Memory Management

4

The MIPS R4000 processor provides a full-featured memory management unit (MMU) which uses an on-chip translation lookaside buffer (TLB) to translate virtual addresses into physical addresses.

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

4.1 Translation Lookaside Buffer (TLB)

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

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

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 (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.

Multiple Matches

If more than one entry in the TLB matches the virtual address being translated, the operation is undefined. To prevent permanent damage to the part, the TLB may be disabled if more than several entries match. The TLB-Shutdown *(TS)* bit in the *Status* register is set to 1 if the TLB is disabled.

† There are virtual-to-physical address translations that occur outside of the TLB. For example, addresses in *the kseg0* and *kseg1* spaces are unmapped translations. In these spaces the physical address is derived by subtracting the base address of the space from the virtual address.

4.2 Address Spaces

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

Virtual Address Space

The processor virtual address can be either 32 or 64 bits wide,† depending on whether the processor is operating in 32-bit or 64-bit mode.

- In 32-bit mode, addresses are 32 bits wide. The maximum user process size is 2 gigabytes (2^{31}).
- In 64-bit mode, 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.

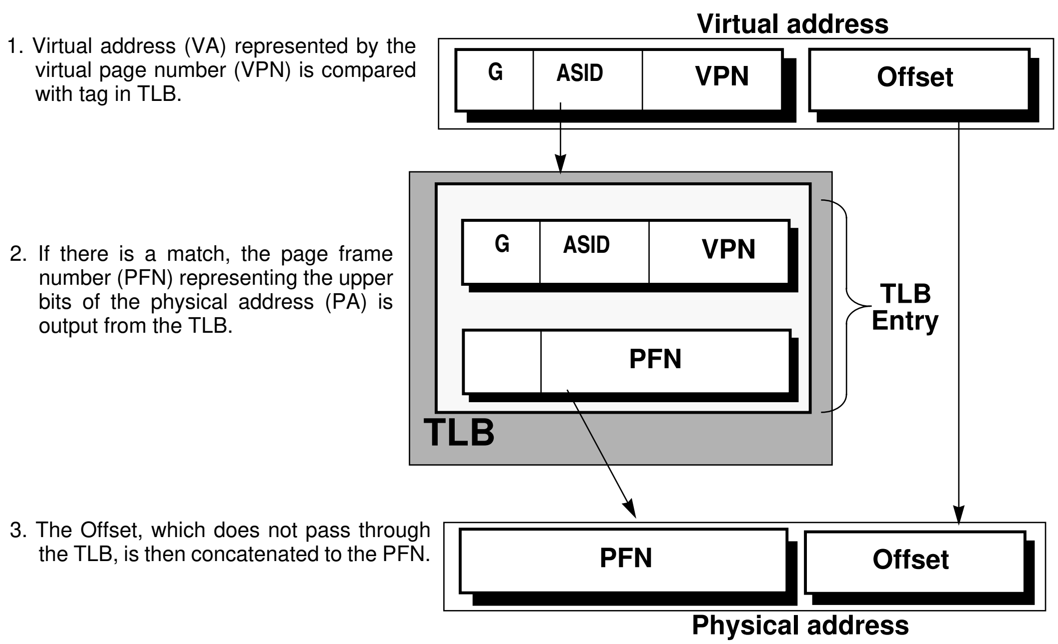


Figure 4-1 Overview of a Virtual-to-Physical Address Translation

† Figure 4-8 shows the 32-bit and 64-bit versions of the processor TLB entry.

As shown in Figures 4-2 and 4-3, the virtual address is extended with an 8-bit address space identifier (ASID), which reduces the frequency of TLB flushing when switching contexts. This 8-bit ASID is in the CP0 *EntryHi* register, described later in this chapter. The *Global* bit (*G*) is in the *EntryLo0* and *EntryLo1* registers, described later in this chapter.

Physical Address Space

Using a 36-bit address, the processor physical address space encompasses 64 gigabytes. The section following describes the translation of a virtual address to a physical address.

Virtual-to-Physical Address Translation

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

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

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

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

Virtual-to-physical translation is described in greater detail throughout the remainder of this chapter; Figure 4-20 is a flow diagram of the process shown at the end of this chapter.

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

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

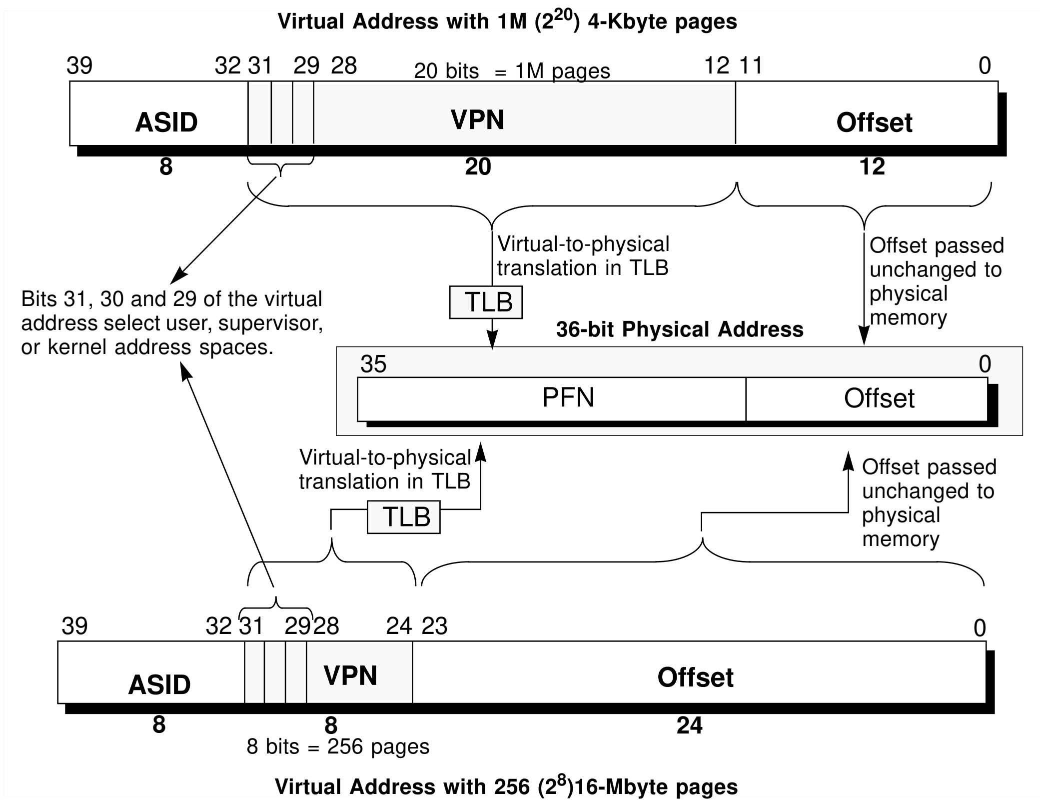


Figure 4-2 32-bit Mode Virtual Address Translation

64-bit Mode Address Translation

Figure 4-3 shows the virtual-to-physical-address translation of a 64-bit mode address. This figure illustrates the two extremes in the range of possible page sizes: a 4-Kbyte page (12 bits) and a 16-Mbyte page (24 bits).

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

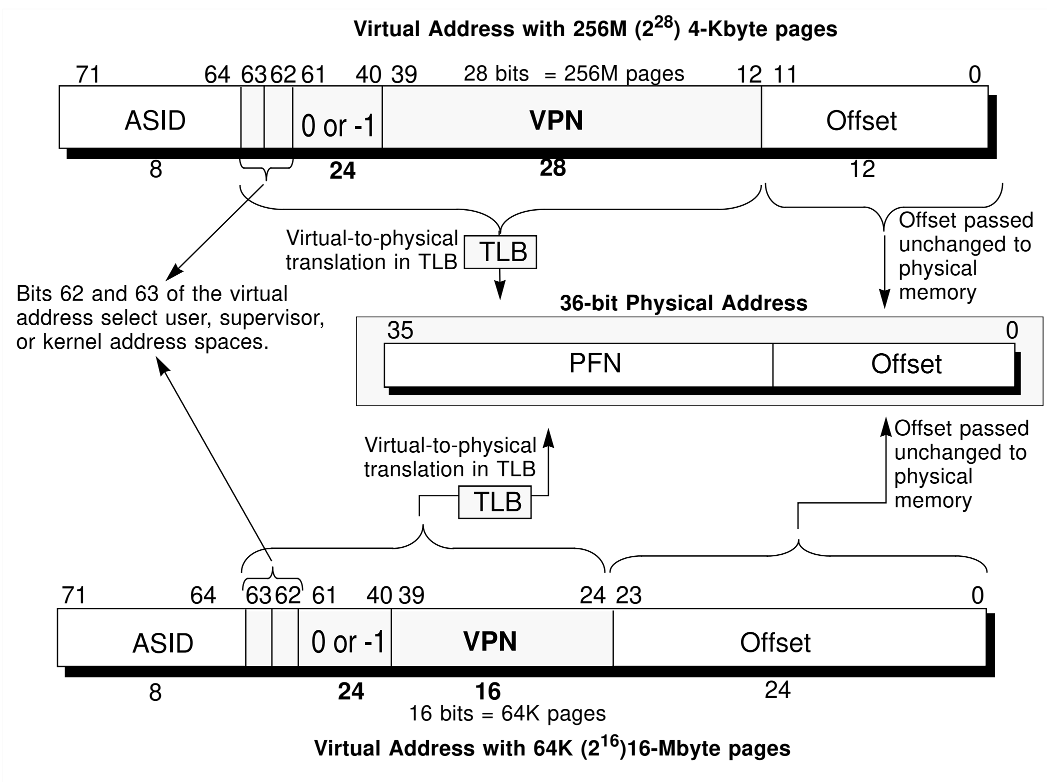


Figure 4-3 64-bit Mode Virtual Address Translation

Operating Modes

The processor has three operating modes that function in both 32- and 64-bit operations:

- User mode
- Supervisor mode
- Kernel mode

These modes are described in the next three sections.

User Mode Operations

In User mode, a single, uniform virtual address space—labelled User segment—is available; its size is:

- 2 Gbytes (2^{31} bytes) in 32-bit mode (*useg*)
- 1 Tbyte (2^{40} bytes) in 64-bit mode (*xuseg*)

Figure 4-4 shows User mode virtual address space.

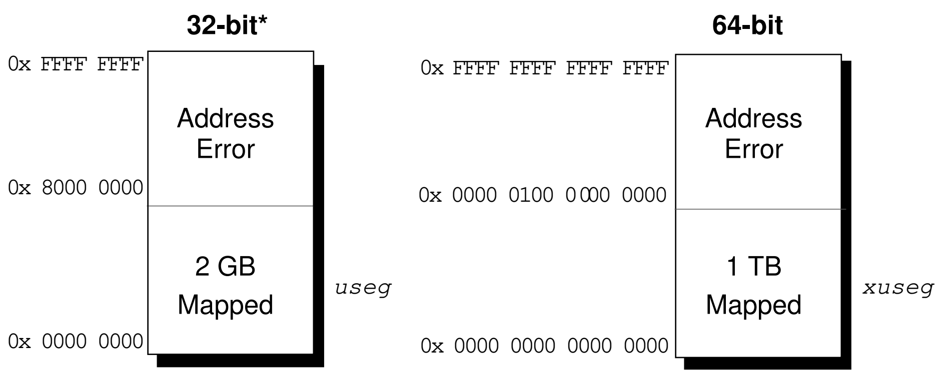


Figure 4-4 User Mode Virtual Address Space

***NOTE:** The R4000 uses 64-bit addresses internally. When the kernel is running in Kernel mode, it initializes registers before switching modes, and saves (or restores, whichever is appropriate) register values on context switches. In 32-bit mode, a valid address must be a 32-bit signed number, where bits 63:32 = bit 31. In normal operation it is not possible for a 32-bit User-mode program to produce invalid addresses. However, although it would be an error, it *is* possible for a Kernel-mode program to erroneously place a value that is not a 32-bit signed number into a 64-bit register, in which case the User-mode program generates an invalid address.

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

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

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

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

- when *UX* = 0, 32-bit *useg* space is selected and TLB misses are handled by the 32-bit TLB refill exception handler
- when *UX* = 1, 64-bit *xuseg* space is selected and TLB misses are handled by the 64-bit XTLB refill exception handler

Table 4-1 lists the characteristics of the two user mode segments, *useg* and *xuseg*.

Table 4-1 32-bit and 64-bit User Mode Segments

Address Bit Values	Status Register Bit Values				Segment Name	Address Range	Segment Size
	KSU	EXL	ERL	UX			
32-bit A(31) = 0	10_2	0	0	0	*useg*	0x0000 0000 through 0x7FFF FFFF	2 Gbyte (2^{31} bytes)
64-bit A(63:40) = 0	10_2	0	0	1	*xuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 Tbyte (2^{40} bytes)

† The cached (*C*) field in a TLB entry determines whether the reference is cached; see Figure 4-8.

32-bit User Mode (*useg*)

In User mode, when *UX* = 0 in the *Status* register, User mode addressing is compatible with the 32-bit addressing model shown in Figure 4-4, and a 2-Gbyte user address space is available, labelled *useg*.

All valid User mode virtual 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.

64-bit User Mode (*xuseg*)

In User mode, when *UX* =1 in the *Status* register, User mode addressing is extended to the 64-bit model shown in Figure 4-4. In 64-bit User mode, the processor provides a single, uniform address space of 2^{40} bytes, labelled *xuseg*.

All valid User mode virtual addresses have bits 63:40 equal to 0; an attempt to reference an address with bits 63:40 not equal to 0 causes an Address Error exception.

Supervisor Mode Operations

Supervisor mode is designed for layered operating systems in which a true kernel runs in R4000 Kernel mode, and the rest of the operating system runs in Supervisor mode.

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

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

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

- when *SX* = 0, 32-bit supervisor space is selected and TLB misses are handled by the 32-bit TLB refill exception handler
- when *SX* = 1, 64-bit supervisor space is selected and TLB misses are handled by the 64-bit XTLB refill exception handler

Figure 4-5 shows Supervisor mode address mapping. Table 4-2 lists the characteristics of the supervisor mode segments; descriptions of the address spaces follow.

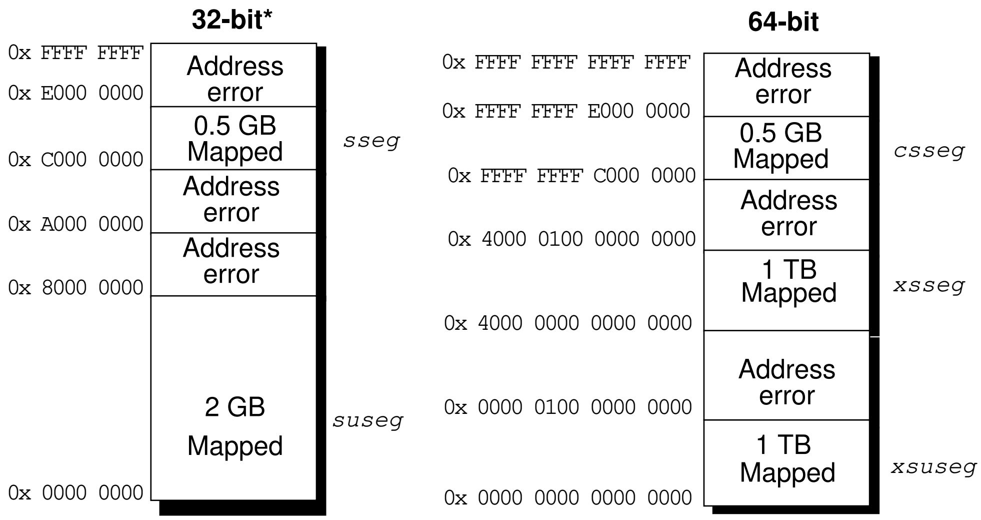


Figure 4-5 Supervisor Mode Address Space

***NOTE:** The R4000 uses 64-bit addresses internally. In 32-bit mode, a valid address must be a 32-bit signed number, where bits 63:32 = bit 31. In normal operation it is not possible for a 32-bit Supervisor-mode program to create an invalid address through arithmetic operations. However 32-bit-mode Supervisor programs must not create addresses using *base register+offset* calculations that produce a 32-bit 2's-complement overflow; in specific, there are two prohibited cases:

- offset with bit 15 = 0 and base register with bit 31 = 0, but (*base register+offset*) bit 31 = 1
- offset with bit 15 = 1 and base register with bit 31 = 1, but (*base register+offset*) bit 31 = 0

Using this invalid address produces an undefined result.

Table 4-2 32-bit and 64-bit Supervisor Mode Segments

Address Bit Values	Status Register Bit Values				Segment Name	Address Range	Segment Size
	KSU	EXL	ERL	SX			
32-bit A(31) = 0	01_2	0	0	0	*suseg*	0x0000 0000 through 0x7FFF FFFF	2 Gbytes (2^{31} bytes)
32-bit A(31:29) = 110_2	01_2	0	0	0	*ssseg*	0xC000 0000 through 0xDFFF FFFF	512 Mbytes (2^{29} bytes)
64-bit A(63:62) = 00_2	01_2	0	0	1	*xsuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 Tbyte (2^{40} bytes)
64-bit A(63:62) = 01_2	01_2	0	0	1	*xsseg*	0x4000 0000 0000 0000 through 0x4000 00FF FFFF FFFF	1 Tbyte (2^{40} bytes)
64-bit A(63:62) = 11_2	01_2	0	0	1	*csseg*	0xFFFF FFFF C000 0000 through 0xFFFF FFFF DFFF FFFF	512 Mbytes (2^{29} bytes)

32-bit Supervisor Mode, 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 Gbytes) 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.

32-bit Supervisor Mode, 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 Mbytes) 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.

64-bit Supervisor Mode, 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 Tbyte) 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.

64-bit Supervisor Mode, 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.

64-bit Supervisor Mode, 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 of the *csseg* is compatible with addressing *sseg* in 32-bit mode. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

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

Kernel Mode Operations

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

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

In conjunction with these bits, the *KX* bit in the *Status* register selects between 32- or 64-bit Kernel mode addressing:

- when *KX* = 0, 32-bit kernel space is selected and all TLB misses are handled by the 32-bit TLB refill exception handler
- when *KX* = 1, 64-bit kernel space is selected and all TLB misses are handled by the 64-bit XTLB refill exception handler

The processor enters Kernel mode whenever an exception is detected and it remains in Kernel mode until an Exception Return (ERET) instruction is executed. The ERET instruction restores the processor to the mode existing prior to the exception.

Kernel mode virtual address space is divided into regions differentiated by the high-order bits of the virtual address, as shown in Figure 4-6. Table 4-3 lists the characteristics of the 32-bit kernel mode segments, and Table 4-4 lists the characteristics of the 64-bit kernel mode segments.

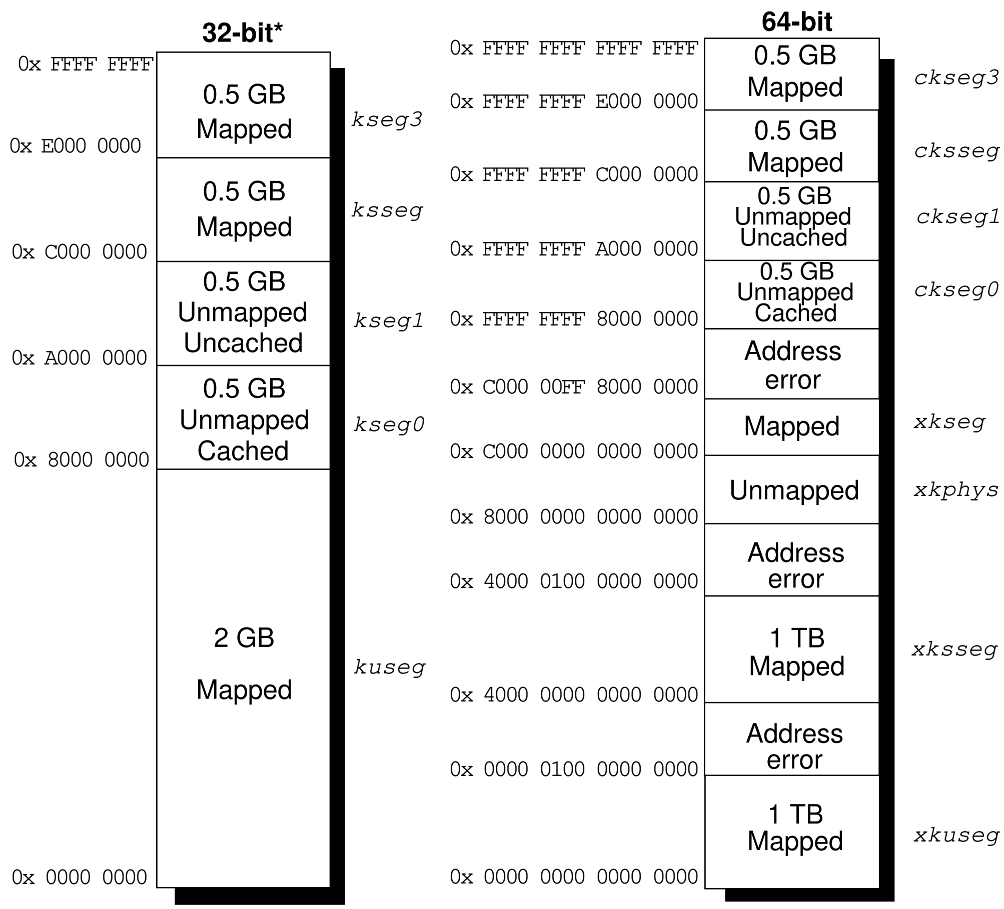


Figure 4-6 Kernel Mode Address Space

***NOTE:** The R4000 uses 64-bit addresses internally. In 32-bit mode, a valid address must be a 32-bit signed number, where bits 63:32 = bit 31; an invalid address produces an undefined result. In 32-bit mode, a Kernel-mode program may use 64-bit instructions, but must not create addresses using *base register+offset* calculations that produce a 32-bit 2's-complement overflow; in specific, there are two prohibited cases:

- offset with bit 15 = 0 and base register with bit 31 = 0, but (*base register+offset*) bit 31 = 1
- offset with bit 15 = 1 and base register with bit 31 = 1, but (*base register+offset*) bit 31 = 0

Table 4-3 32-bit Kernel Mode Segments

Address Bit Values	Status Register Is One Of These Values: KSU	EXL	ERL	KX	Segment Name	Address Range	Segment Size
A(31) = 0	KSU = 00_2 *or* EXL = 1 *or* ERL =1			0	*kuseg*	0x0000 0000 through 0x7FFF FFFF	2 Gbytes (2^{31} bytes)
A(31:29) = 100_2				0	*kseg0*	0x8000 0000 through 0x9FFF FFFF	512 Mbytes (2^{29} bytes)
A(31:29) = 101_2				0	*kseg1*	0xA000 0000 through 0xBFFF FFFF	512 Mbytes (2^{29} bytes)
A(31:29) = 110_2				0	*ksseg*	0xC000 0000 through 0xDFFF FFFF	512 Mbytes (2^{29} bytes)
A(31:29) = 111_2				0	*kseg3*	0xE000 0000 through 0xFFFF FFFF	512 Mbytes (2^{29} bytes)

32-bit Kernel Mode, 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 32-bit *kuseg* virtual address space is selected; it covers the full 2^{31} bytes (2 Gbytes) 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.

When ERL = 1 in the *Status* register, the user address region becomes a 2^{31}-byte unmapped (that is, mapped directly to physical addresses) uncached address space. See the Cache Error exception in Chapter 5 for more information.

32-bit Kernel Mode, 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, 32-bit *kseg0* virtual address space is selected; it is the 2^{29}-byte (512-Mbyte) kernel physical space. References to *kseg0* are not mapped through the TLB; the physical address selected is defined by subtracting 0x8000 0000 from the virtual address. The *K0* field of the *Config* register, described in this chapter, controls cacheability and coherency.

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

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

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

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

32-bit Kernel Mode, 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-Mbyte) supervisor virtual space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

32-bit Kernel Mode, 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-Mbyte) kernel virtual space. The virtual address is extended with the contents of the 8-bit ASID field to form a unique virtual address.

Table 4-4 64-bit Kernel Mode Segments

Address Bit Values	Status Register Is One Of These Values: KSU, EXL, ERL	KX	Segment Name	Address Range	Segment Size
A(63:62) = 00_2	KSU = 00_2 *or* EXL = 1 *or* ERL =1	1	*xksuseg*	0x0000 0000 0000 0000 through 0x0000 00FF FFFF FFFF	1 Tbyte (2^{40} bytes)
A(63:62) = 01_2		1	*xksseg*	0x4000 0000 0000 0000 through 0x4000 00FF FFFF FFFF	1 Tbyte (2^{40} bytes)
A(63:62) = 10_2		1	*xkphys*	0x8000 0000 0000 0000 through 0xBFFF FFFF FFFF FFFF	8 2^{36}-byte spaces
A(63:62) = 11_2		1	*xkseg*	0xC000 0000 0000 0000 through 0xC000 00FF 7FFF FFFF	(2^{40}–2^{31}) bytes
A(63:62) = 11_2 A(61:31) = -1		1	*ckseg0*	0xFFFF FFFF 8000 0000 through 0xFFFF FFFF 9FFF FFFF	512 Mbytes (2^{29} bytes)
A(63:62) = 11_2 A(61:31) = -1		1	*ckseg1*	0xFFFF FFFF A000 0000 through 0xFFFF FFFF BFFF FFFF	512 Mbytes (2^{29} bytes)
A(63:62) = 11_2 A(61:31) = -1		1	*cksseg*	0xFFFF FFFF C000 0000 through 0xFFFF FFFF DFFF FFFF	512 Mbytes (2^{29} bytes)
A(63:62) = 11_2 A(61:31) = -1		1	*ckseg3*	0xFFFF FFFF E000 0000 through 0xFFFF FFFF FFFF FFFF	512 Mbytes (2^{29} bytes)

64-bit Kernel Mode, 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 (that is, mapped directly to physical addresses) uncached address space. See the Cache Error exception in Chapter 5 for more information.

64-bit Kernel Mode, 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.

64-bit Kernel Mode, 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^{36}-byte kernel physical spaces. Accesses with address bits 58:36 not equal to 0 cause an address error.

References to this space are not mapped; the physical address selected is taken from bits 35:0 of the virtual address. Bits 61:59 of the virtual address specify the cacheability and coherency attributes, as shown in Table 4-5.

Table 4-5 Cacheability and Coherency Attributes

Value (61:59)	Cacheability and Coherency Attributes	Starting Address
0	Reserved	0x8000 0000 0000 0000
1	Reserved	0x8800 0000 0000 0000
2	Uncached	0x9000 0000 0000 0000
3	Cacheable, noncoherent	0x9800 0000 0000 0000
4	Cacheable, coherent exclusive	0xA000 0000 0000 0000
5	Cacheable, coherent exclusive on write	0xA800 0000 0000 0000
6	Cacheable, coherent update on write	0xB000 0000 0000 0000
7	Reserved	0xB800 0000 0000 0000

64-bit Kernel Mode, 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:

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

64-bit Kernel Mode, Compatibility Spaces (*ckseg1:0, cksseg, ckseg3*)

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 address, as shown in Figure 4-6, select one of the following 512-Mbyte 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, described in this chapter, controls cacheability and coherency.
- *ckseg1*. This 64-bit virtual address space is an unmapped and uncached region, compatible with the 32-bit address model *kseg1*.
- *cksseg*. This 64-bit virtual address space is the current supervisor virtual space, compatible with the 32-bit address model *ksse*g.
- *ckseg3*. This 64-bit virtual address space is kernel virtual space, compatible with the 32-bit address model *kseg3*.

4.3 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 4-7 plus a 48-entry TLB. The sections that follow describe how the processor uses the memory management-related registers†.

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

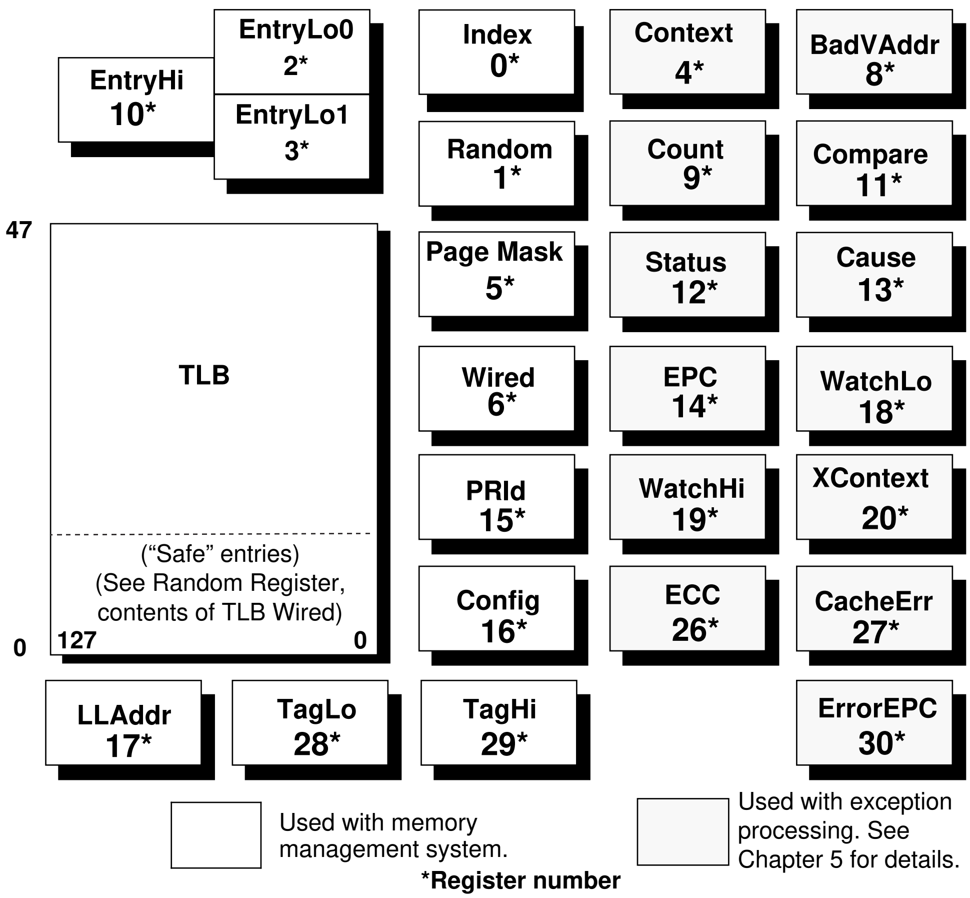


Figure 4-7 CP0 Registers and the TLB

† For a description of CP0 data dependencies and hazards, please see Appendix F.

Format of a TLB Entry

Figure 4-8 shows the TLB entry formats for both 32- and 64-bit modes. Each field of an entry has a corresponding field in the *EntryHi*, *EntryLo0*, *EntryLo1*, or *PageMask* registers, as shown in Figures 4-9 and 4-10; for example the *Mask* field of the TLB entry is also held in the *PageMask* register.

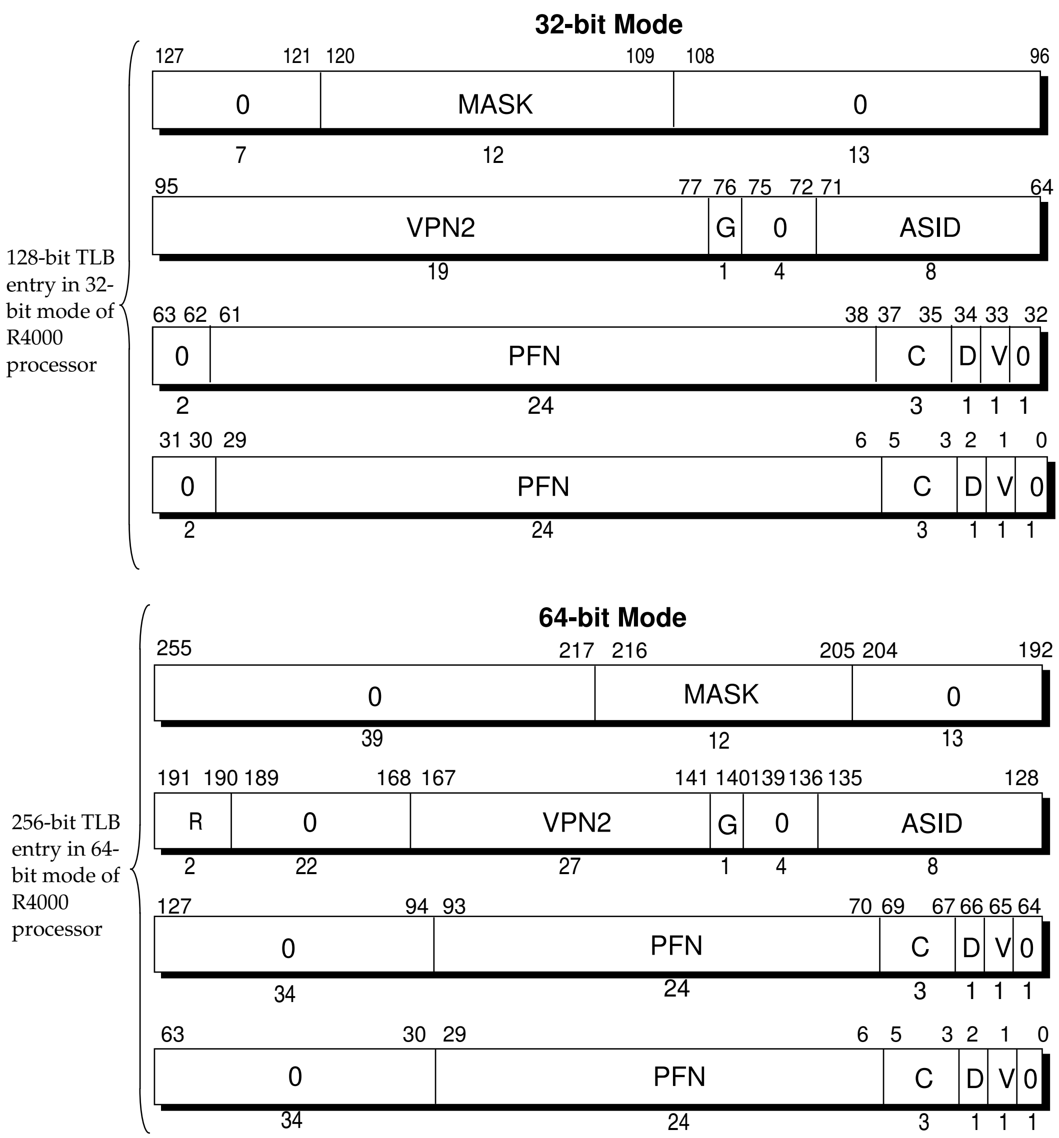


Figure 4-8 Format of a TLB Entry

The format of the *EntryHi*, *EntryLo0*, *EntryLo1*, and *PageMask* registers are nearly the same as the TLB entry. The one exception is the *Global* field (*G* bit), which is used in the TLB, but is reserved in the *EntryHi* register. Figures 4-9 and 4-10 describe the TLB entry fields shown in Figure 4-8.

PageMask Register

32-bit Mode

	31 … 25	24 … 13	12 … 0
Field	0	MASK	0
Width	7	12	13

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

EntryHi Register

32-bit Mode

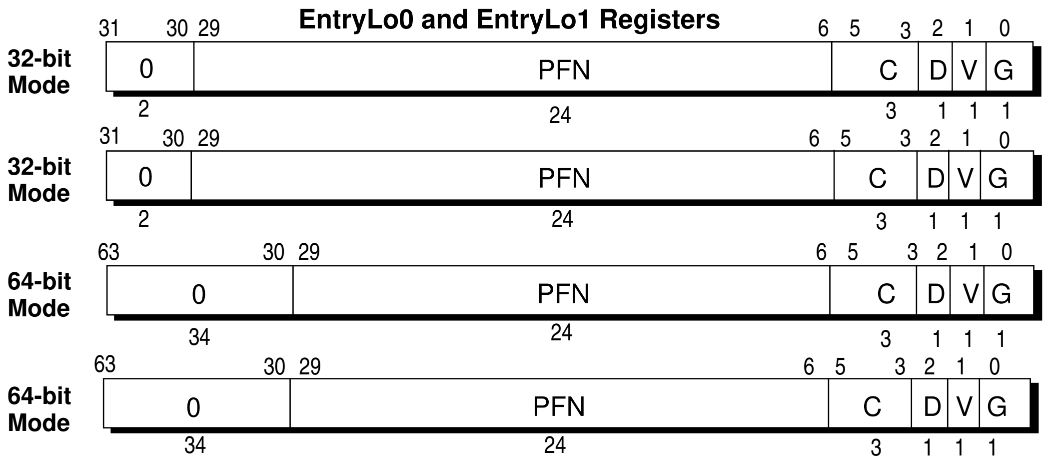
	31 … 13	12 … 8	7 … 0
Field	VPN2	0	ASID
Width	19	5	8

64-bit Mode

	63 62	61 … 40	39 … 13	12 … 8	7 … 0
Field	R	FILL	VPN2	0	ASID
Width	2	22	27	5	8

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 zeroes, and returns zeroes when read.

Figure 4-9 Fields of the PageMask and EntryHi Registers



PFN...... Page frame number; the upper bits of the physical address.
C .... Specifies the TLB page coherency attribute; see Table 4-6.
D .... Dirty. If this bit is set, the page is marked as dirty and, therefore, writable. This bit is actually a write-protect bit that software can use to prevent alteration of data.
V........... Valid. If this bit is set, it indicates that the TLB entry is valid; otherwise, a TLBL or TLBS miss occurs.
G .... Global. If this bit is set in both Lo0 and Lo1, then the processor ignores the ASID during TLB lookup.
0........... Reserved. Must be written as zeroes, and returns zeroes when read.

Figure 4-10 Fields of the EntryLo0 and EntryLo1 Registers

The TLB page coherency attribute (*C*) bits specify whether references to the page should be cached; if cached, the algorithm selects between several coherency attributes. Table 4-6 shows the coherency attributes selected by the *C* bits.

Table 4-6 TLB Page Coherency (C) Bit Values

***C*(5:3) Value**	**Page Coherency Attribute**
0	Reserved
1	Reserved
2	Uncached
3	Cacheable noncoherent (noncoherent)
4	Cacheable coherent exclusive (exclusive)
5	Cacheable coherent exclusive on write (sharable)
6	Cacheable coherent update on write (update)
7	Reserved

CP0 Registers

The following sections describe the CP0 registers, shown in Figure 4-7, that are assigned specifically as a software interface with memory management (each register is followed by its register number in parentheses).

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

Index Register (0)

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

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

Figure 4-11 shows the format of the *Index* register; Table 4-7 describes the *Index* register fields.

Index Register

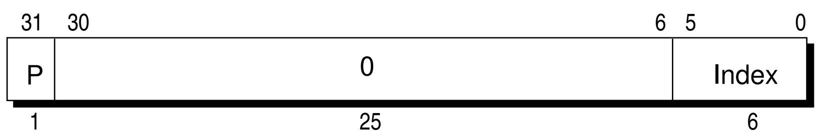


Figure 4-11 Index Register

Table 4-7 Index Register Field Descriptions

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

Random Register (1)

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

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

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

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

Figure 4-12 shows the format of the *Random* register; Table 4-8 describes the *Random* register fields.

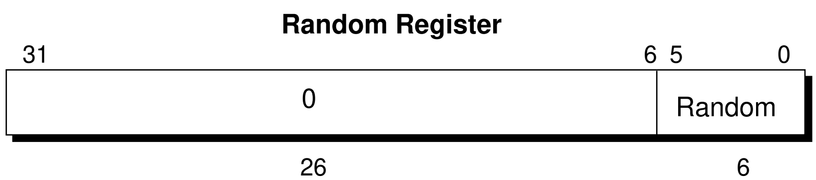


Figure 4-12 Random Register

Table 4-8 Random Register Field Descriptions

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

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 4-10 shows the format of these registers.

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, as shown in Table 4-9.

TLB read and write operations use this register as either a source or a destination; when virtual addresses are presented for translation into physical address, the corresponding bits in the TLB identify which virtual address bits among bits 24:13 are used in the comparison. When the *Mask* field is not one of the values shown in Table 4-9, the operation of the TLB is undefined.

Table 4-9 Mask Field Values for Page Sizes

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

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 4-13. Wired entries are fixed, nonreplaceable entries, which cannot be overwritten by a TLB write operation. Random entries can be overwritten.

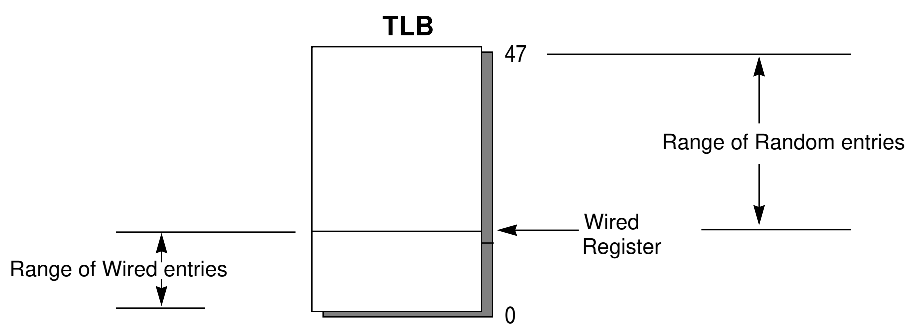


Figure 4-13 Wired Register Boundary

The *Wired* register is set to 0 upon system reset. Writing this register also sets the *Random* register to the value of its upper bound (see *Random* register, above). Figure 4-14 shows the format of the *Wired* register; Table 4-10 describes the register fields.

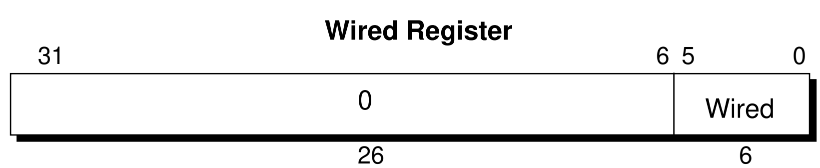


Figure 4-14 Wired Register

Table 4-10 Wired Register Field Descriptions

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

EntryHi Register (CP0 Register 10)

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

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

Figure 4-9 shows the format of this register.

When either a TLB refill, TLB invalid, or TLB modified exception occurs, the *EntryHi* register is loaded with the virtual page number (VPN2) and the ASID of the virtual address that did not have a matching TLB entry. (See Chapter 5 for more information about these exceptions.)

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 4-15 shows the format of the *PRId* register; Table 4-11 describes the *PRId* register fields.

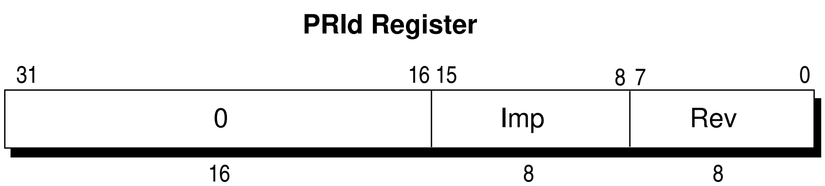


Figure 4-15 Processor Revision Identifier Register Format

Table 4-11 PRId Register Fields

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

The low-order byte (bits 7:0) of the *PRId* register is interpreted as a revision number, and the high-order byte (bits 15:8) is interpreted as an implementation number. The implementation number of the R4000 processor is 0x04. The content of the high-order halfword (bits 31:16) of the register are reserved.

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

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

Config Register (16)

The *Config* register specifies various configuration options selected on R4000 processors; Table 4-12 lists these options.

Some configuration options, as defined by *Config* bits 31:6, are set by the hardware during reset and are included in the *Config* register as read-only status bits for the software to access. Other configuration options are read/write (as indicated by *Config* register bits 5:0) and controlled by software; on reset these fields are undefined.

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

Figure 4-16 shows the format of the *Config* register; Table 4-12 describes the *Config* register fields.

Config Register

31	30 28	27 24	23 22	21	20	19 18	17	16	15	14	13	12	11 9	8 6	5	4	3	2 0
CM	EC	EP	SB	SS	SW	EW	SC	SM	BE	EM	EB	0	IC	DC	IB	DB	CU	K0
1	3	4	2	1	1	2	1	1	1	1	1	1	3	3	1	1	1	3

Figure 4-16 Config Register Format

Table 4-12 Config Register Fields

Field	Description
CM	Master-Checker Mode (1 → Master/Checker Mode is enabled).
EC	System clock ratio: 0 → processor clock frequency divided by 2 1 → processor clock frequency divided by 3 2 → processor clock frequency divided by 4 3 → processor clock frequency divided by 6 (R4400 processor only) 4 → processor clock frequency divided by 8 (R4400 processor only)
EP	Transmit data pattern (pattern for write-back data): 0 → D Doubleword every cycle 1 → DDx 2 Doublewords every 3 cycles 2 → DDxx 2 Doublewords every 4 cycles 3 → DxDx 2 Doublewords every 4 cycles 4 → DDxxx 2 Doublewords every 5 cycles 5 → DDxxxx 2 Doublewords every 6 cycles 6 → DxxDxx 2 Doublewords every 6 cycles 7 → DDxxxxxx 2 Doublewords every 8 cycles 8 → DxxxDxxx 2 Doublewords every 8 cycles
SB	Secondary Cache line size: 0 → 4 words 1 → 8 words 2 → 16 words 3 → 32 words
SS	Split Secondary Cache Mode 0 → instruction and data mixed in secondary cache (joint cache) 1 → instruction and data separated by SCAddr(17)
SW	Secondary Cache port width 0 → 128-bit data path to S-cache 1 → Reserved
EW	System Port width 0 → 64-bit 1, 2, 3 → Reserved
SC	Secondary Cache present 0 → S-cache present 1 → no S-cache present

Table 4-12 (cont.) Config Register Fields

Field Name	Description
SM	Dirty Shared coherency state 0 → Dirty Shared coherency state is enabled 1 → Dirty Shared state is disabled
BE	BigEndianMem 0 → kernel and memory are little endian 1 → kernel and memory are big endian
EM	ECC mode enable 0 → ECC mode enabled 1 → parity mode enabled
EB	Block ordering 0 → sequential 1 → sub-block
0	Reserved. Must be written as zeroes, returns zeroes when read.
IC	Primary I-cache Size (I-cache size = 2^{12+IC} bytes). In the R4000 processor, this is set to 8 Kbytes; in the R4400 processor, this is set to 16 Kbytes.
DC	Primary D-cache Size (D-cache size = 2^{12+DC} bytes). In the R4000 processor, this is set to 8 Kbytes, in the R4400 processor, this is set to 16 Kbytes.
IB	Primary I-cache line size 0 → 16 bytes 1 → 32 bytes
DB	Primary D-cache line size 0 → 16 bytes 1 → 32 bytes
CU	Update on Store Conditional 0 → Store Conditional uses coherency algorithm specified by TLB 1 → SC uses cacheable coherent update on write
K0	*kseg0* coherency algorithm (see *EntryLo0* and *EntryLo1* registers and the *C* field of Table 4-6)

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

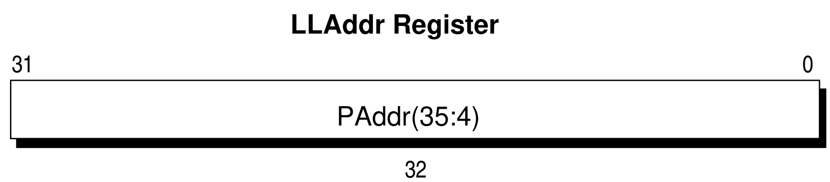


Figure 4-17 LLAddr Register Format

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

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

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

Figure 4-18 shows the format of these registers for primary cache operations. Figure 4-19 shows the format of these registers for secondary cache operations.

Table 4-13 lists the field definitions of the *TagLo* and *TagHi* registers.

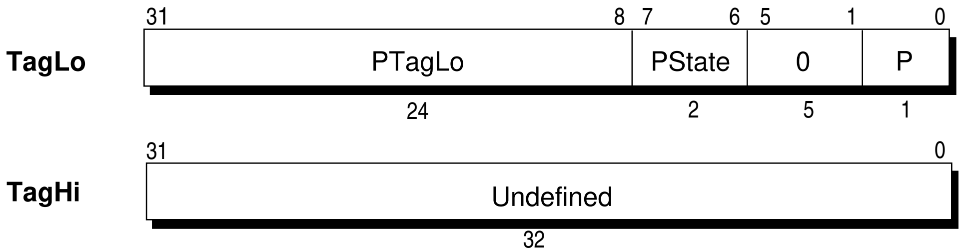


Figure 4-18 TagLo and TagHi Register (P-cache) Formats

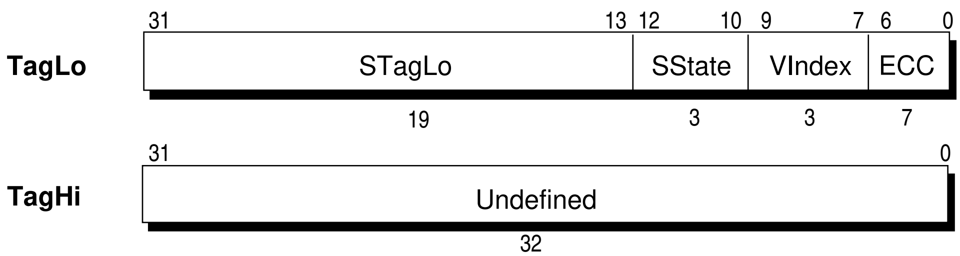


Figure 4-19 TagLo and TagHi Register (S-cache) Formats

Table 4-13 Cache Tag Register Fields

Field	Description
PTagLo	Specifies the physical address bits 35:12
PState	Specifies the primary cache state
P	Specifies the primary tag even parity bit
STagLo	Specifies the physical address bits 35:17
SState	Specifies the secondary cache state
VIndex	Specifies the virtual index of the associated Primary cache line, vAddr(14:12)
ECC	ECC for the *STag*, *SState*, and *VIndex* fields
0	Reserved. Must be written as zeroes, and returns zeroes when read.
Undefined	The TagHi register should not be used.

Virtual-to-Physical Address Translation Process

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

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

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

Figure 4-20 illustrates the TLB address translation process.

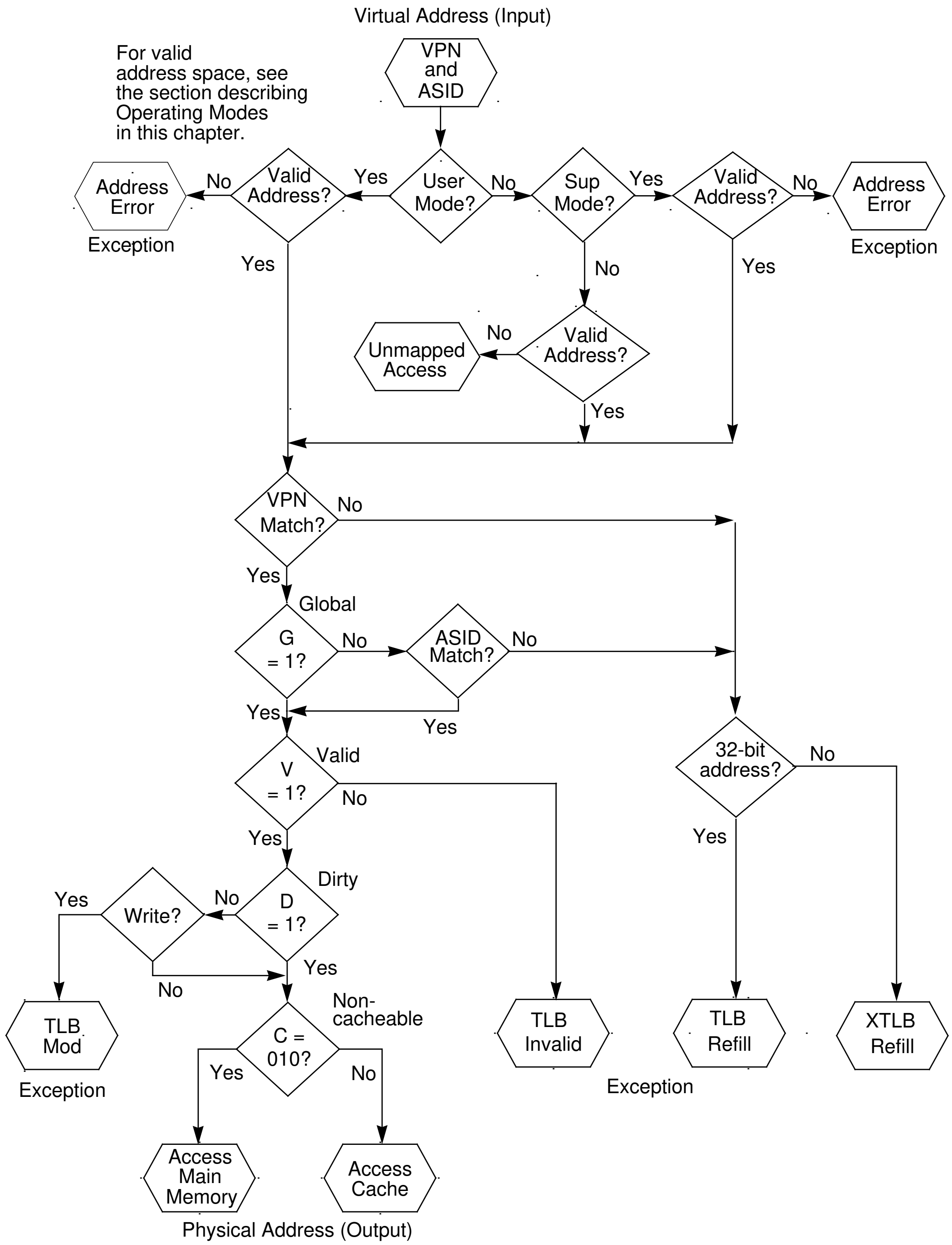


Figure 4-20 TLB Address Translation

TLB Misses

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.

TLB Instructions

Table 4-14 lists the instructions that the CPU provides for working with the TLB. See Appendix A for a detailed description of these instructions.

Table 4-14 TLB Instructions

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

† TLB miss exceptions are described in Chapter 5.