processor is ready to accept an external request, it releases the system interface from master to slave state by deasserting PMaster*.
3. The system interface returns to master state as soon as the external request is completed.

13.4.3 Uncompelled Change to Slave State

An uncompelled change to slave state is the transition of the system interface from master state to slave state, initiated by the processor when a processor read request is pending. PMaster* is deasserted automatically after a read request, and an uncompelled change to slave state then occurs. This transition to slave state allows the external agent to return read response data without arbitrating for bus ownership. After an uncompelled change to slave state, the processor returns to master state at the end of the next external request. This can be a read response or some other type of external request.

When the processor returns from the uncompelled transition depends on the cache status. The processor returns to the master state when the following external request (read response or other external request) is completed after the uncompelled transition to the slave state.

An external agent must note that the processor has performed an uncompelled change to slave state and begin driving the SysAD bus along with the SysCmd bus. As long as the system interface is in slave state, the external agent can begin an external request without arbitrating for the system interface; that is, without asserting EReq*.

If EReq* is inactive at the time the external request is completed, the system interface automatically returns to master state.

Table 13-1 lists the abbreviations and definitions for each of the buses that are used in the timing diagrams that follow.

Table 13-1 System Interface Requests

Scope	Abbreviation	Meaning
Global	Unsd	Unused
SysAD (31:0) bus	Addr	Physical address
	Data<n>	Data element number n of a block of data
SysCmd (4:0) bus	Cmd	An unspecified system interface command
	Read	A processor or external read request command
	Write	A processor or external write request command
	NData	A data identifier for a data element other than the last data element
	NEOD	A data identifier for the last data element

13.5 Processor Request Protocols

Processor request protocols described in this section include:

- Read
- Writ

Note: In the timing diagrams, the two closely spaced, wavy vertical lines, such as those shown in Figure 13-4, indicate one or more identical cycles that are not illustrated due to space constraints.

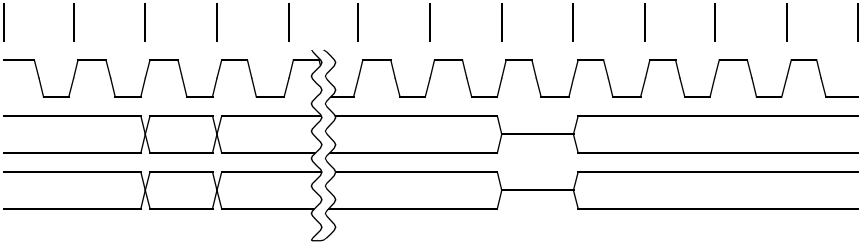


Figure 13-4 Symbol for Undocumented Cycles

13.5.1 Processor Read Request Protocol

A processor read request is issued by outputting a read command on the SysCmd (4:0) bus, a read address on the SysAD (31:0) bus, and asserting PValid*. Only one processor read request may be pending at a time; the processor must wait for an external read response before starting a subsequent read request.

The processor makes an uncompelled change to slave state after the read request cycle by deasserting the PMaster* signal. An external agent then returns the requested data through a read response.

Once the processor enters slave state (starting at cycle 5 in Figure 13-5), the external agent can return the requested data through a read response. The read response returns the requested data or, if the requested data could not be successfully retrieved, indicates on the *SysCmd* (4:0) bus that the returned data is erroneous. If the returned data is erroneous, the processor generates a Bus Error exception.

Figure 13-5 illustrates a processor read request, followed by an uncompelled change to slave state, that occurs as the read request is issued. Figure 13-6 shows the processor read request delayed by the EOK* signal.

The following sequence describes the protocol for a processor read request (the numbered steps below correspond to Figure 13-5 and Figure 13-6).

1. The processor is in the master state. It outputs a read command to *SysCmd* (4:0) and a read address to *SysAD* (31:0) to issue a read request. After the read request is issued, the processor enters the suspended state. Only one read request can be suspended at a time.
2. The processor asserts the PValid* signal to indicate that the current data on *SysCmd* (4:0) and *SysAD* (31:0) is valid.
3. The external agent asserts the EOK* signal for two consecutive cycles to enable issuance of a processor read request. If the EOK* signal is deasserted, the issuance cycle of the read request is delayed.
4. The processor deasserts the PMaster* signal on the first cycle after the read request is accepted, followed by an uncompelled shift to slave state.
5. The processor releases *SysCmd* (4:0) and *SysAD* (31:0) at the same time as the PMaster* signal is deasserted.
6. An external agent can drive *SysCmd* (4:0) and *SysAD* (31:0) from the first cycle after the PMaster* signal is deasserted.

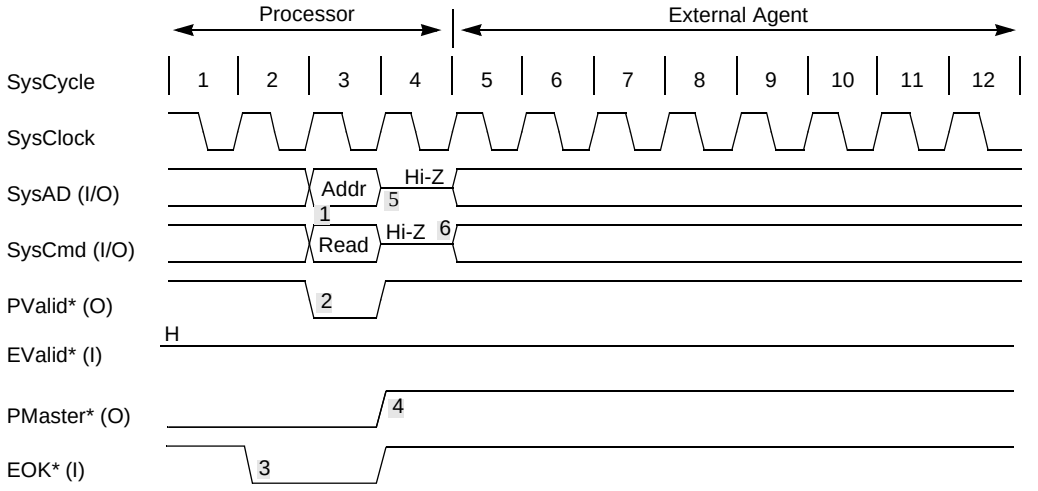


Figure 13-5 Uncompelled Transition Following Processor Read Request

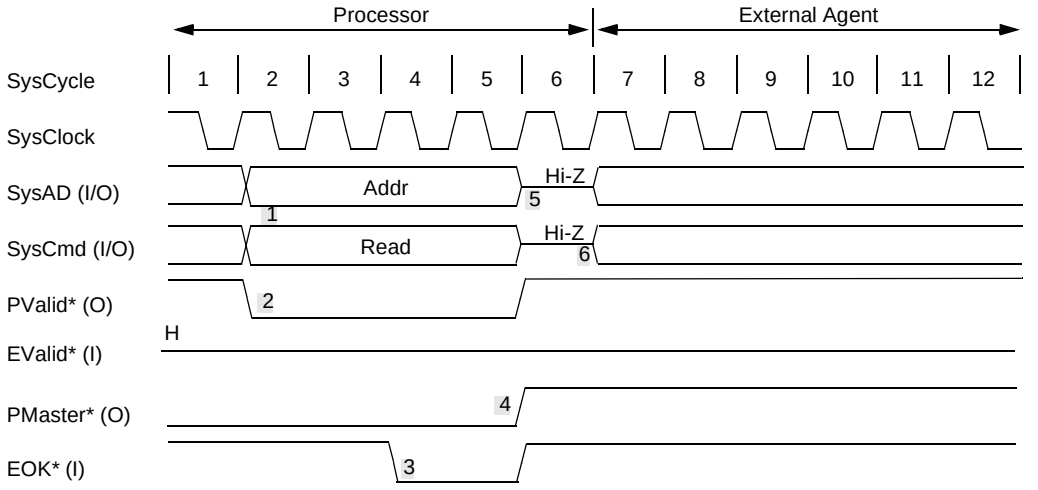


Figure 13-6 Delayed Processor Read Request

13.5.2 Processor Write Request Protocol

Processor write requests are issued by outputting a write command on the SysCmd (4:0) bus and a write address on the SysAD (31:0) bus, and asserting the PValid* signal. After that, a data identifier is output on SysCmd (4:0), write data is output on SysAD (31:0), and the PValid* signal is asserted active during the cycles necessary for transferring the data. The transfer rate at this time is set by the *EP* bits of the Config register.

The data cycle is based on the size of the write request.

- 1 to 4 bytes—Single data cycle
- 5 to 7 bytes—Divided into two single write requests (one is 4 byte long and the other is 1 to 3 bytes long)
- 8 bytes or more—Block data cycle in 4-byte units

The last data is appended with an NEOD (End of Data) data identifier.

Figure 13-7 shows a processor block write request with write data pattern W, and Figure 13-8 shows a processor block write request with write data pattern Wxx.

The following sequence describes the protocol of the processor write request (the numbers correspond to the numbers in Figure 13-7 and Figure 13-8).

1. The processor is in the master state. It outputs a write command on SysCmd (4:0) and a write address on SysAD (31:0) to issue a write request.
2. The processor asserts the PValid* signal to indicate that the current data on SysCmd (4:0) and SysAD (31:0) are valid.
3. The external agent asserts the EOK* signal for two consecutive cycles to enable issuance of a processor write request. If the EOK* signal is deasserted, the issuance cycle of the write request is delayed.
4. The processor outputs a data identifier on SysCmd (4:0) and write data on SysAD (31:0).
5. The processor asserts the PValid* signal for the cycles necessary for data transfer, and transfers the data.
6. The last data is appended with the EOD data identifier.

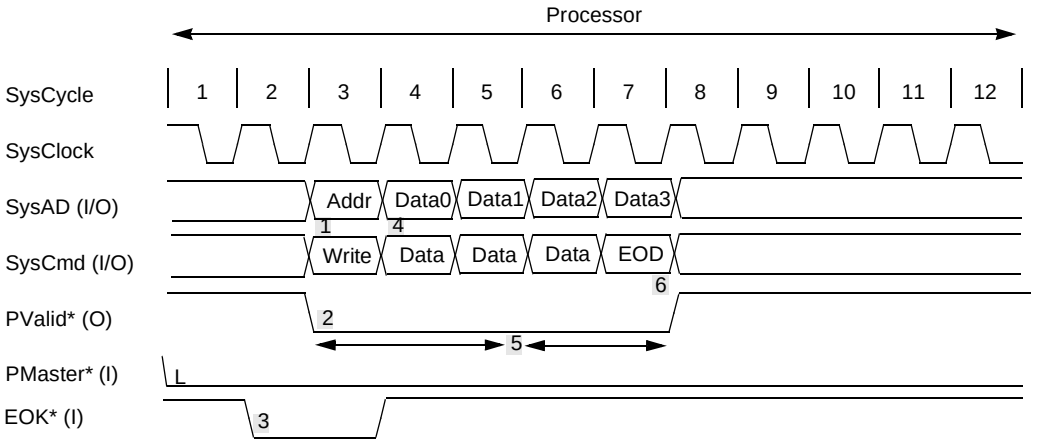


Figure 13-7 Processor Block Write Request (Write Data Pattern: W)

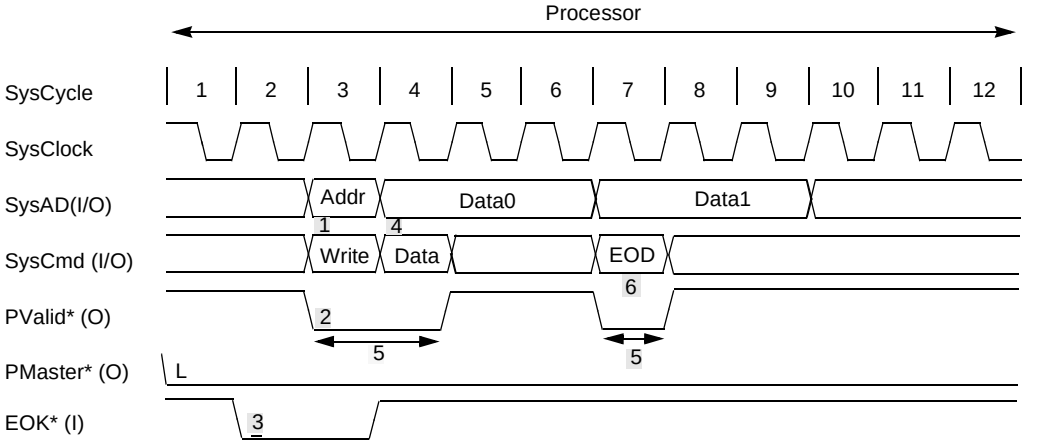


Figure 13-8 Processor Block Write Request (Write Data Pattern: Wxx)

13.5.3 Processor Request Flow Control

The external agent uses the EOK* signal to control the flow of processor read requests. The processor repeats the current address cycle until the EOK* signal is asserted active. This address cycle continues for 1 cycle after the EOK* signal is asserted, and then the issuance cycle ends. The EOK* signal must be asserted for at least two consecutive cycles.

Figure 13-9 and Figure 13-10 show how to use the EOK* signal (the numbers in the description below correspond to the numbers in Figure 13-9 and Figure 13-10).

1. Because the EOK* signal on the previous cycle is inactive, the processor request is delayed and the address cycle does not end.
2. Because the EOK* signal on the previous cycle is active, the processor request is not delayed and the address cycle ends.

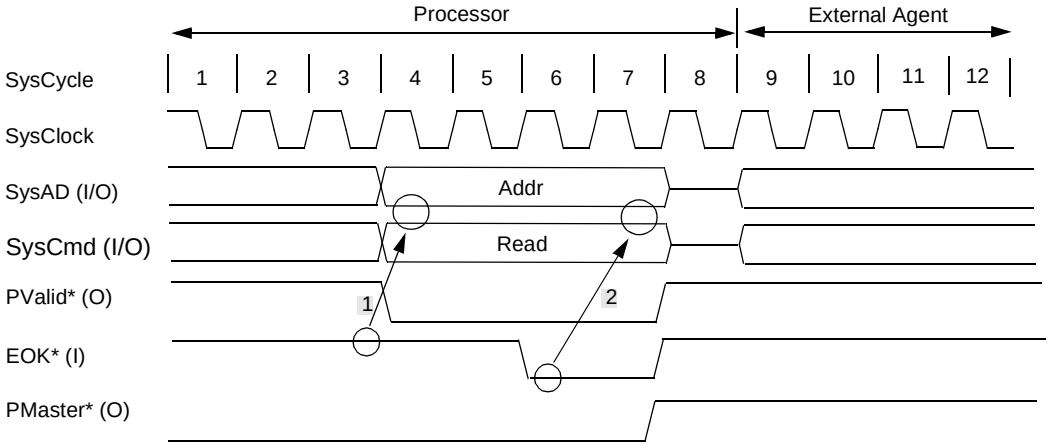


Figure 13-9 Delayed Processor Read Request

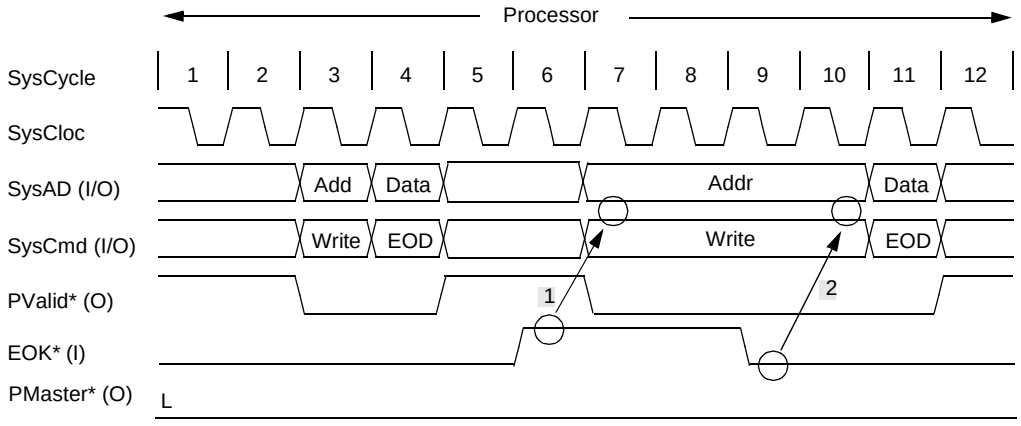


Figure 13-10 Delayed Second Processor Write Request

13.5.4 Successive Processing of Requests

13.5.4.1 Successive processor write requests

The processor write requests may be successively operated as follows.

- Using data pattern “W”—the processor write requests are processed without wait states, as shown in Figure 13-11
- Using data pattern “Wxx”—the write requests are separated by a wait state of two cycles, as shown in Figure 13-12.

The processor write requests may be successively issued in the following four cases:

- Successive single write request
- Successive block write request
- Block write request after single write reques
- Single write request after block write reques

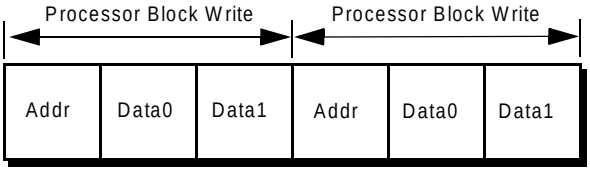


Figure 13-11 Successive Block Write Requests (Write Data Pattern: W)

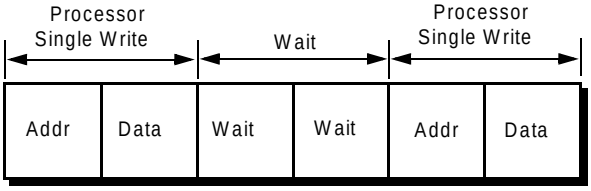


Figure 13-12 Successive Single Write Requests (Write Data Pattern: Wxx)

13.5.4.2 Processor write request followed by processor read request

Figure 13-13 shows a case where a processor read request follows a processor write request using write data pattern W.

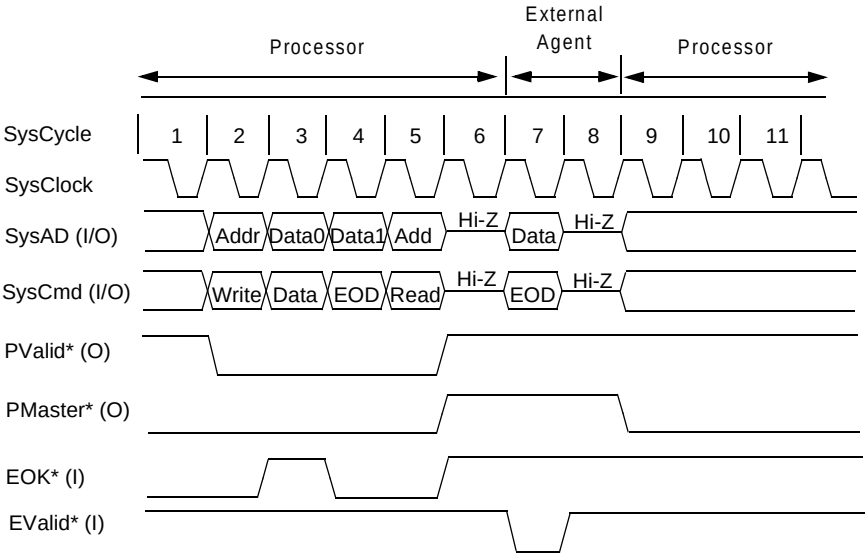


Figure 13-13 Processor Write Request Followed by Processor Read Request

13.5.4.3 Processor read request followed by processor write request

Figure 13-14 shows a case where a processor write request follows a processor read request using write data pattern W.

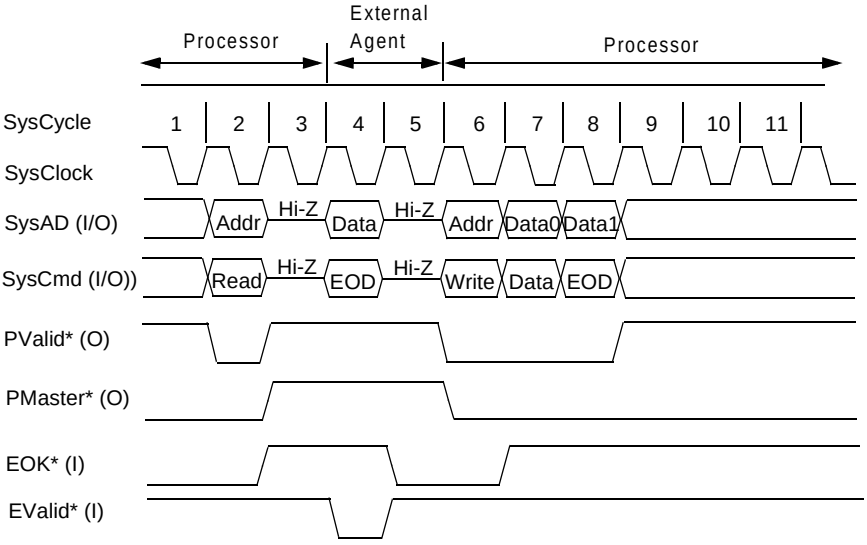


Figure 13-14 Processor Single Read Request Followed by Block Write Request

13.5.4.4 Processor write request followed by external write request

Figure 13-15 shows a case where processor write requests are followed by an external write request using write data pattern W.

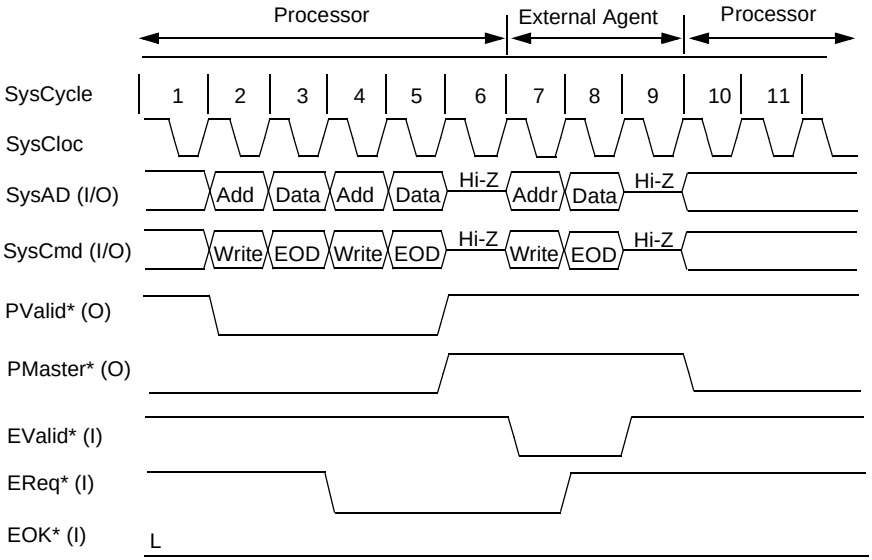


Figure 13-15 Processor Write Requests Followed by External Write Request

13.6 External Request Protocols

External requests can only be issued with the system interface in slave state. The EReq* signal is asserted by the external agent to arbitrate for the system interface, which then waits for the processor to release the system interface to slave state. If the system interface is already in slave state, the external agent can begin an external request immediately.

After issuing an external request, the external agent must return mastership of the system interface to the processor, as described below.

Following the description of the arbitration protocol, this section also describes the following external request protocols:

- Writ
- Read response

13.6.1 External Arbitration Protocol

Usually, the processor serves as the bus master. However, the processor relinquishes control of the bus and enters the slave state in the following cases:

- If the external agent issues a request and the system interf responds to that request
- After the processor has issued a read request

Arbitration to allow the processor to enter the slave state from the master state is performed using the handshake signals (EReq*, PReq*, and PMaster*) of the system interface.

13.6.1.1 State transition on read response

While the processor read request is kept pending, the processor enters the slave state by deasserting the PMaster* signal inactive, and the external agent returns read response data.

If the EReq* signal is deasserted inactive, the processor remains in the slave state until the read response data is returned, and then returns to the master state by asserting the PMaster* signal active.

The external agent can remain in the master state as long as the EReq* signal remains active when the read response is returned.

13.6.1.2 Acquiring bus control with EReq*

If the processor is in the master state when the external agent has issued an external request, the external agent gains control of the bus by asserting the EReq* signal active and waiting until the processor deasserts the PMaster* signal inactive. When the processor deasserts the PMaster* signal inactive, the external agent has bus control.

Once the external agent has entered the master state, it can remain in the master state as long as the EReq* signal is asserted active. When EReq* is deasserted, the processor acquires bus control two cycles later.

Figure 13-16 shows the arbitration protocol of the external request issued by the external agent.

The following sequence describes the arbitration protocol (the numbers in the sequence correspond to the numbers in Figure 13-16).

1. The external agent continues asserting the EReq* signal active to issue an external request.
2. When the processor is ready to process the external request, it deasserts the PMaster* signal inactive.
3. The processor puts *SysAD* (31:0) and *SysCmd* (4:0) in the high-impedance state.
4. The external agent should drive *SysAD* (31:0) and *SysCmd* (4:0) one cycle after the PMaster* signal has been deasserted inactive.
5. The external agent can deassert the EReq* signal inactive in the last cycle of the external request (2 cycles before the external agent enters the slave state), except when it executes another external request.
6. The external agent should put *SysAD* (31:0) and *SysCmd* (4:0) in the high-impedance state on completion of the external request.

If the external agent has entered the master state by issuing the processor read request, the external agent must always return read request data. If the external agent has entered the master state by using the EReq* signal, any command and data can be issued in accordance with the arbitration process. This means that the processor always satisfies any request from the external agent.

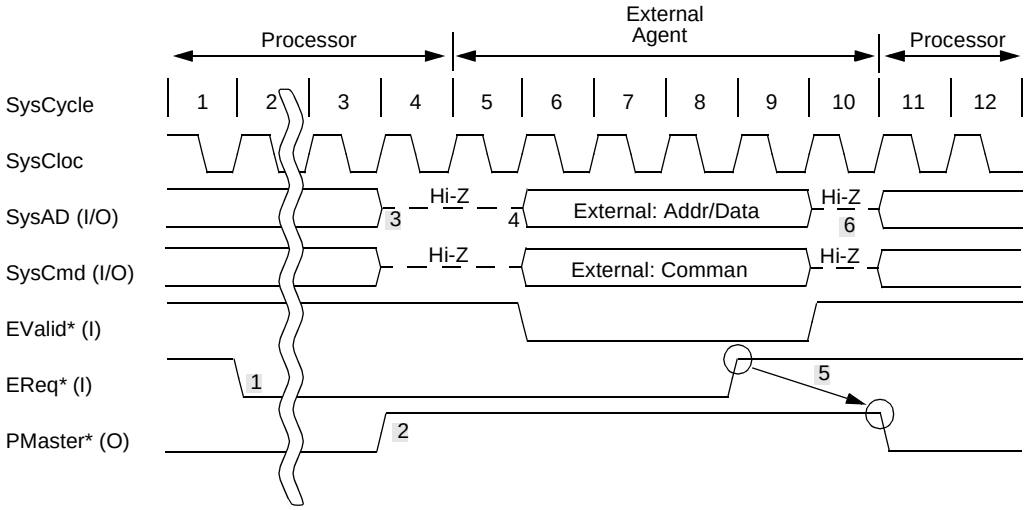


Figure 13-16 Arbitration of External Request

13.6.1.3 Restoring bus control with PReq*

Once the external agent has entered the master state, the processor cannot stop the operation of the external agent. However, the processor can request bus control by asserting the PReq* signal. At this time, the external agent must deassert the EReq* signal inactive in response to the request by the processor, giving consideration to the priority of the system.

The processor asserts the PMaster* signal two cycles after the EReq* signal has deasserted to inform the external agent that the processor has regained bus control.

Figure 13-17 illustrates how the processor requests bus control and how the external agent releases the bus in response.

At reset (when the Reset* or ColdReset* signal is asserted), the processor enters the master state, and the external agent enters the slave state.

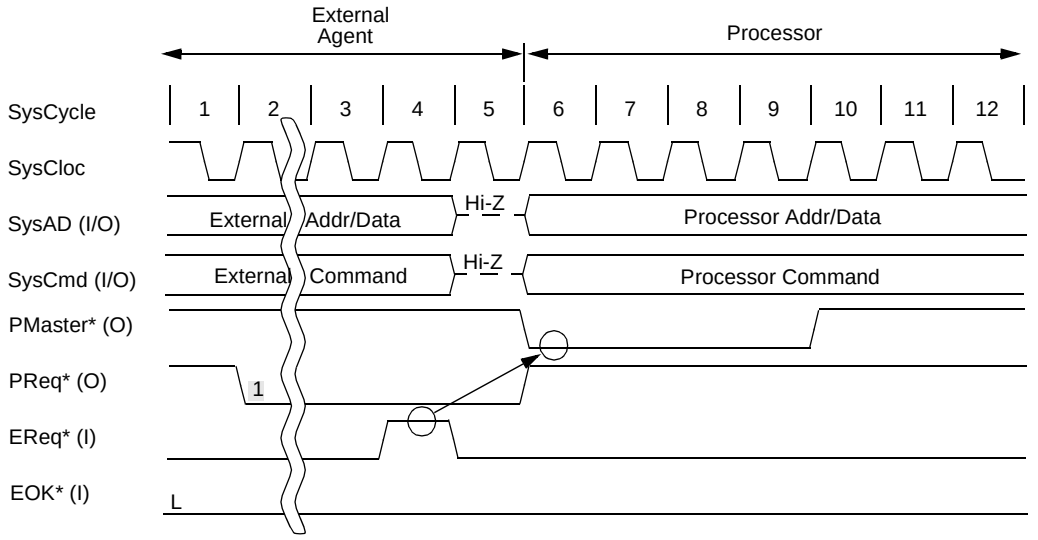


Figure 13-17 Bus Arbitration by the Processor

13.6.2 External Write Request Protocol

External write requests are similar in operation to a processor single write request, except that the EValid* signal is asserted in place of the PValid* signal.

When the processor is in slave state, the external agent can perform an external write request by outputting a write command on the SysCmd (4:0) bus, a write address on the SysAD (31:0) bus, and asserting the EValid* signal for one cycle. This is followed by outputting a data identifier on the SysCmd (4:0) bus, data on the SysAD (31:0) bus, and holding EValid* asserted for one more cycle. The data identifier of the data cycle must contain an end of data (NEOD) cycle indication. The EReq* signal is kept asserted while the external write request is issued.

After the data cycle is issued, the write request is completed. The external agent releases the SysCmd (4:0) and SysAD (31:0) buses and allows the system interface to enter master state.

An external write request with the processor generated in master state is illustrated in Figure 13-18.

Figure 13-21 shows an example in which the external agent issues an external write request following a read response. The external write request cannot be issued while read response data is transferred. It can be issued before the read response or after the last read data response. Only interrupt processing can be invoked by an external write request.

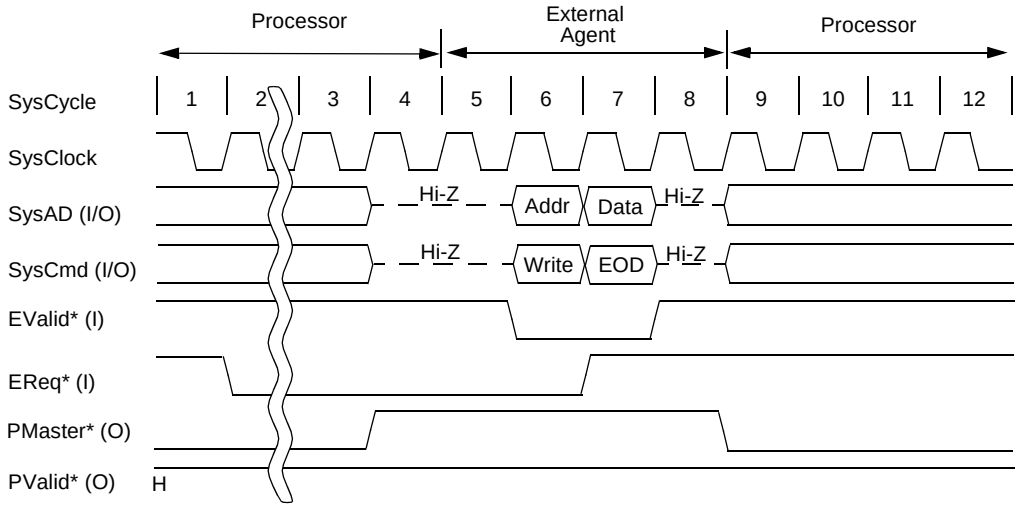


Figure 13-18 External Write Request Protocol

13.6.3 External Read Response Protocol

Before an external agent returns data to the processor, the processor sends a read request, then moves to slave state. The external agent then returns the data through a number of data cycles sufficient for the requested data size.

The SysCmd (4:0) and SysAD (31:0) buses are released after the last data cycle is issued. If the EReq* signal is inactive at this time, the processor returns to the master state two cycles after the last data cycle.

The data identifier associated with a data cycle may indicate that data transferred during this cycle is erroneous; however, an external agent must return a specific data block whether or not the data is erroneous. If a read response includes one or more erroneous data cycles, the processor generates a Bus Error exception.

Read response data can be transferred to the processor only when a processor read request is pending. If a read response is transferred to the processor while no processor read request is pending, the operation of the processor is undefined.

A processor read request followed by a read response is illustrated in Figure 13-19. A read response for a processor block read response with the processor already in slave state is illustrated in Figure 13-20.

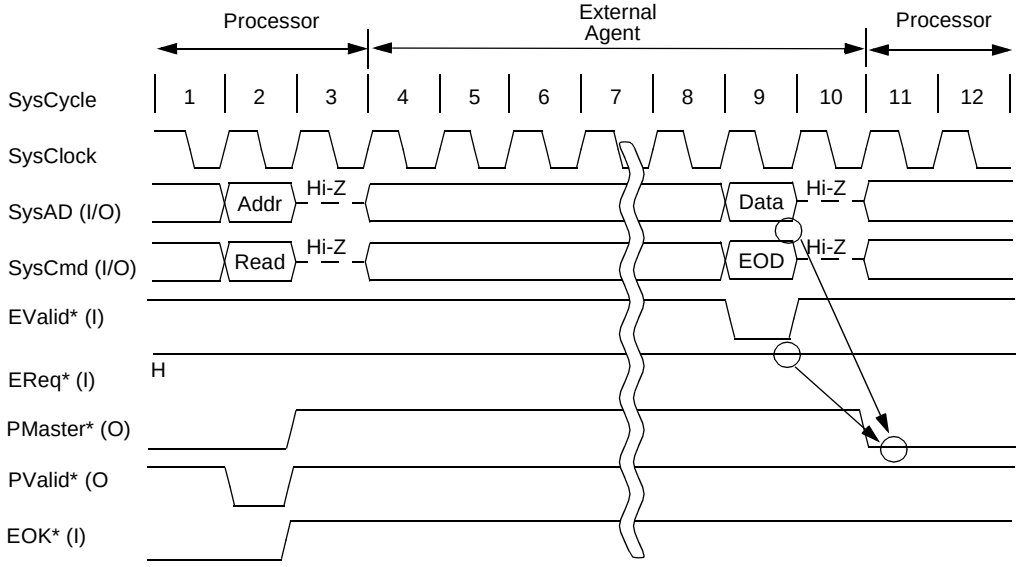


Figure 13-19 Read Request/Read Response Protocol

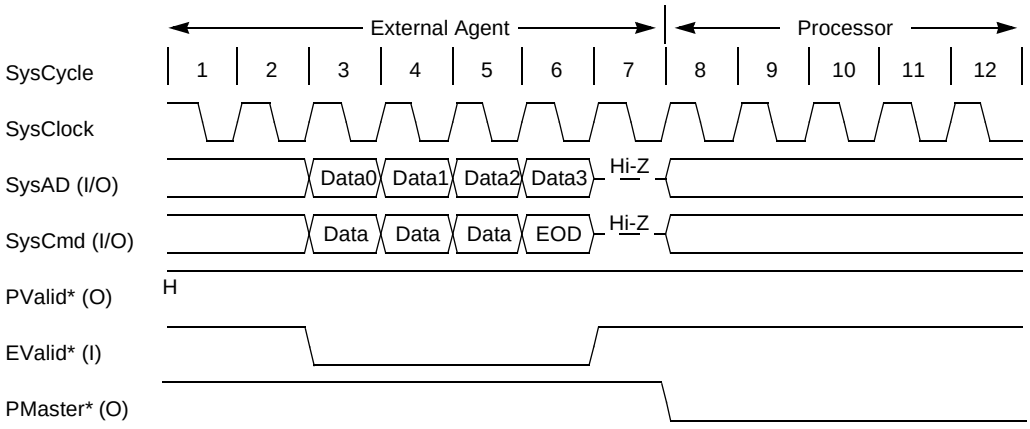


Figure 13-20 Block Read Response in Slave State

Figure 13-21 shows the case of an external write request being issued following a read response to a processor read request. The following sequence describes the protocol (the numbers in the following description correspond to the numbers in Figure 13-21).

1. The external agent returns response data to the processor read request.
2. To issue an external request following the read response, assert the EReq* signal active in the cycle in which an EOD is returned. In this case, the PMaster* signal remains inactive two cycles after the EOD.
3. Because the external agent is in the master state, it can issue the external write request.
4. Deassert the EReq* signal inactive up to the data cycle of the external write request. In this case, the PMaster* signal is asserted active two cycles after the EOD and bus control is returned to the processor.

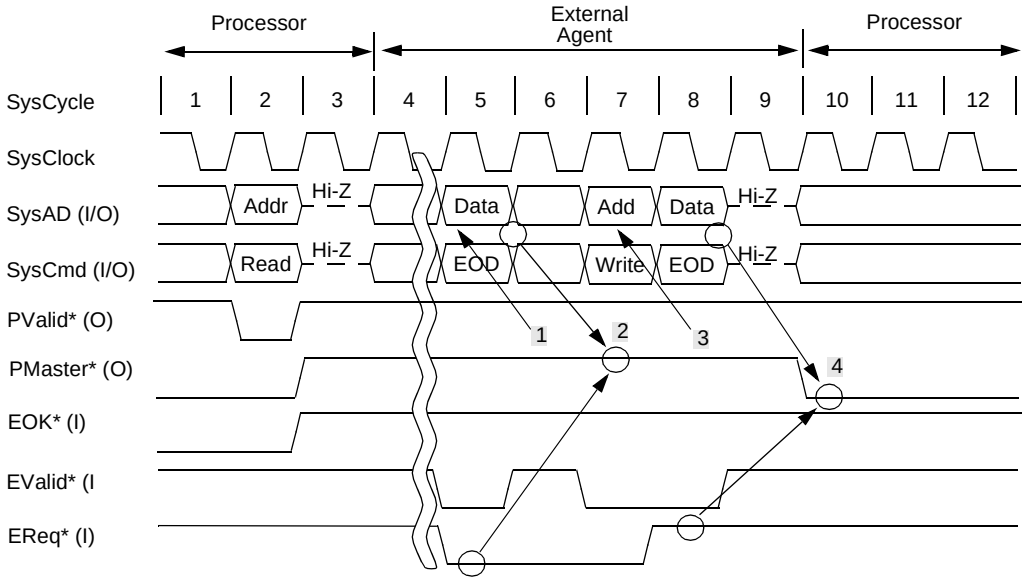


Figure 13-21 External Write Request Following Read Request

Figure 13-22 shows an example in which an external write request interrupts a read response to a processor read request. Cycle 5 in Figure 13-22 transfers the write data for the external write request in cycle 4, and cycle 7 transfers the read response data.

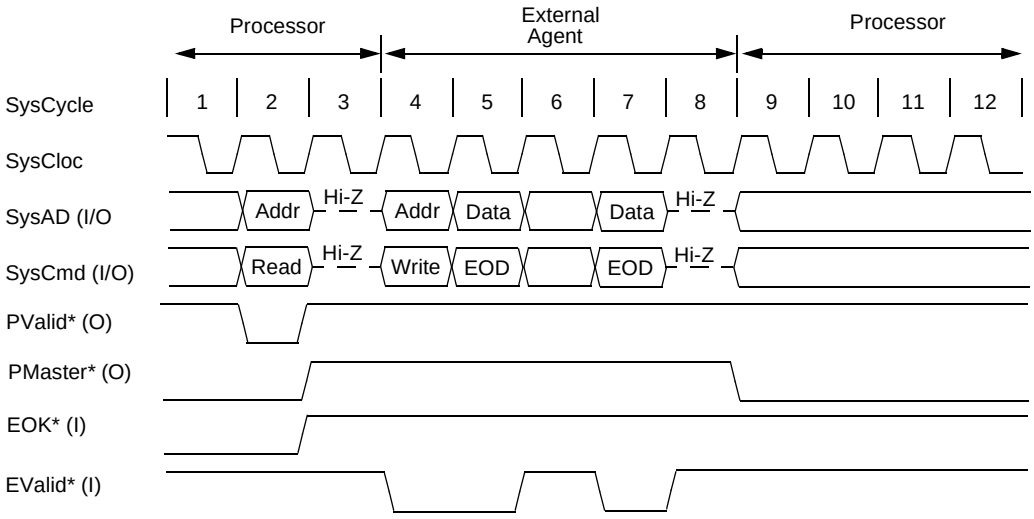


Figure 13-22 External Write Request Interrupts Processor Read Request

As shown in Figure 13-22, even if the external request interrupts the processor read request, the processor remains in the slave state until the read response data is returned.

13.7 Discarding and Re-Executing Commands

13.7.1 Re-Execution of Processor Commands

The external agent executes and controls the processor command with the EOK* signal. When the processor serves as the master, the processor cannot issue a command until the EOK* signal is active for at least two cycles.

If the EOK* signal is active for only one cycle before the processor issues a command and then becomes inactive in the next cycle in which the command is issued, the processor command is discarded. At this time, the external agent must ignore the discarded command.

13.7.1.1 Discarding a write command

When a write command is discarded, the processor issues the write data and then reissues the write command. The external agent must ignore the write data following the discarded write command.

13.7.1.2 Discarding a read command

When a read command is discarded, the processor enters the slave state in the cycle following the address cycle of the read request. If the EReq* signal is not active at this time, the processor returns to the master state again one cycle later, and reissues the read request.

13.7.2 Discarding and Re-Executing a Write Command

Figure 13-23 illustrates how a processor single write request is discarded and re-executed. The following sequence describes the protocol (the numbers in the following description correspond to the numbers in Figure 13-23).

1. Because the EOK* signal is active one cycle before (cycle 2) the write request of Data0, this cycle is the issuance cycle.
2. Because the EOK* signal is active in the write request cycle of Data0 (cycle 3), the next cycle is a normal data cycle.
3. Because the EOK* signal is active one cycle (cycle 4) before the write request of Data1, this cycle is the issuance cycle.
4. Because the EOK* signal is inactive in the write request cycle of Data1 (cycle 5), the data of the next cycle is discarded. At this time, data and command is output to *SysAD* (31:0) and *SysCmd* (4:0), which must be ignored by the external agent.
5. Because the EOK* signal is inactive one cycle (cycle 6) before the write request of the second Data1, the write request is delayed.
6. Because the EOK* signal is active one cycle (cycle 9) before the write request of the second Data1, this cycle is the issuance cycle.
7. Because the EOK* signal is active in the write request cycle (cycle 10) of the second Data1, the next cycle is a normal data cycle.

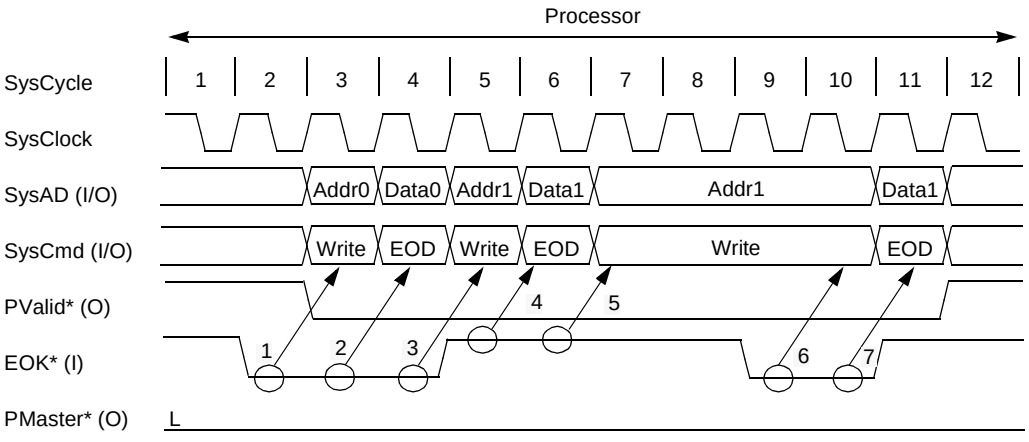


Figure 13-23 Discarding and Re-Executing a Processor Single Write Request

13.7.3 Discarding and Re-Executing a Read Command

Figure 13-24 illustrates how a processor single read request is discarded and re-executed. The following sequence describes the protocol (the numbers in the following description correspond to the numbers in Figure 13-24).

1. Because the EOK* signal is low in cycle 5, the processor tries to issue a command and address (cycle 6).
2. If the EOK* signal is high at this point, the processor discards this read request and enters the slave state in the next cycle.
3. Because the EReq* signal is inactive, the processor returns to the master state again and reissues the read request. Because the EOK* signal is low in both cycles 7 and 8, the issuance cycle of the read request is accepted.
4. The external agent outputs data from the requested address.

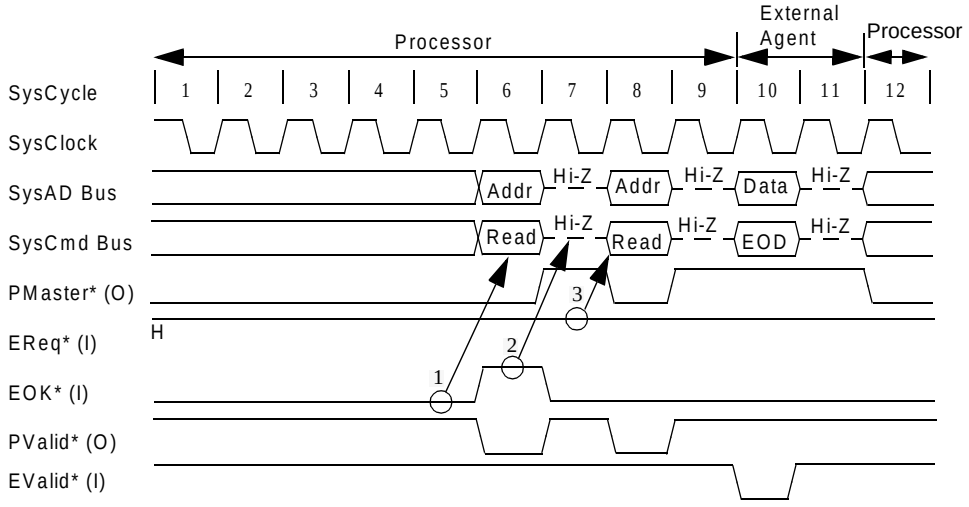


Figure 13-24 Discarding and Re-Executing a Processor Single Read Request

13.7.4 Executing and Discarding a Command

13.7.4.1 When the external agent requests bus control

The external agent requests the bus by asserting the EReq* signal active. At this time, the external agent can acquire bus control after it has accepted one processor read/write request only, or without accepting any request.

If the EReq* signal is asserted active while the external agent delays the processor request by deasserting the EOK* signal inactive, the external agent can forcibly acquire bus control.

13.7.4.2 When the processor requests bus control

The processor requests bus mastership by asserting the PReq* signal active. At this time, the external agent should transfer bus control to the processor, giving consideration to the priority of the system. If the external agent keeps the EReq* signal inactive for more than one cycle, control of the bus is released.

The processor acquires bus control by asserting the PMaster* signal active two cycles after the EReq* signal has become inactive. If the EOK* signal is active at this time, the processor can issue a request.

Figure 13-25 shows an example in which the external agent has entered the slave state (i.e., the EReq* signal is inactive) from the master state, and then acquires bus control again after accepting one processor request.

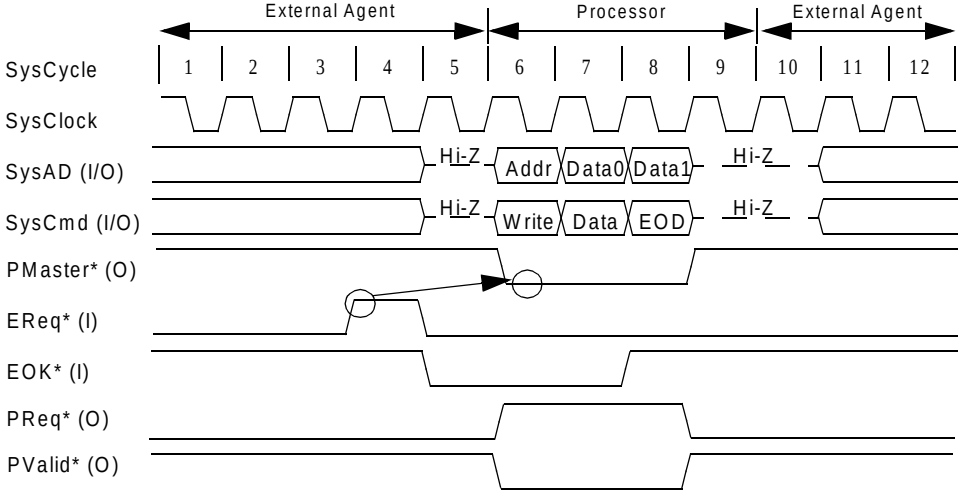


Figure 13-25 External Agent Releasing Bus Control in Response to Processor Request

13.8 SysADC (3:0) Protocol

In the R43K mode, SysADC (3:0) are unused.

13.9 Data Flow Control

The system interface supports a maximum data rate of one word per cycle.

13.9.1 Read Response

An external agent may transfer data to the processor at the maximum data rate of the system interface. The rate at which data is transferred to the processor can be controlled by the external agent, which asserts the EValid* signal on the cycle in which data is transferred. The processor accepts cycles as valid only when EValid* is asserted and the SysCmd (4:0) bus contains a data identifier. Thereafter, the processor continues to accept data until it receives the data word tagged as the last one.

Data identifier EOD indicates the last data word. Without receiving this indication, the system interface hangs up as a protocol error. In this case, the protocol error state indicated by PReq* toggling at half the MasterClock (i.e., SysClock) rate should invoke a reset initialization.

13.9.2 Write Request

The rate at which the processor transfers data to an external agent is programmable through the *EP* bits of the Config register (setting at reset is W) signal. Data patterns are defined using the letters W and x, where W indicates a word data cycle and x indicates an unused cycle. For example, a Wxx data pattern indicates a data rate of one word every three cycles. The VR5432 in R43K mode has two data transfer rates: W and Wxx. (A processor block write request with a Wxx data pattern (one word every three cycles) was shown in Figure 13-8.)

13.9.3 Independent Transfer on the SysAD (31:0) Bus

In general applications, the SysAD (31:0) bus is a point-to-point connection, running from the processor to a bidirectional register transceiver residing in an external agent. For these applications, the SysAD (31:0) bus has only two possible devices to connect, the processor and the external agent.

Certain applications may require connection of additional drivers and receivers to the SysAD (31:0) bus, to allow transfers over the SysAD (31:0) bus that the processor is not involved in. These are called *independent transfers*. To perform an independent transfer, the external agent must coordinate control of the SysAD (31:0) bus by using arbitration handshake signals (i.e., EReq*, PMaster*, and PReq*).

An independent transfer on the SysAD (31:0) bus follows this sequence:

1. The external agent asserts the EReq* signal and requests mastership of the SysAD (31:0) bus, to issue an external request.
2. The processor deasserts the PMaster* signal and releases the system interface to slave state.
3. The external agent then allows the independent transfer to take place on the SysAD (31:0) bus, making sure that the EValid* signal is not asserted during the transfer.
4. When the transfer is completed, the external agent deasserts EReq* to return the system interface to master state.

To connect multiple devices, separate enable signals for controlling each device are required to allow the nonprocessor chips to communicate.

13.9.4 System Endianness

The endianness of the system is determined by the BigEndian pin. Software can check system endianness by reading the *BE* bit of the Config register. If this bit is set to 1, the system is running in Big-Endian mode; otherwise, the system is running in Little-Endian mode.

13.10 System Interface Cycle Time

The processor has minimum and maximum cycle counts for the time required for various processor transactions and for the processor response time to external requests. Processor requests themselves are constrained by the system interface protocol, and request cycle counts can be determined by examining the protocol. System interface interactions that can vary within minimum and maximum cycle counts include the waiting period for the processor to release the system interface to slave state in response to an external request (release latency).

The remainder of this section describes and tabulates the minimum and maximum cycle counts for these system interface interactions.

13.10.1 Release Latency Time

Release latency time is defined as the number of cycles the processor can wait to release the system interface to slave state in response to an external request. When no processor requests are in progress, internal activity can cause the processor to wait some number of cycles before releasing the system interface. Release latency time is therefore the number of cycles from EReq* being asserted active until PMaster* is deasserted inactive.

There are two categories of release latency time:

1. When the EReq* signal is asserted one cycle before the last cycle of a processor request
2. When the EReq* signal is not asserted during a processor request, or is asserted during the last cycle of a processor request

Table 13-2 shows the minimum and maximum release latency time for requests that fall into categories 1 and 2.

Caution: The values in Table 13-2 are approximations and subject to change.

Table 13-2 Release Latency Time for External Requests

Category	Minimum PCycles	Maximum PCycles
1	4	6
2	4	24

13.11 System Interface Commands and Data Identifiers

System interface commands specify the types and attributes of any system interface request; this specification is made during the address cycle for the request. System interface data identifiers specify the attributes of data transferred during a system interface data cycle. The following sections describe the syntax of the protocol, the encoding of the bit patterns used for system interface commands, and data identifiers.

Reserved bits and reserved fields must be driven to 1 for system interface commands and data identifiers associated with external requests. For system interface commands and data identifiers associated with processor requests, reserved bits and reserved fields in the commands and data identifiers are undefined.

13.12 Command and Data Identifier Syntax

System interface commands and data identifiers are encoded in 5 bits and are transferred on the SysCmd (4:0) bus from the processor to an external agent, or from an external agent to the processor, during address and data cycles.

Bit 4 (the most-significant bit) of the SysCmd (4:0) bus determines whether the current content of the SysCmd bus is a command or a data identifier and therefore, whether the current cycle is an address cycle or a data cycle. For system interface commands, *SysCmd* (4) must be driven to 0. For system interface data identifiers, *SysCmd* (4) must be driven to 1.

Table 13-3 Encoding of SysCmd (4) for System Interface Commands

Bit	Meaning
SysCmd (4)	Attributes 0: Command (address) 1: Data identifier

13.12.1 System Interface Command Syntax

This section describes the *SysCmd* (4:0) bus encoding for system interface commands. Figure 13-26 shows the common encoding used for all system interface commands.

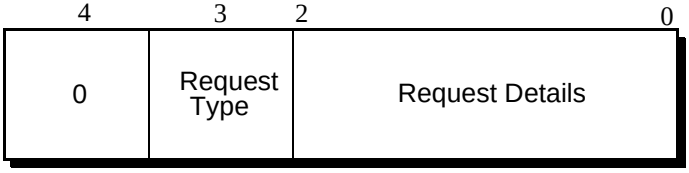


Figure 13-26 System Interface Command Bit Definition

SysCmd (4) must be driven to 0 for all system interface commands.

SysCmd (3) specifies the system interface request type, which may be read or write.

Table 13-4 Encoding of SysCmd (3) for System Interface Commands

Bit	Meaning
SysCmd (3)	Command 0: Read request 1: Write request

SysCmd (2:0) are specific to each type of request and are defined in the following sections.

13.12.2 Read Requests

Figure 13-27 shows the format of a SysCmd read request.

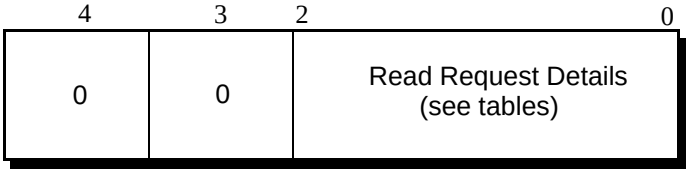


Figure 13-27 Read Request SysCmd (4:0) Bit Definition

Table 13-5 through Table 13-7 list the encoding of *SysCmd* (2:0) bits for read requests.

Table 13-5 Encoding of SysCmd (2) for Read Requests

Bit	Meaning
SysCmd (2)	Read attributes 0: Single read 1: Block read

Table 13-6 Encoding of SysCmd (1:0) for Block Read Requests

Bit	Meaning
SysCmd (1:0)	Read block size 0: 2 words 1: 4 words 2: 8 words 3: Reserved

Note: The read block size of 8 words is available only when the address starts at A[4:0] = 00000. Otherwise, the 8-word block is broken into 4-word blocks.

Table 13-7 Encoding of SysCmd (1:0) for Single Read Requests

Bit	Meaning
SysCmd (1:0)	Read data size 0: 1 byte valid (byte) 1: 2 bytes valid (halfword) 2: 3 bytes valid 3: 4 bytes valid (word)

Table 13-8 Processor Read Orders

Transfer Types	VR5432 in R43K Mode (Little-Endian System)	VR5432 in R43K Mode (Big-Endian System)
Block (A[4:0] = 00000) (D_n[31:0] is the low-order word and D_n[63:32] is the high-order word of the doubleword D_n[63:0])	1. D_0[31:0] 2. D_0[63:32] 3. D_1[31:0] 4. D_1[63:32] 5. D_2[31:0] 6. D_2[63:32] 7. D_3[31:0] 8. D_3[63:32]	1. D_0[63:32] 2. D_0[31:0] 3. D_1[63:32] 4. D_1[31:0] 5. D_2[63:32] 6. D_2[31:0] 7. D_3[63:32] 8. D_3[31:0]
Block (A[4:0] = 01000)	1. D_1[31:0] 2. D_1[63:32] 3. D_0[31:0] 4. D_0[63:32]	1. D_1[63:32] 2. D_1[31:0] 3. D_0[63:32] 4. D_0[31:0]
Block (A[4:0] = 11000)	5. D_3[31:0] 6. D_3[63:32] 7. D_2[31:0] 8. D_2[63:32]	5. D_3[63:32] 6. D_3[31:0] 7. D_2[63:32] 8. D_2[31:0]
Block (A[4:0] = 10000)	1. D_2[31:0] 2. D_2[63:32] 3. D_3[31:0] 4. D_3[63:32]	1. D_2[63:32] 2. D_2[31:0] 3. D_3[63:32] 4. D_3[31:0]
Block (A[4:0] = 00000)	5. D_0[31:0] 6. D_0[63:32] 7. D_1[31:0] 8. D_1[63:32]	5. D_0[63:32] 6. D_0[31:0] 7. D_1[63:32] 8. D_1[31:0]
Block (A[4:0] = 11000)	1. D_3[31:0] 2. D_3[63:32] 3. D_2[31:0] 4. D_2[63:32]	1. D_3[63:32] 2. D_3[31:0] 3. D_2[63:32] 4. D_2[31:0]
Block (A[4:0] = 01000)	5. D_1[31:0] 6. D_1[63:32] 7. D_0[31:0] 8. D_0[63:32]	5. D_1[63:32] 6. D_1[31:0] 7. D_0[63:32] 8. D_0[31:0]
Doubleword	1. D[31:0] 2. D[63:32]	1. D[63:32] 2. D[31:0]
Word (or Partial Word)	1. W[31:0]	1. W[31:0]

13.12.3 Write Requests

Figure 13-28 shows the format of a SysCmd write request.

Table 13-9 lists the write attributes encoded in *SysCmd* (2). Table 13-10 lists the block write replacement attribute encoded in *SysCmd* (1:0). Table 13-11 lists the single write request encoded in bits *SysCmd* (1:0).

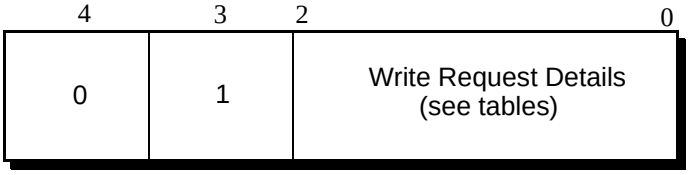


Figure 13-28 Write Request SysCmd (4:0) Bus Bit Definition

Table 13-9 Encoding of SysCmd (2) for Write Requests

Bit	Meaning
SysCmd (2)	Write attributes 0: Single write 1: Block write

Table 13-10 Encoding of SysCmd (1:0) for Block Write Requests

Bit	Meaning
SysCmd (1:0)	Write block size 0: 2 words 1: 4 words 2: Reserved 3: Reserved

Table 13-11 Encoding of SysCmd (1:0) for Single Write Requests

Bit	Meaning
SysCmd (1:0)	Write data size 0: 1 byte valid (byte) 1: 2 bytes valid (halfword) 2: 3 bytes valid 3: 4 bytes valid (word)

Table 13-12 Processor Write Orders

Transfer Types	VR5432 in R43K Mode (Little-Endian System)	VR5432 in R43K Mode (Big-Endian System)
Block (D_n[31:0] is the low-order word and D_n[63:32] is the high-order word of the doubleword D_n[63:0])	First 4-word block: 1. D_0[31:0] 2. D_0[63:32] 3. D_1[31:0] 4. D_1[63:32] Second 4-word block: 5. D_2[31:0] 6. D_2[63:32] 7. D_3[31:0] 8. D_3[63:32]	First 4-word block: 1. D_0[63:32] 2. D_0[31:0] 3. D_1[63:32] 4. D_1[31:0] Second 4-word block: 5. D_2[63:32] 6. D_2[31:0] 7. D_3[63:32] 8. D_3[31:0]
Doubleword	1. D[31:0] 2. D[63:32]	1. D[63:32] 2. D[31:0]
Word (or Partial Word)	1. W[31:0]	1. W[31:0]

The external agent can only write one word at a time to the VR5432; it takes one bus clock to transfer the word from the external agent to the VR5432.

13.12.4 System Interface Data Identifier Syntax

This section defines the encoding of the SysCmd (4:0) bus for system interface data identifiers. Figure 13-29 shows a common encoding used for all system interface data identifiers.

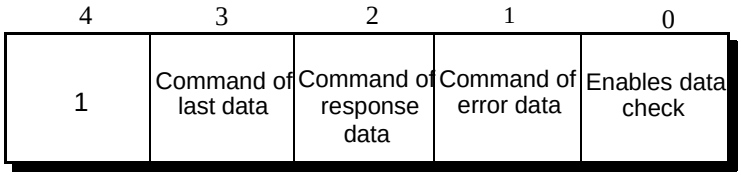


Figure 13-29 Data Identifier SysCmd (4:0) Bus Bit Definition

SysCmd (4) must be set to 1 for all system interface data identifiers.

13.12.5 Data Identifier Bit Definitions

This section describes *SysCmd* (3:0) bit definitions.

SysCmd (3) marks the last data element.

SysCmd (2) indicates whether or not the data is response data. Response data is data returned in response to a read request.

SysCmd (1) indicates whether or not the data element is error free. Erroneous data contains an uncorrectable error and is returned to the processor, resulting in a Bus Error exception. Because the VR5432 in VR4300 Compatibility mode does not have a parity check function, the processor does not transfer data by setting the error bit to 1.

SysCmd (0) enables a data check (reversed function). Because the VR5432 in R43K mode does not have a data check function, the processor outputs 1 (data check disabled) when it transfers data. When the external agent transfers data, the processor ignores this bit. However, set this bit to 1 to disable checking.

Table 13-13 lists the encoding of *SysCmd* (3:0) for processor data identifiers.

Table 13-14 lists the encoding of *SysCmd* (3:0) for external data identifiers.

Table 13-13 Processor Data Identifier Encoding of SysCmd (3:0)

Bit	Meaning
SysCmd (3)	Last-data-element indication 0: Last data element, or data element on single transfer 1: Not the last data element
SysCmd (2)	Reserved
SysCmd (1)	Reserved: error data indication The processor outputs 0 (error free)
SysCmd (0)	Reserved: data check enabled Processor outputs 1 (data check disabled)

Table 13-14 External Data Identifier Encoding of SysCmd (3:0)

Bit	Meaning
SysCmd (3)	Last-data-element indication 0: Last data element, or data element on single transfer 1: Not the last data element
SysCmd (2)	Response data indication 0: Data is response data 1: Data is not response data
SysCmd (1)	Error data indication 0: Data is error free 1: Data is erroneous
SysCmd (0)	Reserved: data check enabled Processor ignores this bit (external agent transfers 1)

13.13 System Interface Addresses

System interface addresses are full 32-bit physical addresses output on the SysAD (31:0) bus during address cycles.

13.13.1 Addressing Conventions

Addresses associated with word or partial word data transfers are aligned for the size of the data element. The system uses the following address conventions.

- Addresses associated with block requests are aligned to requeste doubleword boundaries; the low-order 3 bits of the address are 0
- Word requests set the low-order 2 bits of the address to 0.
- Halfword requests set the low-order bit of the address to 0
- Byte and tribyte requests use the byte address

13.13.2 Sublock Order Data Retrieval

Sublock order is the order in which data is returned in response to a processor block read request. The processor delivers the address of the requested doubleword within the block. An external agent must return the block of data in sublock order, starting with the addressed doubleword.

For block write requests, the processor always delivers the address of the doubleword at the beginning of the block; the processor delivers data beginning with the doubleword at the beginning of the block and progresses sequentially through the doublewords that form the block. See Appendix A for more details.

Initialization Interface

14

The VR5432 processor performs cold and warm resets, which use the ColdReset* and Reset* input signals.

- **Cold reset**. Restarts all clocks and resets the JTAG logic after th power supply is stable A cold reset completely reinitializes the internal state machines of the processor without saving any state information.
- **Warm reset**. Restarts the processor, but does not affect clocks or JTAG circuitry. A warm reset preserves the processor's internal state.

14.1 Processor Reset Signals

This section describes the ColdReset* and Reset* signals.

ColdReset*. The ColdReset* signal must be asserted (low) for either a power-on reset or a cold reset. ColdReset* must be deasserted synchronously with SysClock.

Reset*. The Reset* signal must be asserted for any reset sequence. It can be asserted synchronously or asynchronously for a cold reset, or synchronously to initiate a warm reset. Reset* must be deasserted synchronously with SysClock.

14.1.1 Power-On Reset

The sequence for a power-on reset is listed below.

1. Power-on reset requires a stable Vcc of at least +2.35 volts from the +2.5-volt power source to the processor and a VccIO of at least +3.0 volts from the +3.3-volt power source to the processor. It also requires a stable, continuous system clock at the processor operational frequency.
2. ColdReset* is asserted for at least 64 KB (2^{16}) SysClock cycles after the power is stable. ColdReset* must be deasserted synchronously with SysClock.
3. After ColdReset* is deasserted synchronously, Reset* is deasserted to allow the processor to begin running. (Reset* must be held asserted for at least 64 SysClock cycles after the deassertion of ColdReset*.) Reset* must be deasserted synchronously with SysClock.

The mode bits BigEndian, DivMode (1:0), and BypassPLL are latched one cycle after deassertion of ColdReset*. All of these signals must be continuously driven to the value desired by the user during ColdReset* assertion. Upon reset, the processor drives the SysAD bus. After Reset* is deasserted, the processor branches to the Reset exception vector and begins executing the Cold Reset exception code.

Figure 14-1 shows the power-on system reset timing diagram.

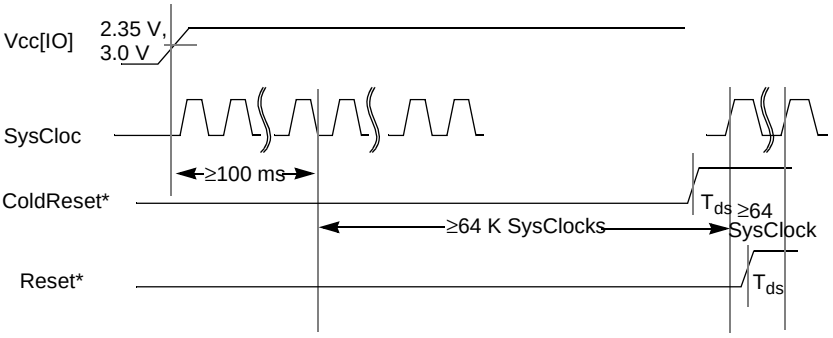


Figure 14-1 Power-On Reset Timing Diagram

14.1.2 Cold Reset

A cold reset requires the same sequence as a power-on reset except that the power is presumed to be stable before the assertion of the reset inputs.

Figure 14-2 shows the cold reset timing diagram.

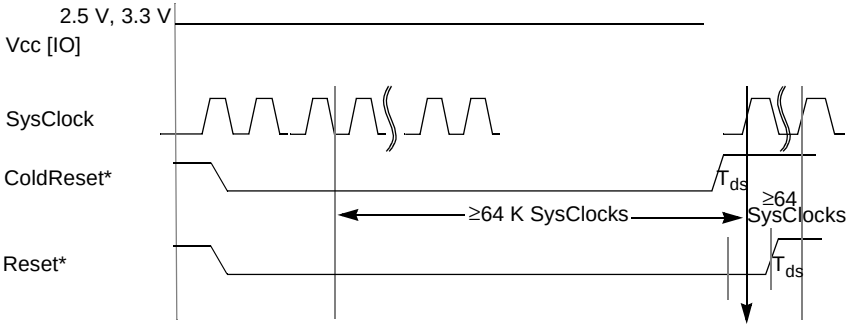


Figure 14-2 Cold Reset Timing Diagram

14.1.3 Warm Reset

To execute a warm reset, the Reset* input is asserted synchronously with SysClock. It is then held asserted for at least 64 SysClock cycles before being deasserted synchronously with SysClock. A warm reset forces the processor to start with a Soft Reset exception.

A warm reset is used to reset the processor without affecting the clocks; in other words, a warm reset is a logic reset. This allows the processor to retain as much of its state as possible for debugging. Because a warm reset takes effect immediately upon assertion of the Reset* signal, multicycle operations such as a cache miss or a Floating-Point instruction may be aborted, with some loss of data.

Upon reset, the processor drives the SysAD bus. After Reset* is deasserted, the processor branches to the Reset exception vector and begins executing the Reset exception code. If a reset is asserted in the middle of a SysAD transaction, care must be taken to reset all external agents to avoid SysAD bus contention.

Figure 14-3 shows the warm reset timing diagram.

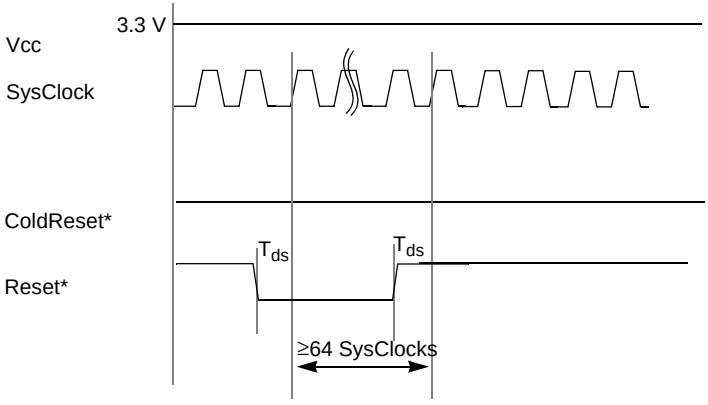


Figure 14-3 Warm Reset Timing Diagram

14.1.4 Processor Reset State

After a power-on reset, cold reset, or warm reset, all processor internal state machines are reset, and the processor begins execution at the reset vector.

Upon warm reset, the processor states that have been updated by the time reset was applied are preserved. The state that is committed but not updated is lost, because all state machines abort and return to the reset state. Therefore, the precise state of the caches depends on whether or not a cache miss sequence was interrupted by the reset. Also, since the processor implements nonblocking loads, the register update is killed if the Load instruction has not yet been completed. The branch prediction table is initialized as well.

14.2 Processor Initialization Signals

The VR5432 processor provides four input signal types that are sampled at processor initialization. These signals are used to set up the system clock-to-processor clock ratio, the memory organization, the clock source, and the system interface protocol.

- DivMode (1:0) sets up the clock ratio between the internal processo clock (PClock) and the external system clock (SysClock). Supported ratios are 4:1, 3:1, 2.5:1, and 2:1.
- The BigEndian input signal determines the byte ordering that the processor will use during operation. When asserted, the memor organization is big endian
- The BypassPLL* input signal selects whether the on-chip clock is supplied externally or generated by the on-chip PLL in synchronization with an external clock
- The OptionR43K* input signal selects whether the native R5432 system or VR4300-compatible protocol is used.

Clock Interface

15

15.1 Basic System Clocks

The clock signals used in the VR5432 processor are described in the following sections.

15.1.1 SysClock/MasterClock

The processor bases all internal and external clocking on the single SysClock input signal. When the VR4300-compatible system interface protocol is selected, this signal is referred to as MasterClock. However, the functionality does not change, and it will be called SysClock for the remainder of this chapter.

15.1.2 PClock

The processor generates an internal clock, PClock, at the initialization interface specified by the frequency multiplier of SysClock and phase-aligned to SysClock. For half-frequency multipliers (i.e., 2.5:1), the PClock is aligned on the opposite phase of SysClock.

15.2 Alignment to SysClock

- Processor output data changes a minimum of T_{dm} ns and becomes stable a maximum of T_{do} ns after the rising edge of SysClock. This drive time is the sum of the maximum delay through the processo output drivers and the maximum clock-to-Q delay of the processor output registers.
- Process r input data must be stable for a maximum of T_{ds} ns before the rising edge of SysClock and must remain stable a minimum of T_{dh} ns after the rising edge of SysClock.

15.3 Phase-Locked Loop (PLL)

The processor aligns PClock and SysClock with internal phase-locked loop (PLL) circuits that generate aligned clocks. By their nature, PLL circuits are only capable of generating aligned clocks for SysClock frequencies within a limited range.

Clocks generated using PLL circuits contain some inherent inaccuracy, or jitter; a clock aligned with SysClock by the PLL can lead or trail SysClock by as much as the related maximum jitter.

Figure 15-1 shows the SysClock timing parameters.

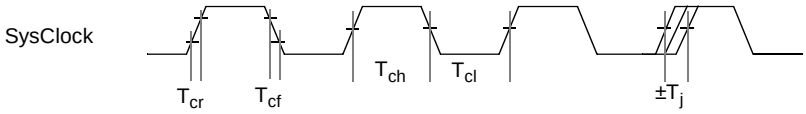


Figure 15-1 SysClock Timing

Figure 15-2 shows the input timing parameters.

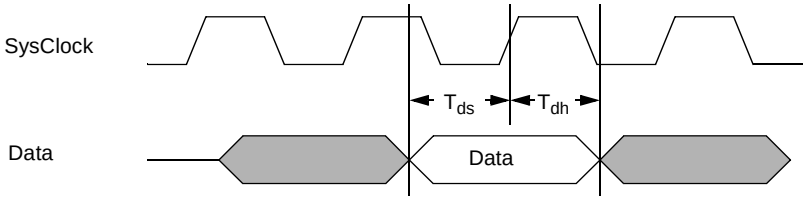


Figure 15-2 Input Timing

Figure 15-3 shows the output timing parameters measured at the midpoint of the rising clock edge.

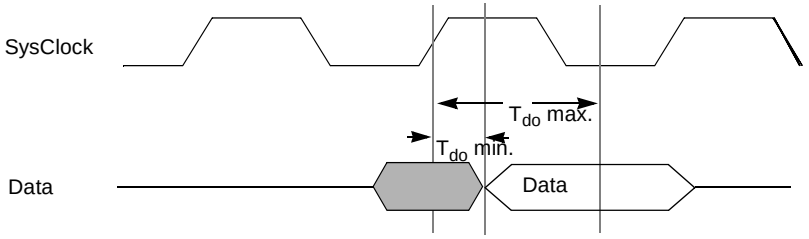


Figure 15-3 Output Timing

The SysClock input must meet the maximum rise time (T_{cr}), maximum fall time (T_{cf}), minimum T_{ch} time, minimum T_{cl} time, and T_j input jitter parameters for proper operation of the PLL.

15.4 Bypass PLL Mode

When the BypassPLL* signal is asserted, the VR5432 processor can be put into Bypass PLL mode. This mode requires BypassPLL* to be asserted statically from the cold reset sequence. When this occurs, the clock signal input to the chip using the SysClock signal will be used internally as the core clock (PClock).

The core clock will operate in a 2:1 clock ratio with the system clock. The alignment of the two clock signals is determined at the deassertion of the ColdReset* signal. Internal circuitry detects the deassertion of ColdReset* and causes the clock alignment to be consistent at that point. The setup for ColdReset* is relative to the SysClock signal. The rising edge of the SysClock signal to which ColdReset* sets up becomes a falling edge of the system clock internally. Therefore, the following rising edge of SysClock will correspond to the rising edge of the system clock, so that all signals coming into and leaving the chip will be synchronous.

Figure 15-4 shows the two cases that can occur during the cold reset sequence from the perspective of the internally generated system clock. As shown, the SysClock signal becomes the core clock source, which runs twice as fast as the system clock used to synchronize all external signals. When ColdReset* is deasserted, the second rising edge of the SysClock signal corresponds to the alignment of the clocks. In addition to the clock adjustment shown in Figure 15-4, it also shows that data must be correctly set up to the edge corresponding to the rising edge of the internal system clock.

The user also should be aware that there is a delay for the clocks internally due to the internal clock distribution. Without an internal PLL operating, the delays can be significant. Allowances should be made in the system or test environment for this delay.

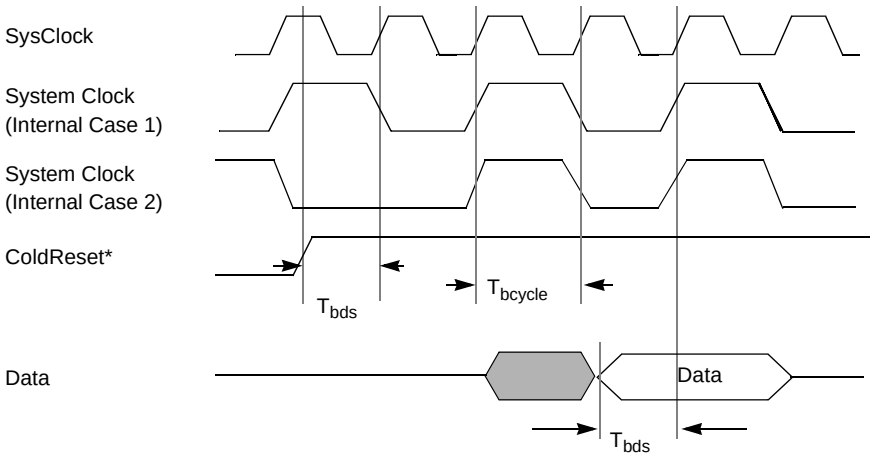


Figure 15-4 Bypass PLL Reset Timing

Index

C

D

E

F

J

K

L

M

N

O

P

R

S

T

U

V

W

X

NEC

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

- 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, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara
Tel: 800-366-9782
Fax: 800-729-9288

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

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

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

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 253-8311
Fax: 250-3583

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

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

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

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

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

NEC do Brasil S.A.
Sao Paulo-SP,Brasil
Tel: 011-889-1680
Fax: 011-889-1689

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

NEC

U13751EU5V0UMJ1