Coprocessor 0

14

This chapter describes the Coprocessor 0 operation, concentrating on the CP0 register definitions and the R10000 processor implementation of CP0 instructions.

The Coprocessor 0 (CP0) registers control the processor state and report its status. These registers can be read using MFC0 instructions and written using MTC0 instructions. CP0 registers are listed in Table 14-1.

Table 14-1 Coprocessor 0 Registers

Register No.	Register Name	Description
0	Index	Programmable register to select TLB entry for reading or writing
1	Random	Pseudo-random counter for TLB replacement
2	EntryLo0	Low half of TLB entry for even VPN (Physical page number)
3	EntryLo1	Low half of TLB entry for odd VPN (Physical page number)
4	Context	Pointer to kernel virtual PTE table in 32-bit addressing mode
5	Page Mask	Mask that sets the TLB page size
6	Wired	Number of wired TLB entries (lowest TLB entries not used for random replacement)
7	*Undefined*	*Undefined*
8	BadVAddr	Bad virtual address
9	Count	Timer count
10	EntryHi	High half of TLB entry (Virtual page number and ASID)
11	Compare	Timer compare
12	Status	Processor Status Register
13	Cause	Cause of the last exception taken
14	EPC	Exception Program Counter
15	PRId	Processor Revision Identifier
16	Config	Configuration Register (secondary cache size, etc.)
17	LLAddr	Load Linked memory address
18	WatchLo	Memory reference trap address (low bits Adr[39:32])
19	WatchHi	Memory reference trap address (high bits Adr[31:3])
20	XContext	Pointer to kernel virtual PTE table in 64-bit addressing mode
21	FrameMask	Mask the physical addresses of entries which are written into the TLB
22	BrDiag	Branch Diagnostic register
23	*Undefined*	*Undefined*
24	*Undefined*	*Undefined*
25	PC	Performance Counters
26	ECC	Secondary cache ECC and primary cache parity
27	CacheErr	Cache Error and Status register
28	TagLo	Cache Tag register - low bits
29	TagHi	Cache Tag register - high bits
30	ErrorEPC	Error Exception Program Counter

Coprocessor 0 instructions are enabled if the processor is in Kernel mode, or if bit 28 (*CU0*) is set in the *Status* register. Otherwise, executing one of these instructions generates a Coprocessor 0 Unusable exception.

14.1 Index Register (0)

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

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

Figure 14-1 shows the format of the *Index* register; Table 14-2 describes the *Index* register fields.

Index Register

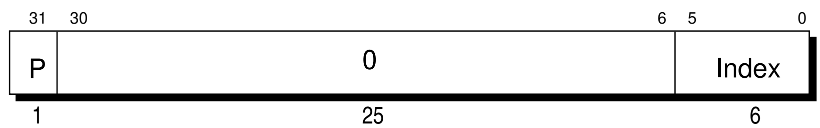


Figure 14-1 Index Register

Table 14-2 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.

14.2 Random Register (1)

The *Random* register is a read-only register of which six bits index an entry in the TLB. This register decrements when any instruction graduates at that particular cycle, and its values range between an upper and a lower bound, as follows:

- The lower bound is set by the number of TLB entries reserved for exclusive use by the operating system (the contents of the *Wired* register).
- The upper bound is set by the total number of TLB entries minus 1 (64 – 1 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 14-2 shows the format of the *Random* register; Table 14-3 describes the *Random* register fields.

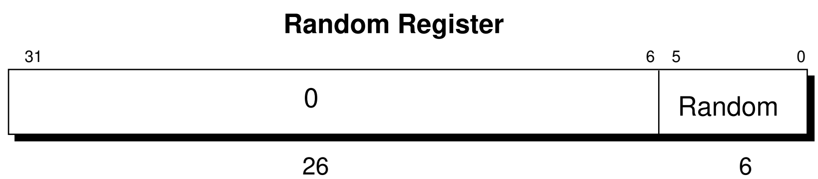


Figure 14-2 Random Register

Table 14-3 Random Register Field Descriptions

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

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

The *EntryLo* register consists of two registers with 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 14-3 shows the format of these registers.

EntryLo0 and EntryLo1 Registers

63 62	61 34	33 6	5 3	2	1	0
UC	0	PFN	C	D	V	G
		28	3	1	1	1

UC........ Uncached attribute.
PFN...... Page frame number; the upper bits of the physical address.
C .... Specifies the TLB page coherency attribute.
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 invalid exception 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 14-3 Fields of the EntryLo0 and EntryLo1 Registers

The *PFN* fields of the *EntryLo0* and *EntryLo1* registers span bits 33:6 of the 40-bit physical address.

Two additional bits for the mapped space's *uncached attribute* can be loaded into bits 63:62 of the *EntryLo* register, which are then written into the TLB with a TLB Write. During the address cycle of processor double/single/partial-word read and write requests, and during the address cycle of processor *uncached accelerated* block write requests, the processor drives the uncached attribute on SysAD[59:58] . The same *EntryLo* registers are used for the 64-bit and 32-bit addressing modes. In both modes the registers are 64 bits wide, however when the MIPS III ISA is not enabled (32-bit User and Supervisor modes) only the lower 32 bits of the *EntryLo* registers are accessible.

MIPS III is disabled when the processor is in 32-bit Supervisor or User mode. Loading of the integer registers is limited to bits *31:0*, sign-extended through bits *63:32*. *EntryLo[33:31]* or *PFN[39:38]* can only be set to all zeroes or all ones. In 32- and 64-bit modes, the *UC* and *PFN* bits of both *EntryLo* registers are written into the TLB. The *PFN* bits can be masked by setting bits in the *FrameMask* register (described in this chapter) but the *UC* bits cannot be masked or initialized in 32-bit User or Supervisor modes. In 32-bit Kernel mode, MIPS III is enabled and 64-bit operations are always available to program the *UC* bits.

There is only one *G* bit per TLB entry, and it is written with *EntryLo0[0]* and *EntryLo1[0]* on a TLB write.

14.4 Context (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 CPU 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 14-4 shows the format of the *Context* register; Table 14-4 describes the *Context* register fields.

Context Register

63 … 23	22 … 4	3 … 0
PTEBase	BadVPN2	0
41	19	4

Figure 14-4 Context Register Format

Table 14-4 Context Register Fields

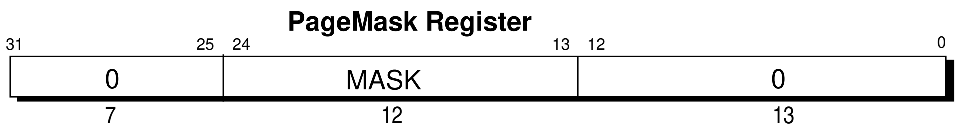
Field	Description
BadVPN2	This field is written by hardware on a miss. It contains the virtual page number (VPN) of the most recent virtual address that did not have a valid translation.
PTEBase	This field is a read/write field for use by the operating system. It is normally written with a value that 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-Kbyte 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.

14.5 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 14-5. Format of the register is shown in Figure 14-5.

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 14-5, the operation of the TLB is undefined.



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

Figure 14-5 PageMask Register

Table 14-5 Mask Field Values for Page Sizes

Page Size (Mask)	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

14.6 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 14-6. Wired entries are fixed, nonreplaceable entries, which cannot be overwritten by a TLB write operation. Random entries can be overwritten.

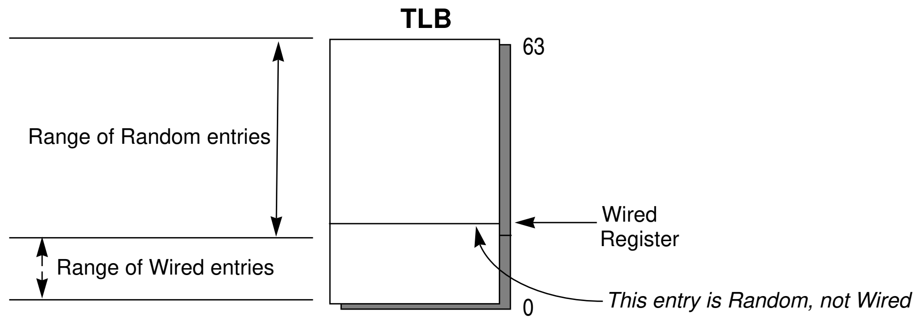


Figure 14-6 Wired Register Boundary

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 14-7 shows the format of the *Wired* register; Table 14-6 describes the register fields.

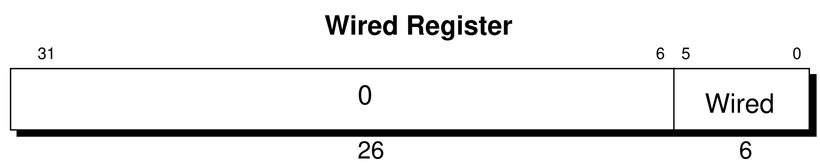


Figure 14-7 Wired Register

Table 14-6 Wired Register Field Descriptions

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

14.7 BadVAddr Register (8)

The Bad Virtual Address register (*BadVAddr*) is a read-only register that displays the most recent virtual address that caused either a TLB or Address Error exception. The *BadVAddr* register remains unchanged during Soft Reset, NMI, or Cache Error exceptions. Otherwise, the architecture leaves this register undefined.

Figure 14-8 shows the format of the *BadVAddr* register.

BadVAddr Register

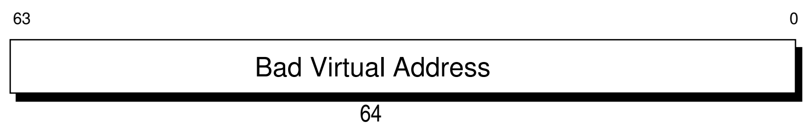


Figure 14-8 BadVAddr Register Format

14.8 Count and Compare Registers (9 and 11)

The *Count* and *Compare* registers are 32-bit read/write registers whose formats are shown in Figure 14-9.

The *Count* register acts as a real-time timer. Like the R4400 implementation, the R10000 *Count* register is incremented every *other* PClk cycle. However, unlike the R4400, the R10000 processor has no Timer Interrupt Enable boot-mode bit, so the only way to disable the timer interrupt is to negate the interrupt mask bit, *IM[7]*, in the Status register. This means the timer interrupt cannot be disabled without also disabling the *Performance Counter* interrupt, since they share *IM[7]*.

The *Compare* register can be programmed to generate an interrupt at a particular time, and is continually compared to the *Count* register. Whenever their values equal, the interrupt bit *IP[7]* in the *Cause* register is set. This interrupt bit is reset whenever the *Compare* register is written.

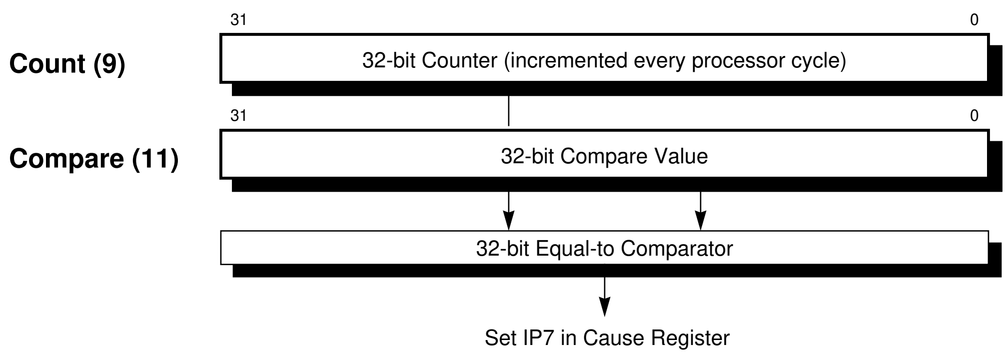


Figure 14-9 Count and Compare Registers

14.9 EntryHi Register (10)

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

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

Figure 14-10 shows the format of this register.

EntryHi Register

63 62	61 44	43 13	12 8	7 0
R	FILL	VPN2	0	ASID
2	18	31	5	8

VPN2....Virtual page number divided by two (maps to two pages); upper bits of the virtual address
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 14-10 EntryHi Register

In 64-bit addressing mode, the *VPN2* field contains bits 43:13 of the 44-bit virtual address.

In 32-bit addressing mode only the lower 32 bits of the *EntryHi* register are used, so the format remains the same as in the R4400 processor's 32-bit addressing mode. The *FILL* field is ignored on write and read as zeroes, as it was in the R4400 implementation.

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.

14.10 Status Register (12)

The *Status* register (SR) is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. The following list describes the more important *Status* register fields; Figure 14-11 shows the format of the entire register, and Table 14-8 describes the *Status* register fields.

Some of the important fields include:

- The 4-bit *Coprocessor Usability* (*CU*) field controls the usability of 4 possible coprocessors. Regardless of the *CU0* bit setting, CP0 is always usable in Kernel mode. The *XX* bit enables the MIPS IV ISA in User mode.
- By default, the R10000 processor implements the same user instruction set as the R4400 processor. To enable execution of the MIPS IV instructions in User mode, the *MIPS IV User Mode* bit, *(XX*) of the CP0 *Status* register must be set.

  The MIPS IV instruction extension uses COP1X as the opcode; this designation was COP3 in the R4400 processor. For this reason the *CU3* bit is omitted in the R10000 processor, and is used as the *XX* bit. In *Kernel* and *Supervisor* modes, the state of the *XX* bit is ignored, and MIPS IV instructions are always available.

  Mode bit settings are shown in Table 14-7; dashes in the table represent *don't cares*.

Table 14-7 ISA and Status Register Settings for User, Supervisor and Kernel Mode Operations

Mode	UX	SX	KX	XX	MIPS II	MIPS III	MIPS IV
User	0	-	-	0	Yes	No	No
	0	-	-	1	Yes	No	Yes
	1	-	-	0	Yes	Yes	No
	1	-	-	1	Yes	Yes	Yes
Supervisor	-	0	-	-	Yes	No	Yes
	-	1	-	-	Yes	Yes	Yes
Kernel	-	-	-	-	Yes	Yes	Yes

NOTE: Operation with the MIPS IV ISA does not assume or require that the MIPS III instruction set or 64-bit addressing be enabled — *KX, SX* and *UX* may all be set to zero.

- The *Reduced Power (RP*) bit is reserved and should be zero. The R10000 processor does not define a reduced power mode.
- The *Reverse-Endian (RE)* bit, bit 25, reverses the endianness of the machine. The processor can be configured as either little-endian or big-endian at system reset; reverse-endian selection is available in Kernel and Supervisor modes, and in the User mode when the *RE* bit is 0. Setting the *RE* bit to 1 inverts the User mode endianness.
- The 9-bit *Diagnostic Status* (*DS*) field is used for self-testing, and checks the cache and virtual memory system. This field is described in Table 14-9 and Figure 14-12.
- The 8-bit *Interrupt Mask* (*IM*) field controls the enabling of eight interrupt conditions. Interrupts must be enabled before they can be asserted, and the corresponding bits are set in both the *Interrupt Mask* field of the *Status* register and the *Interrupt Pending* field of the *Cause* register.
- The processor mode is undefined if the *KSU* field is set to 3 (11_2). The R10000 processor implements this as User mode.

Status Register

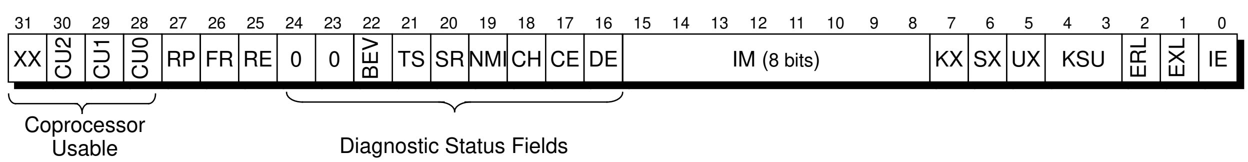


Figure 14-11 Status Register

Status Register Fields

Table 14-8 describes the *Status* register fields.

Table 14-8 Status Register Fields

Field	Description
XX	Enables execution of MIPS IV instructions in User mode. 1 → MIPS IV instructions usable 0 → MIPS IV instructions unusable
CU	Controls the usability of each of the four coprocessor unit numbers. CP0 is always usable when in Kernel mode, regardless of the setting of the CU_0 bit. 1 → usable 0 → unusable
RP	In the R4400 processor, this bit enables reduced-power operation by reducing the internal clock frequency. In the R10000 processor, this bit should be set to zero.
FR	Enables additional floating-point registers 0 → 16 registers 1 → 32 registers
RE	*Reverse-Endian* bit, valid in User mode.
DS	*Diagnostic Status* field (see Figure 14-12).
IM	*Interrupt Mask*: controls the enabling of each of the external, internal, and software interrupts. An interrupt is taken if interrupts are enabled, and the corresponding bits are set in both the *Interrupt Mask* field of the *Status* register and the *Interrupt Pending* field of the *Cause* register. 0 → disabled 1→ enabled
KX	Enables 64-bit addressing in Kernel mode. The extended-addressing TLB refill exception is used for TLB misses on kernel addresses. 0 → 32–bit 1 → 64–bit
SX	Enables 64-bit addressing and operations in Supervisor mode. The extended-addressing TLB refill exception is used for TLB misses on supervisor addresses. 0 → 32–bit 1 → 64–bit
UX	Enables 64-bit addressing and operations in User mode. The extended-addressing TLB refill exception is used for TLB misses on user addresses. 0 → 32–bit 1 → 64–bit

Table 14-8 (cont.) Status Register Fields

Field	Description
KSU	Mode bits 11_2 → Undefined (implemented as User mode) 10_2 → User 01_2 → Supervisor 00_2 → Kernel
ERL	Error Level; set by the processor when Reset, Soft Reset, NMI, or Cache Error exception are taken. 0 → normal 1 → error
EXL	Exception Level; set by the processor when any exception other than Reset, Soft Reset, NMI, or Cache Error exception are taken. 0 → normal 1 → exception
IE	Interrupt Enable 0 → disable all interrupts 1 → enables all interrupts

Diagnostic Status Field

The 9-bit *Diagnostic Status* (*DS*) field is used for self-testing, and checks the cache and virtual memory system. This field is described in Table 14-9 and shown Figure 14-12.

Some of the important *DS* fields include:

- In the R4400, the *TS* bit of the diagnostic field indicates a TLB *shutdown* has occurred due to matching of multiple virtual page entries during address translation. In the R10000 processor, the *TS* bit indicates a TLB write has introduced an entry that would allow matching of more than one virtual page entry during translation. In this case, the TLB entries that allow the multiple matches, even in the *Wired* area, are invalidated before the new TLB entry is written. This prevents multiple matches during address translation.

  The *TS* bit is updated for each TLB write. It can also be read and written by software (in the R4400, the *TS* bit is read-only); to clear the *TS* bit one needs to write a 0 into it. As in the R4400, Reset/Soft Reset/NMI exceptions also clear the *TS* bit.

- The *NMI* bit is new to the R10000 processor; it distinguishes between Soft Reset and NMI exceptions. Both exceptions set the *SR* bit to 1; the NMI exception sets the *NMI* bit to 1, whereas the Soft Reset exception sets it to 0.
- The *CE* bit is reserved in the R10000 processor and should be a 0.

24 23	22	21	20	19	18	17	16
0	BEV	TS	SR	NMI	CH	CE	DE
2	1	1	1	1	1	1	1

Figure 14-12 Diagnostic Status Field

Table 14-9 Status Register Diagnostic Status Bits

Bit	Description
BEV	Controls the location of TLB refill and general exception vectors. 0 → normal 1→ bootstrap
TS	This bit is set when a TLB write presents an entry that matches any other virtual page entry in the TLB. Should this occur, any TLB entries that allow multiple matches, even in the *Wired* area, are invalidated before this new entry can be written into the TLB. This prevents multiple matches during address translation. 0 → normal 1→ TLB shutdown has occurred.
SR	1→ Indicates a Soft Reset or NMI exception.
NMI	1→ Indicates a nonmaskable interrupt has occurred. Used to distinguish between a Soft Reset and a nonmaskable interrupt in a Soft Reset exception.
CH	Hit (tag match and valid state) or miss indication for last CACHE Hit Invalidate, Hit Write Back Invalidate for a secondary cache. 0 → miss 1 → hit
CE	Reserved in the R10000, and should be set to 0.
DE	Specifies that cache parity or ECC errors cannot cause exceptions. 0 → parity/ECC remain enabled 1 → disables parity/ECC
0	Reserved. Must be written as zeroes, and returns zeroes when read.

Coprocessor Accessibility

Three *Status* register *CU* bits control coprocessor accessibility: *CU0*, *CU1*, and *CU2* enable coprocessors 0, 1, and 2, respectively. If a coprocessor is unusable, any instruction that accesses it generates an exception.

The following describes the coprocessor implementations and operations on the R10000:

- Coprocessor 0 is always enabled in kernel mode, regardless of the *CU0* bit.
- Coprocessor 1 is the floating-point coprocessor. If *CU1* is 0 (disabled), all floating-point instructions generate a Coprocessor Unusable exception. In MIPS IV, the COP3 instruction is replaced with a second floating-point instruction, COP1X. In addition, new functions are added to COP1 (see Chapter 15). The floating-point branch conditional and compare instructions are expanded to use the eight Floating-Point *Status* register condition bits, instead of the original single bit. If any of these extra bits are referenced (cc > 0) when not using the MIPS IV ISA, an Unimplemented Instruction exception is taken. The integer conditional move (MOVC) instruction tests a floating-point condition bit; it causes a coprocessor unusable exception if coprocessor 1 is disabled.
- Coprocessor 2 is defined, but does not exist in the R10000; its instructions (COP2, LWC2, LDC2, SWC2, SDC2) always cause an exception, but the exception code depends upon whether the coprocessor, as indicated by *CU2*, is enabled.
- Coprocessor 3 has been removed from the MIPS III ISA, and is no longer defined. If MIPS IV is disabled, the coprocessor 3 instruction (COP3) always causes a Reserved Instruction exception.

14.11 Cause Register (13)

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

Figure 14-13 shows the fields of this register; Table 14-10 describes the *Cause* register fields. A 5-bit exception code (*ExcCode*) indicates one of the causes, as listed in Table 14-11.

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 14-10 Cause Register Fields

Field	Description
BD	Indicates whether the last exception taken occurred in a branch delay slot. 1 → delay slot 0 → normal
CE	Coprocessor unit number referenced when a Coprocessor Unusable exception is taken. This bit is undefined for any other exception.
IP	Indicates an interrupt is pending. This bit remains unchanged for NMI, Soft Reset, and Cache Error exceptions. 1 → interrupt pending 0 → no interrupt
ExcCode	Exception code field (see Table 14-11)
0	Reserved. Must be written as zeroes, and returns zeroes when read.

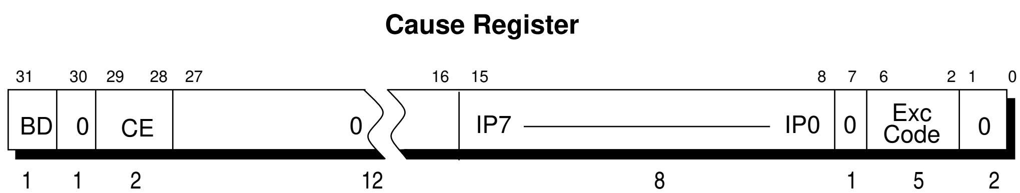


Figure 14-13 Cause Register Format

Table 14-11 Cause Register ExcCode Field

Exception Code Value	Mnemonic	Description
0	Int	Interrupt
1	Mod	TLB modification exception
2	TLBL	TLB exception (load or instruction fetch)
3	TLBS	TLB exception (store)
4	AdEL	Address error exception (load or instruction fetch)
5	AdES	Address error exception (store)
6	IBE	Bus error exception (instruction fetch)
7	DBE	Bus error exception (data reference: load or store)
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Arithmetic Overflow exception
13	Tr	Trap exception
14	–	Reserved
15	FPE	Floating-Point exception
16–22	–	Reserved
23	WATCH	Reference to *WatchHi*/*WatchLo* address
24–30	–	Reserved
31	–	Reserved

14.12 Exception Program Counter (14)

The Exception Program Counter (*EPC*)[†] is a read/write register that contains the address at which processing resumes after an exception has been serviced.

For synchronous exceptions, the *EPC* register contains either:

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

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

Figure 14-14 shows the format of the *EPC* register.

EPC Register

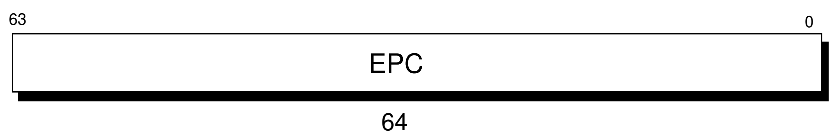


Figure 14-14 EPC Register Format

† The *ErrorEPC* register provides a similar capability, described later in this chapter.

14.13 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 14-15 shows the format of the *PRId* register; Table 14-12 describes the *PRId* register fields.

PRId Register

31 … 16	15 … 8	7 … 0
0	Imp (0x09)	Rev
16	8	8

Figure 14-15 Processor Revision Identifier Register Format

Table 14-12 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 R10000 processor is 0x09. 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, software should not rely on the revision number in the *PRId* register to characterize the chip.

14.14 Config Register (16)

The R10000 processor's *Config* register has a different format from that of the R4400, since the R10000 processor has different mode bits and configurations, however some fields are still compatible: *K0*, *DC*, *IC*, and *BE*. The value of bits 24:0 are taken directly from the Mode bit settings during a reset sequence; refer to Table 8-1 for these bit definitions. Table 14-13 shows the R10000 *Config* register fields, along with values which are hardwired into the register at boot time; Figure 14-16 shows the *Config* register format.

Table 14-13 Config Register Field Definitions

Field	Bits	Name	Hardwired Values
K0	2:0	Coherency algorithm 000_2 → reserved 001_2 → reserved 010_2 → uncached 011_2 → cacheable noncoherent 100_2 → cacheable coherent exclusive 101_2 → cacheable coherent exclusive on write 110_2 → reserved 111_2 → uncached accelerated	
DN	4:3	Device number	
CT	5	CohPrcReqTar	
PE	6	PrcElmReq	
PM	8:7	PrcReqMax	
EC	12:9	SysClkDiv	
SB	13	SCBlkSize	
SK	14	SCCorEn	
BE	15	MemEnd	
SS	18:16	SCSize	
SC	21:19	SCClkDiv	
	25:22	Reserved	0
DC	28:26	Primary data cache size (hardwired to 011_2)	32 Kbytes
IC	31:29	Primary instruction cache size (hardwired to 011_2)	32 Kbytes

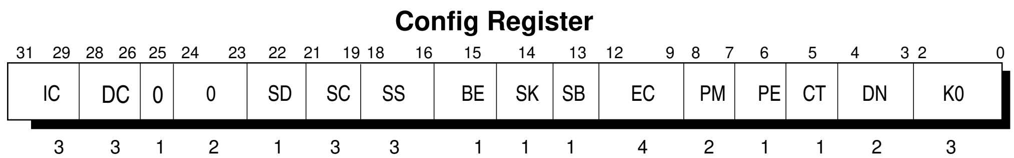


Figure 14-16 Config Register Format

14.15 Load Linked Address (LLAddr) Register (17)

Physical addresses for Load Link instructions are no longer written into this register. *LLAddr* is implemented as a read/write scratch register used for NT compatibility.

Figure 14-17 shows the format of the *LLAddr* register.

LLAddr Register

31 … 0
R/W (NT)
32

Figure 14-17 LLAddr Register Format

14.16 WatchLo (18) and WatchHi (19) Registers

WatchHi and *WatchLo* are 32-bit read/write registers which contain a physical address of a doubleword location in main memory. If enabled, any attempt to read or write this location causes a Watch exception. This feature is used for debugging.

Bits 7:0 of the *WatchHi* register contain bits 39:32 of the trap physical address, shown in Figure 14-18. The WatchLo register contains physical address bits 31:3. The remaining bits of the register are ignored on write and read as zero.

Table 14-14 describes the *WatchLo* and *WatchHi* register fields.

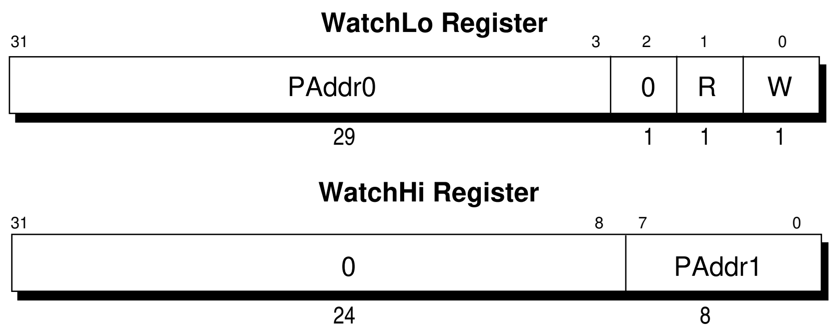


Figure 14-18 WatchLo and WatchHi Register Formats

Table 14-14 WatchHi and WatchLo Register Fields

Field	Description
PAddr1	Bits 39:32 of the physical address
PAddr0	Bits 31:3 of the physical address
R	Trap on load references if set to 1
W	Trap on store references if set to 1
0	Ignored on write and read as zero.

14.17 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 no longer shares the information provided in the *BadVAddr* register, as it did in the R4400.

The *XContext* register is for use with the XTLB refill handler, which loads TLB entries for references to a 64-bit address space, and is included solely for operating system use. The operating system sets the PTE base field in the register, as needed. Normally, the operating system uses the *Context* register to address the current page map, which resides in the kernel-mapped segment *kseg3*.

Figure 14-19 shows the format of the *XContext* register; Table 14-15 describes the *XContext* register fields.

XContext Register

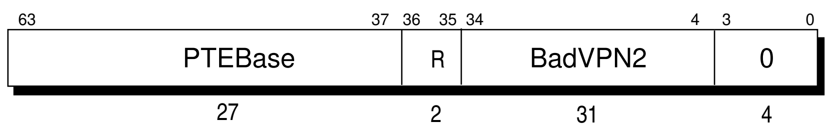


Figure 14-19 XContext Register Format

The 31-bit *BadVPN2* field holds bits 43:13 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-Kbyte 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 14-15 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.

14.18 FrameMask Register (21)

The *FrameMask* register is new with the R10000 processor. It masks bits of the *EntryLo0* and *EntryLo1* registers so that these masked bits are not passed to the TLB while doing a TLB write (either TLBWI or TLBWR).

A zero in the *FrameMask* register allows its corresponding bit in the *EntryLo*[*1,0]* registers to pass to the TLB; a one in the *FrameMask* register masks off its corresponding bit in the *EntryLo* registers and passes a zero to the TLB. Bits 15:0 of the *FrameMask* register control bits 33:18 of the *EntryLo* registers.

The remaining bits of this register are ignored on write and read as zeroes. The content of this register is set to zero after a processor reset or a power-up event.

Figure 14-20 shows the *FrameMask* register format.

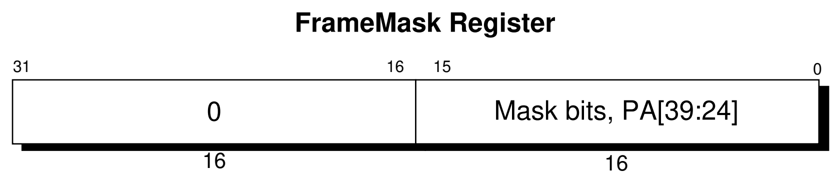


Figure 14-20 Framemask Register Format

14.19 Diagnostic Register (22)

CP0 register 22, the *Diagnostic* register, is a new 64-bit register for processor-specific diagnostic functions. (Since this register is designed for local use, the diagnostic functions are subject to change without notice.) Currently, this register helps test the ITLB, branch caches, and the branch prediction scheme. In addition, it provides choices for branch prediction algorithms, to help diagnostic program writing.

The ten fields of the *Diagnostic* register, shown in Figure 14-21, are described below. All fields are read-only (all writes are ignored).

ITLBM: this field is a 4-bit read-only counter. This field is incremented by one for each ITLB miss, and any overflow is ignored. Its value is undefined during reset, and its value is meaningless when used in an unmapped space.

BSIdx: this field defines the entry in the branch stack to be used for the latest conditional branch decoded. Its value is meaningless if the latest branch was an unconditional branch.

DBRC: this field disables the use of the branch return cache (BRC).

BRCV: this field indicates whether or not the branch return cache (BRC) is valid. BRC has only one entry (four instructions).

BRCW: this field indicates whether or not the latest branch (JAL, JALR RX, BGEZAL, BGEZALL, BLTZAL, or BLTZALL) caused a write into BRC. It is not affected by any other type of branch.

BRCH: this field indicates whether or not the latest branch (JR r31 or JALR rx,r31) has a BRC hit. It not affected by any other type of branch.

MP: this field indicates whether or not the latest conditional branch verified was mispredicted.

BPMode: this is a read-write field for branch prediction algorithm control.

00_2: 2-bit counter scheme

01_2: all conditional branches are predicted not taken

10_2: all conditional branches are predicted taken

11_2: forward conditional branches are predicted not taken and backward conditional branches are predicted taken.

The default mode is 00 on processor reset.

BPState: this field contains the new 2-bit state for a conditional branch after it is verified. It is also used to hold the 2-bit state to read/write when a branch prediction table read/write operation is executed.

BPIdx: this field contains the index to the *BPT* for *BPT* read/write/initialization operations, and should contain *VA[11:3]* of the branch for BPT read/write operations. The upper six bits of the *BPIdx* field contain the line address for BPT line initialization operations; the lower three bits of *BPIdx* are ignored.

BPOp: this field indicates the following BPT operations:

00_2: BPT read

01_2: BPT write

10_2: initializes BPT line to all zeroes (strongly not taken)

11_2: initializes BPT line to all ones (strongly taken).

Figure 14-21 shows the format of the *Diagnostic* register.

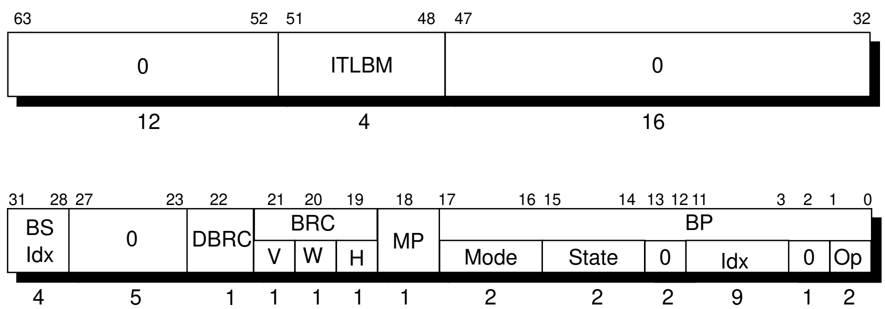


Figure 14-21 Diagnostic Register Format

14.20 Performance Counter Registers (25)

The R10000 processor defines two performance counters and two associated control registers, which are mapped into CP0 register 25. An encoding in the MTC0/MFC0 instructions on register 25 indicates which counter or control register is used.

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 counter asserts an interrupt, *IP[7]*, when its most significant bit (bit 31) becomes one (the counter overflows) and the associated performance control register enables the interrupt.

The counting continues after counter overflow whether or not an interrupt is signalled.

The format of the control registers are shown in Figure 14-22.

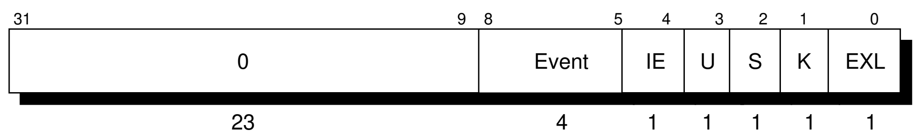


Figure 14-22 Control Register Format

The fields of the *Control* register are:

- The *Event* field specifies the event to be counted, listed in Table 14-16.

Table 14-16 Counter Events

Event	Counter 0	Counter 1
0	Cycles	Cycles
1	Instructions issued	Instructions graduated
2	Load/prefetch/sync/CacheOp issued	Load/prefetch/sync/CacheOp graduated
3	Stores issued	Stores graduated
4	Store conditional issued	Store conditional graduated
5	Failed store conditional	Floating-point instructions graduated
6	Branches decoded	Quadwords written back from primary data cache
7	Quadwords written back from secondary cache	TLB refill exceptions
8	Correctable ECC errors on secondary cache data	Branches mispredicted
9	Instruction cache misses	Data cache misses
10	Secondary cache misses (instruction)	Secondary cache misses (data)
11	Secondary cache way mispredicted (instruction)	Secondary cache way mispredicted (data)
12	External intervention requests	External intervention request hits in secondary cache
13	External invalidate requests	External invalidate request hits in secondary cache
14	Virtual coherency condition	Stores or prefetches with store hint to *CleanExclusive* secondary cache blocks
15	Instructions graduated	Stores or prefetches with store hint to *Shared* secondary cache blocks

- The *IE* bit enables the assertion of *IP[7]* when the associated counter overflows.
- The *U, S, K,* and *EXL* bits indicate the processor modes in which the event is counted: U is user mode; S is supervisor mode; K is kernel mode when *EXL* and *ERL* both are set to 0; the system is in kernel mode and handling an exception when *EXL* is set to 1, as shown in Table 14-20.

These modes can be set individually; for example, one could set all four bits to count a certain event in all processor modes except during a cache error exception.

The following rules apply:

- R10000 defines "issue" as the point in time that an instruction becomes active and is issued to a functional unit for execution.
- R10000 defines "graduation" as the point in time that the functional unit successfully completes the execution of an instruction. It is the oldest active instruction.
- The number of executed branches is the difference between decoded branches and mispredicted branches. Although newer branches may be speculatively decoded before an older branch is resolved as mispredicted, these newer branch instructions, nullified by the older misprediction, are not counted as mispredicted.
- When an integer multiply or divide instruction graduates, it is counted as two graduated instructions.
- Any instruction that sets the FP *Status* register bits (EVZOUI) is counted as a graduated floating point instruction.
- TLB misses are counted as either nascent or within the TLB handler.

The performance counters and associated control registers are written by using an MTC0 instruction, as shown in Table 14-17.

Table 14-17 Writing Performance Registers Using MTC0

Opcode[15:11]	Opcode[1:0]	Operation
11001	00	Move to Performance Control 0
11001	01	Move to Performance Counter 0
11001	10	Move to Performance Control 1
11001	11	Move to Performance Counter 1

The performance counters and associated control registers are read by using a MFC0 instruction, as shown in Table 14-18.

Table 14-18 Reading Performance Registers Using MFC0

Opcode[15:11]	Opcode[1:0]	Operation
11001	00	Move from Performance Control 0
11001	01	Move from Performance Counter 0
11001	10	Move from Performance Control 1
11001	11	Move from Performance Counter 1

The format of the performance control registers are shown in Table 14-19.

Table 14-19 Performance Control Register Format

[8:5]	[4]	[3:0]
Event select	IP[7] interrupt enable	Count enable (U/S/K/EXL)

The count enable field specifies whether counting is to be enabled during User, Supervisor, Kernel, and/or Exception level mode. Any combination of count enable bits may be asserted.

All unused bits in the performance control registers are reserved.

All counting is disabled when the *ERL* bit of the CP0 *Status* register is asserted.

Table 14-20 defines the operation of the count enable bits of the performance control registers.

Table 14-20 Count Enable Bit Definition

Count Enable Bit	Count Qualifier (CP0 ***Status*** Register Fields)
U	KSU = 2 (User mode), EXL = 0, ERL = 0
S	KSU = 1 (Supervisor mode), EXL = 0, ERL = 0
K	KSU = 0 (Kernel mode), EXL = 0, ERL = 0
EXL	EXL = 1, ERL = 0

The following rules apply:

- The performance counter registers may be preloaded with an MTC0 instruction, and counting is enabled by asserting one or more of the count enable bits in the performance control registers.
- The interrupt enable bit must be asserted to cause *IP[7]*.
- To determine the cause of the interrupt, the interrupt handler routine must query the following:
  - the performance counter register
  - the interrupt enable bit of the associated performance control register of both counters
- If neither of the counters caused the interrupt, *IP[7]* must be the result of the CP0 *Count* register matching the CP0 *Compare* register.

14.21 ECC Register (26)

The R10000 processor implements a 10-bit read/write *ECC* register which is used to read and write the secondary cache data ECC or the primary cache data parity bits. (Tag ECC and parity are loaded to and stored from the *TagLo* register.) Unlike the R4400, the only CacheOps that use *ECC* register are *Index Load Data* and *Index Store Data*.

In the R4400, both the primary instruction and data caches are parity byte-protected.

In the R10000 processor, the following protection schemes are used:

- The primary instruction cache is word-protected (where one word contains 36 bits), and one parity bit is used for each instruction word (*IP* in Figure 14-23).
- The primary data cache is byte-protected, with four bits used for each 32-bit data word (*DP* in Figure 14-23).
- Each quadword of the secondary cache data uses nine bits of ECC and one bit of parity (*SP* and *ECC* in Figure 14-23).

The primary instruction CacheOps load or store one instruction word at a time; therefore, one bit is used in the ECC register. The primary data CacheOps load or store four bytes at a time; therefore, four bits are used in the ECC register. The secondary CacheOps use ECC[9] as the parity bit and ECC[8:0] as the 9-bit ECC. For the *Index Store Data* cacheops, the unused bits are ignored. For *Index Load Data* CacheOps, the unused a bits are with zeroes.

Figure 14-23 shows the format of the *ECC* register; Table 14-21 describes the register fields.

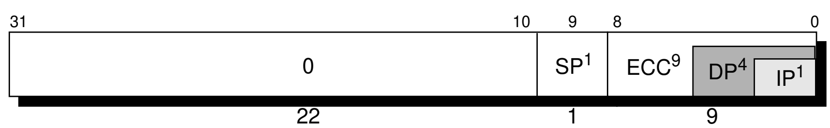


Figure 14-23 ECC Register Format

Table 14-21 ECC Register Fields

Field	Description
SP	A 1-bit field specifying the parity bit read from or written to a secondary cache.
ECC	An 9-bit field specifying the ECC bits read from or written to a secondary cache.
DP	An 4-bit field specifying the parity bits read from or written to a primary data cache.
IP	An 1-bit field specifying the parity bit read from or written to a primary instruction cache.
0	Reserved. Must be written as zeroes, and returns zeroes when read.

14.22 CacheErr Register (27)

The *CacheErr* register is a 32-bit read-only register that handles ECC errors in the secondary cache or system interface, and parity errors in the primary caches.

R10000 processor correction policy is as follows:

- Parity errors cannot be corrected.
- Single-bit ECC errors can be corrected by hardware without taking a Cache Error exception.
- Double-bit ECC errors can be detected but not corrected by hardware.
- All uncorrectable errors take Cache Error exceptions unless the *DE* bit of the *Status* register is set.
- As in the R4400, cache errors are imprecise.

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

CacheErr Register Format for Primary Instruction Cache Errors

Figure 14-24 shows the format of the *CacheErr* register when a primary instruction cache error occurs.

31 30	29	28	27 26	25 24	23 22	21 14	13 6	5 0
00	EW	0	D	TA	TS	0	PIdx	0
2	1	1	2	2	2	8	8	6

Figure 14-24 CacheErr Register Format for Primary Instruction Cache Errors

EW: set when *CacheErr* register is already holding the values of a previous error

D: data array error (way1 || way0)

TA: tag address array error (way1 || way0)

TS: tag state array error (way1 || way0)

PIdx: primary cache virtual block index, VA[13:6]

CacheErr Register Format for Primary Data Cache Errors

Figure 14-25 shows the format of the *CacheErr* register when a primary data cache error occurs.

31 30	29	28	27 26	25 24	23 22	21 20	19 14	13 3	2 0
01	EW	EE	D	TA	TS	TM	0	PIdx	0
2	1	1	2	2	2	2	6	11	3

Figure 14-25 CacheErr Register Format for Primary Data Cache Errors

EW: set when *CacheErr* register is already holding the values of a previous error

EE: tag error on an inconsistent block

D: data array error (way1 || way0)

TA: tag address array error (way1 || way0)

TS: tag state array error (way1 || way0)

TM: tag mod array error (way1 || way0)

PIdx: primary cache virtual double word index, VA[13:6]

CacheErr Register Format for Secondary Cache Errors

Figure 14-26 shows the format of the *CacheErr* register when a secondary cache error occurs.

31 30	29	28	27 26	25 24	23	22 6	5 0
10	EW	0	D	TA	0	SIdx	0
2	1	1	2	2	1	17	6

Figure 14-26 CacheErr Register Format for Secondary Cache Errors

EW: set when *CacheErr* register is already holding the values of a previous error

D: uncorrectable data array error (way1 || way0)

TA: uncorrectable tag array error (way1 || way0)

SIdx: secondary cache physical block index (PA[22:6] for 16-word block size or PA[22:7] for 32-word block size)

CacheErr Register Format for System Interface Errors

Figure 14-27 shows the format of the *CacheErr* register when a System interface error occurs.

31 30	29	28	27 26	25	24	23	22 6	5 0
11	EW	EE	D	SA	SC	SR	SIdx	0
2	1	1	2	1	1	1	17	6

Figure 14-27 CacheErr Register Format for System Interface Errors

EW: set when *CacheErr* register is already holding the values of a previous error

EE: data error on a *CleanExclusive* or *DirtyExclusive*

D: uncorrectable system block data response error (way1 || way0)

SA: uncorrectable system address bus error

*S*C: uncorrectable system command bus error

*S*R: uncorrectable system response bus error

SIdx: secondary cache physical block index

14.23 TagLo (28) and TagHi (29) Registers

The *TagHi* and *TagLo* registers are 32-bit read/write registers used to hold the following:†

- the primary cache tag and parity
- the secondary cache tag and ECC
- the data in primary or secondary caches for certain CacheOps

TagHi/Lo formats in the R10000 processor differ from those in the R4400 due to changes in CacheOps and cache architecture. R10000 formats depend on the type of CacheOp executed and the cache to which it is applied. The reserved fields are read as zeroes after executing an *Index Load Tag* or an *Index Load Data* CacheOp and ignored when executing an *Index Store Tag* or an *Index Store Data* CacheOp.

To ensure NT kernel compatibility, the *TagLo* register is implemented as a 32-bit read/write register. The value written by an MTC0 instruction can be retrieved by a MFC0 instruction, unless an intervening CACHE instruction has modified the content.

This section gives the TagLo and TagHi register formats for the following CacheOp and cache combinations:

- CacheOp is Index Load/Store Tag
  - primary instruction cache operation
  - primary data cache operation
  - secondary cache operation
- CacheOp is Index Load/Store Data
  - primary instruction cache operation
  - primary data cache operation
  - secondary cache operation

† To ensure NT kernel compatibility, the TagLo register is implemented as a 32-bit read/write register. The value written by a MTC0 instruction can be retrieved by a MFC0 instruction, unless intervening CACHE instructions modify the content.

CacheOp is Index Load/Store Tag

This section describes the three states of the *TagLo* and *TagHi* registers, when the CacheOp is an *Index Load/Store Tag* for the following operations:

- primary instruction cache operation
- primary data cache operation
- secondary cache operation

Primary Instruction Cache Operation

If the CacheOp is an *Index Load/Store Tag* for a primary instruction cache operation, the fields of the *TagHi* and *TagLo* registers are defined as follows:

PTag0: contains physical address bits [35:12] stored in the cache tag

PState: contains the primary instruction cache state for the line, as follows:

1 = *Valid*

0 = *Invalid*

LRU: indicates whether or not this line is the least recently used of the set

SP: state even parity bit for the *PState* field

TP: tag even parity bit.

PTag1: contains physical address bits [39:36] stored in the cache tag

Figure 14-28 shows the fields of the *TagHi* and *TagLo* registers.

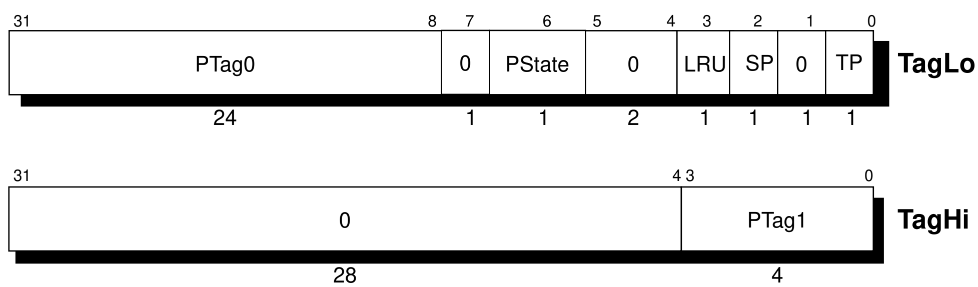


Figure 14-28 TagHi/Lo Register Fields in Primary Instruction Cache When CacheOp is Index Load/Store Tag

Primary Data Cache Operation

If the CacheOp is an *Index Load/Store Tag* for primary data cache operations, the fields of the *TagHi* and *TagLo* registers are defined as follows:

State Modifier: holds the status of the line, as follows:

001_2 = neither refilled or written

010_2 = this line may have been written and inconsistent from the secondary cache (*W* bit)

100_2 = this line is being refilled (*Refill* bit).

PTag1: contains physical address bits [39:36] stored in the cache tag

PTag0: contains physical address bits [35:12] stored in the cache tag

PState: together with the *Refill* bit of the *State Modifier* in the *TagHi* register, *PState* determines the state of the cache block in the primary data cache, as shown in Table 14-22.

Table 14-22 PState Field Definition in TagHi/Lo Registers, For Primary Data Cache Operation When CacheOp is Index Load/Store Tag

PState	Refill=0	Refill=1
00_2	Invalid	Refill *clean* (block is being refilled)
01_2	Shared	Upgrade Share (converting *shared* to *dirty*)
10_2	Clean Exclusive	Upgrade Clean (converting *clean* to *dirty*).
11_2	Dirty Exclusive	Refill *dirty* (block is being refilled for a store)

LRU: indicates whether this line was the least recently used of the set

SP: state even parity bit for the *PState* field and the *Way* bit

Way: indicates which secondary cache set contains the primary cache line for this tag

TP: tag even parity bit.

Figure 14-29 shows the fields of the *TagHi* and *TagLo* registers.

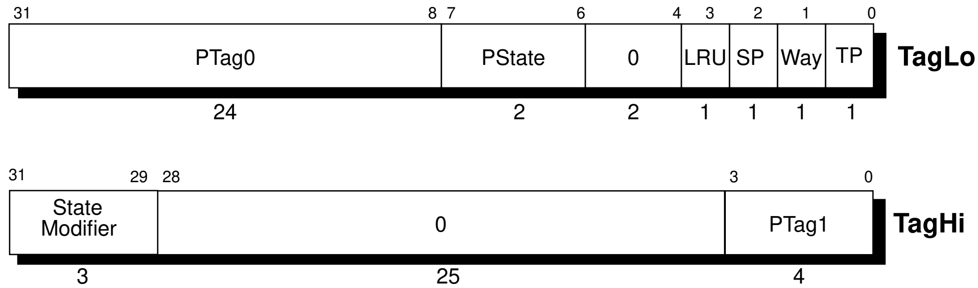


Figure 14-29 TagHi/Lo Register Fields in Primary Data Cache When CacheOp is Index Load/Store Tag

Secondary Cache Operation

If the CacheOp is an *Index Load/Store Tag* for secondary cache operations, the fields of the *TagHi* and *TagLo* registers are defined as follows:

STag0: contains physical address bits [35:18] stored in the cache tag

SState: contains the secondary cache state of the line*, as follows:*

- 00_2 = *Invalid*
- 01_2 = *Shared*
- 10_2 = *Clean Exclusive*
- 11_2 = *Dirty Exclusive*

VIndex (virtual index): contains only two bits of significance since the32 Kbyte 2-way set associative primary caches are addressed using only two untranslated address bits (VA[13:12]) plus the offset within the virtual page.

ECC: contains the ECC for the *STag*, *SState* and *VIndex* fields.

MRU: indicates whether this line was the most recently used in the set

STag1: contains the physical address bits [39:36] stored in the cache tag.

Figure 14-30 shows the fields of the *TagHi* and *TagLo* registers.

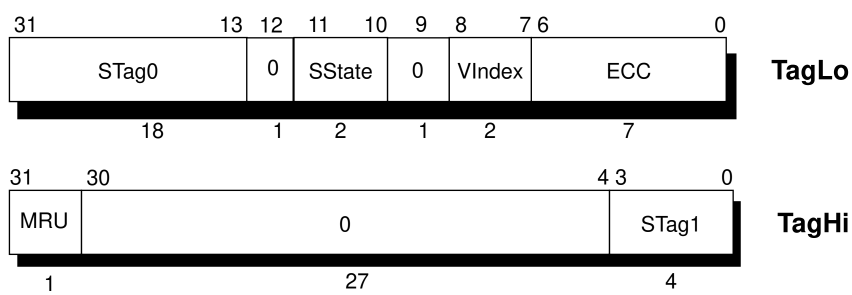


Figure 14-30 TagHi/Lo Register Fields in Secondary Cache When CacheOp is Index Load/Store Tag

CacheOp is Index Load/Store Data

This section describes the following three states of the *TagLo* and *TagHi* registers, when the CacheOp is an *Index Load/Store Data*:

- primary instruction cache operation
- primary data cache operation
- secondary cache operation

Primary Instruction Cache Operation

If the CacheOp is an *Index Load/Store Data* for the primary instruction cache, the *TagHi* register stores the most significant four bits of a 36-bit instruction, as shown in Figure 14-31; the rest of the instruction is stored in the *TagLo* register.

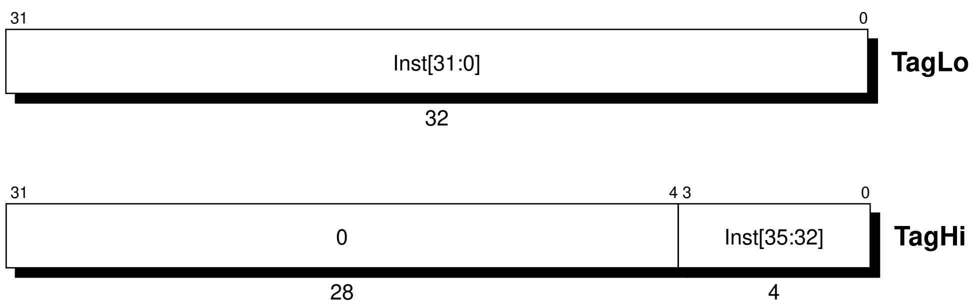


Figure 14-31 TagHi/Lo Register Fields in Primary Instruction Cache When CacheOp is Index Load/Store Data

Primary Data Cache Operation

If the CacheOp is *Index Load/Store Data* for primary data cache, the *TagHi* register is not used. The *TagLo* registers contains a 32-bit data word for the cache operation, as shown in Figure 14-32.

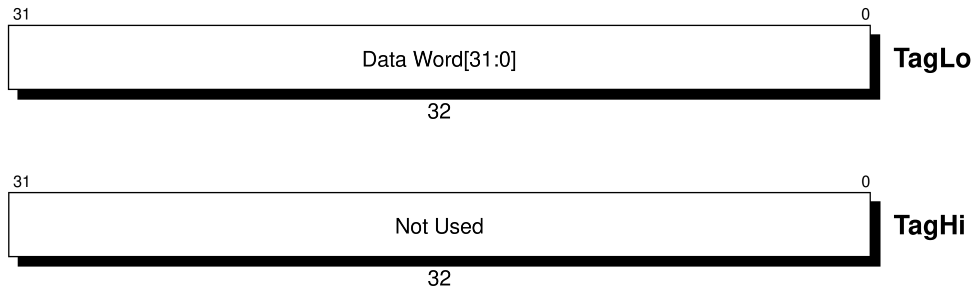


Figure 14-32 TagHi/Lo Register Fields in Primary Data Cache When CacheOp is Index Load/Store Data

Secondary Cache Operation

If the CacheOp is *Index Load/Store Data* for the secondary cache, a doubleword of data is required for the CacheOp. The *TagHi* register stores the upper 32 bits of the doubleword and the *TagLo* register stores the lower 32 bits, as shown below in Figure 14-33.

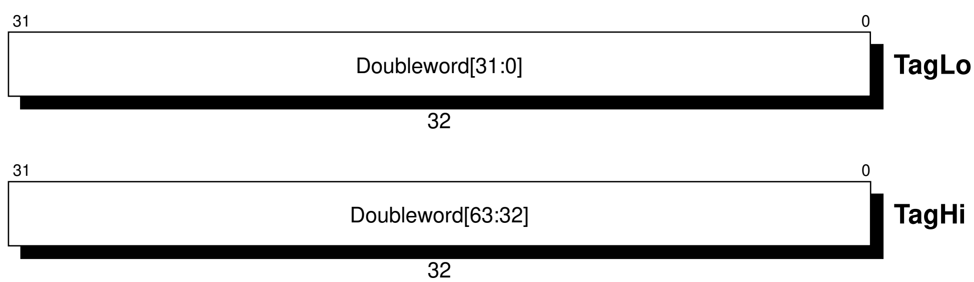


Figure 14-33 TagHi/Lo Register Fields in Secondary Cache When CacheOp is Index Load/Store Data

14.24 ErrorEPC Register (30)

The *ErrorEPC* register is similar to the *EPC* register, except that *ErrorEPC* is used on ECC and 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. Figure 14-34 shows the format of the *ErrorEPC* register.

ErrorEPC Register

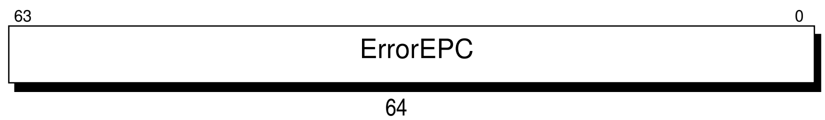


Figure 14-34 ErrorEPC Register Format

14.25 CP0 Instructions

Table 14-23 lists the CP0 instructions defined for the R10000 processor. Since they are implementation dependent, they are included here and not in the MIPS ISA manual.

Table 14-23 CP0 Instructions

OpCode	Description	ISA
CACHE	Cache Operation	III
DMFC0	Doubleword Move From CP0	III
DMTC0	Doubleword Move To CP0	III
ERET	Exception Return	III
MFC0	Move from CP0	I
MTC0	Move to CP0	I
TLBP	Probe TLB for Matching Entry	I
TLBR	Read Indexed TLB Entry	I
TLBWI	Write Indexed TLB Entry	I
TLBWR	Write Random TLB Entry	I

Hazards

The processor detects most of the pipeline hazards in hardware, including CP0 hazards and load hazards. No NOP instructions are required to correct instruction sequences.

Branch on Coprocessor 0

On the R4400 processor, CacheOps that hit in the specified cache set the *CH* bit in the Diagnostic field of the CP0 *Status* register (bit 18). Though it was undocumented, this bit could be tested by the *Branch on Coprocessor 0* instructions (*bc0t*, *bc0f*, *bc0tl*, *bc0fl*).

The R10000 processor also implements the *CH* bit but it is not associated with a Coprocessor 0 condition. Instead, execution of a branch on Coprocessor 0 instruction takes a Reserved Instruction exception.

CP0 Move Instructions

The R10000 processor implements Coprocessor 0 move instructions, MTC0, MFC0, DMTC0, and DMFC0, exactly the same as in the R4400 processor, even though some operations are undefined during certain conditions. The exact operations of CP0 move instructions on 32/64-bit CP0 registers are summarized Table 14-24.

Table 14-24 CP0 Move Instructions

Instruction	CP0 Register Size	MIPS 3 Enable?	Operation
MFC0 rt,rd	32 or 64	Don’t care	$rt \leftarrow rd_{31}^{32} \Vert rd_{31..0}$
MTC0 rt,rd	32	Don’t care	$rd \leftarrow rt_{31..0}$
	64	Don’t care	$rd \leftarrow rt_{63..0}$
DMFC0 rt,rd	32	Yes	undefined ($rt \leftarrow 0^{32} \Vert rd_{31..0}$)
	64	Yes	$rt \leftarrow rd_{63..0}$
	32 or 64	No	Reserved Instruction exception
DMTC0 rt,rd	32	Yes	undefined ($rd \leftarrow rt_{31..0}$)
	64	Yes	$rd \leftarrow rt_{63..0}$
	32 or 64	No	Reserved Instruction exception.

The returned value of MFC0/DMFC0 from a non-existing CP0 register is undefined.

CACHE Instruction

CACHE Cache CACHE

31–26	25–21	20–16	15–0
CACHE 1 0 1 1 1 1	base	op (see Table below)	offset
6	5	5	16

Format: CACHE op, offset(base)

Description:

The 16 bit *offset* is sign-extended and added to the contents of general register *base* to form a CacheOp virtual address (VA). The VA is translated to a physical address (PA) through the TLB, and the 5-bit opcode (decoded inTable 14-25) specifies a cache operation for that address, together with the affected cache. Operation of this instruction on any combination not listed in the tables below is undefined. The operation of this instruction on uncached addresses is also undefined.

More detailed descriptions of the CacheOps listed below are given separately, in Chapter 10.

Table 14-25 CACHE Instruction Op Field Encoding

Op Field	CACHE Instruction Variation	Target Cache
00000	Index Invalidate	(I)
00100	Index Load Tag	(I)
01000	Index Store Tag	(I)
10000	Hit Invalidate	(I)
10100	Cache Barrier	
11000	Index Load Data	(I)
11100	Index Store Data	(I)
00001	Index WriteBack Invalidate	(D)
00101	Index Load Tag	(D)
01001	Index Store Tag	(D)
10001	Hit Invalidate	(D)
10101	Hit WriteBack Invalidate	(D)
11001	Index Load Data	(D)
11101	Index Store Data	(D)
00011	Index WriteBack Invalidate	(S)
00111	Index Load Tag	(S)
01011	Index Store Tag	(S)
10011	Hit Invalidate	(S)
10111	Hit WriteBack Invalidate	(S)
11011	Index Load Data	(S)
11111	Index Store Data	(S)

CACHE Cache (continued) CACHE

Fill, *Create Dirty*, *Hit WriteBack* and *Hit Set Virtual* are not supported in the R10000 processor.

The R10000 processor adds two new CacheOps: *Index Load Data* (110_2) and *Index Store Data* (111_2). These changes are also reflected in the CP0 *TagHi*, *TagLo* and *ECC* registers.

The primary instruction and data caches have a block size of 16 words and 32 bytes (8 data words), respectively.

> NOTE: A 32-bit instruction is predecoded into a 36-bit instruction word before entering the primary instruction cache. The instruction fetch addresses remain the same and are not affected by the predecode.

The secondary cache, a unified cache, has a block size of either 64 or 128 bytes, configurated during reset. For a cache of $2^{CACHESIZE}$ bytes with $2^{BLOCKSIZE}$ bytes per tag,

$$VA_{CACHESIZE\text{-}2..BLOCKSIZE}$$

specifies the block for the primary cache, and

$$PA_{CACHESIZE\text{-}2..BLOCKSIZE}$$

specifies the block for the secondary cache.

For the Index CacheOps, address bit 0 is used to specify the way, 0 or 1, for the CacheOp. For this reason, bit 0 is not checked for alignment-type Address Error exception for the Index CacheOps. For CacheOps that access data in caches,

$$VA_{BLOCKSIZE\text{-}1..2}$$

specifies a word within a block for primary caches, and

$$PA_{BLOCKSIZE\text{-}1..3}$$

specifies a doubleword in the secondary cache.

A cache *hit* accesses the specified cache as normal data references, and performs the specified operation if the cache block contains valid data at the specified physical address. If the cache line is invalid or contains a differing physical address (a cache *miss*), no operation is performed. Since the R10000 processor uses 2-way set associative caches, the Hit operation performs tag comparison in both ways of the cache. No index needs to be provided for such CacheOps. If both ways register a hit, the execution of the CacheOp is undefined.

CACHE — Cache (continued) — CACHE

Write back from the primary data cache goes to the secondary cache, and write back from the secondary cache goes to the system interface. The primary data cache is written back to the secondary cache before the secondary cache is written back to the system interface; the address to be written is based by the cache tag, rather than the translated PA from the CacheOp instruction. A secondary cache write back also interrogates the primary data cache for any dirty inconsistent data.

When a line is invalidated in the secondary cache, all subset lines in the primary caches are also invalidated.

CacheOps are serialized with respect to cached loads/stores and CP0 instructions. Therefore, in general, there are no hazards for CacheOps. However, if the CacheOps modify the current instruction fetching stream, they may not work properly since the instruction fetch pipeline usually prefetches and buffers instructions and CacheOps are not serialized with respect to the instruction fetch pipeline. Programmers should be aware of such potential hazards; one solution is to put a COP0 instruction after the CacheOp to prevent the speculative execution and force the CacheOp to complete, and then use a Jump Register instruction to flush the instruction fetch pipeline. Succeeding instructions will then be re-fetched from caches.

If CP0 is not usable, a Coprocessor Unusable exception is taken. CacheOps may induce Address Error or TLBL exceptions (Refill or Invalid) during address translation, but never take a TLBS or Mod exception. The virtual address is used to index the cache for an Index CacheOp, but need not match the cache physical tag; unmapped addresses may be used to avoid TLB exceptions.

The R10000 processor does not support the *CE* bit, and programmers must supply correct parity bits or ECC for some CacheOps.

The R10000 processor supports the *CH* bit for secondary CacheOps, Hit Invalidate, and Hit WriteBack Invalidate. As in the R4400, a hit sets the *CH* bit of the *Status* register, and a miss resets it. This bit is readable and writable by software.

For a detailed description of the individual CacheOps, see Chapter 10.

Operation:

32, 64	T:	vAddr ← $((\text{offset}_{15})^{48} \mathbin{\|\|} \text{offset}_{15...0})$ + GPR[base]
		(pAddr, uncached) ← AddressTranslation (vAddr, DATA)
		CacheOp (op, vAddr, pAddr)

Exceptions:

Coprocessor unusable exception

DMFC0 Instruction

DMFC0 — Doubleword Move From System Control Coprocessor — DMFC0

31–26	25–21	20–16	15–11	10–0
COP0 0 1 0 0 0 0	DMF 0 0 0 0 1	rt	rd	0 0 0 0 0 0 0 0 0 0 00
6	5	5	5	11

Format: DMFC0 rt, rd

Description:

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

This operation is defined for the R10000 operating in 64-bit mode and in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception. All 64-bits of the general register destination are written from the coprocessor register source. The operation of DMFC0 on a 32-bit coprocessor 0 register is undefined.

Operation:

64	T:	data ← CPR[0,rd]
	T+1:	GPR[rt] ← data

Exceptions:

Coprocessor unusable exception

Reserved instruction exception (R10000 in 32-bit user mode
R10000 in 32-bit supervisor mode)

DMTC0 Instruction

DMTC0 — Doubleword Move To System Control Coprocessor — DMTC0

31–26	25–21	20–16	15–11	10–0
COP0 0 1 0 0 0 0	DMT 0 0 1 0 1	rt	rd	0 0 0 0 0 0 0 0 0 0 00
6	5	5	5	11

Format: DMTC0 rt, rd

Description:

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

This operation is defined for the R10000 operating in 64-bit mode or in 32-bit kernel mode. Execution of this instruction in 32-bit user or supervisor mode causes a reserved instruction exception.

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

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

Operation:

```
64      T:      data ← GPR[rt]
        T+1:    CPR[0,rd] ← data
```

Exceptions:

Coprocessor unusable exception (R10000 in 32-bit user mode
R10000 in 32-bit supervisor mode)

ERET Instruction

ERET Exception Return ERET

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

Format: ERET

Description:

ERET is the R10000 instruction for returning from an interrupt, exception, or error trap. Unlike a branch or jump instruction, ERET does not execute the next instruction.

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

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

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

If there is no exception (*EXL*=0 and *ERL*=0 in the *Status* register), execution of an ERET instruction is meaningless.

Execution of an ERET when *ERL*=0, regardless of the state of *EXL*, sets *EXL* to 0 and a jump is taken to the address presently held in the *EPC* register, even when there is no exception.

Operation:

32, 64	T:	if $SR_2 = 1$ then
		    PC $\leftarrow$ ErrorEPC
		    SR $\leftarrow SR_{31...3} \Vert 0 \Vert SR_{1...0}$
		else
		    PC $\leftarrow$ EPC
		    SR $\leftarrow SR_{31...2} \Vert 0 \Vert SR_0$
		endif
		LLbit $\leftarrow$ 0

Exceptions:

Coprocessor unusable exception

MFC0 Instruction

MFC0 — Move From System Control Coprocessor — MFC0

31–26	25–21	20–16	15–11	10–0
COP0 0 1 0 0 0 0	MF 0 0 0 0 0	rt	rd	0 000 0000 0000
6	5	5	5	11

Format: MFC0 rt, rd

Description:

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

Operation:

32	T:	data $\leftarrow$ CPR[0,rd]
	T+1:	GPR[rt] $\leftarrow$ data
64	T:	data $\leftarrow$ CPR[0,rd]
	T+1:	GPR[rt] $\leftarrow (data_{31})^{32} \parallel data_{31...0}$

Exceptions:

Coprocessor unusable exception

Move To/From the Performance Counter

The R10000 processor defines two performance counters, and their associated event specifier registers, which are mapped into the CP0 register 25. The following instructions are used to perform an MTC0 to or an MFC0 from a performance counter or an event specifier register.

MFPC — Move from Performance Counter — MFPC

31–26	25–21	20–16	15–11	10–6	5–1	0
COP0	00000	rt	11001	0	reg	1
6	5	5	5	5	5	1

MTPC — Move to Performance Counter — MTPC

31–26	25–21	20–16	15–11	10–6	5–1	0
COP0	00100	rt	11001	0	reg	1
6	5	5	5	5	5	1

MFPS — Move from Performance Event Specifier — MFPS

31–26	25–21	20–16	15–11	10–6	5–1	0
COP0	00000	rt	11001	0	reg	1
6	5	5	5	5	5	1

MTPS — Move to Performance Event Specifier — MTPS

31–26	25–21	20–16	15–11	10–6	5–1	0
COP0	00100	rt	11001	0	reg	1
6	5	5	5	5	5	1

Format:

MFPC	rt, reg	— Move from performance counter
MTPC	rt, reg	— Move to performance counter
MFPS	rt, reg	— Move from performance event specifier
MTPS	rt, reg	— Move to performance event specifier

reg can be either a performance counter or an event specifier; only register 0 and 1 are valid in the R10000 implementation.

MTC0 Instruction

MTC0 Move To System Control Coprocessor MTC0

31–26	25–21	20–16	15–11	10–0
COP0 0 1 0 0 0 0	MT 0 0 1 0 0	rt	rd	0 0 0 0 0 0 0 0 0 0 00
6	5	5	5	11

Format: MTC0 rt, rd

Description:

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

Operation:

32, 64	T:	data ← GPR[rt]
	T+1:	CPR[0,rd] ← data

Exceptions:

Coprocessor unusable exception

TLBP Instruction

TLBP — Probe TLB For Matching Entry — TLBP

31–26	25	24–6	5–0
COP0 0 1 0 0 0 0	CO 1	0 000 0000 0000 0000 0000	TLBP 0 0 1 0 0 0
6	1	19	6

Format: TLBP

Description:

The *Index* register is loaded with the address of the TLB entry whose contents match the contents of the *EntryHi* register. If no TLB entry matches, the high-order bit of the *Index* register is set to 0x80000000, as it is in the R4400 processor.

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

Operation:

```
32   T:   Index← 1 || 0^25 || undefined^6
          for i in 0...TLBEntries–1
               if (TLB[i]_95...77 = EntryHi_31...12) and (TLB[i]_76 or
               (TLB[i]_71...64 = EntryHi_7...0)) then
                    Index ← 0^26 || i_5...0
               endif
          endfor
64   T:   Index← 1 || 0^25 || undefined^6
          for i in 0...TLBEntries–1
               if (TLB[i]_171...141 and not (0^15 || TLB[i]_216...205))
                = EntryHi_43...13) and not (0^15 || TLB[i]_216...205)) and
               (TLB[i]_140 or (TLB[i]_135...128 = EntryHi_7...0)) then
                    Index ← 0^26 || i_5...0
               endif
          endfor
```

Exceptions:

Coprocessor unusable exception

TLBR Instruction

TLBR Read Indexed TLB Entry TLBR

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

Format: TLBR

Description:

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

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

In the R4400, this instruction had to be executed in unmapped spaces, and in the R10000 processor it can be executed in unmapped spaces without any hazard. In addition, TLBR can be executed in mapped spaces.

Operation:

32 T: $\text{PageMask} \leftarrow \text{TLB[Index}_{5...0}]_{127...96}$
$\text{EntryHi} \leftarrow \text{TLB[Index}_{5...0}]_{95...64}$ and not $\text{TLB[Index}_{5...0}]_{127...96}$
$\text{EntryLo1} \leftarrow \text{TLB[Index}_{5...0}]_{63...32}$
$\text{EntryLo0} \leftarrow \text{TLB[Index}_{5...0}]_{31...0}$

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

Exceptions:

Coprocessor unusable exception

TLBWI Instruction

TLBWI — Write Indexed TLB Entry — TLBWI

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

Format: TLBWI

Description:

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

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

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

In the R4400, this instruction had to be executed in unmapped spaces, and in the R10000 processor it can be executed in unmapped spaces without any hazard.

There is no hazard to executing a TLB write in mapped space unless the write affects those instructions that have been fetched and buffered by the processor. If necessary, a flush to the instruction-fetch pipeline, such as execution of a jump register instruction, after a TLB write can avoid this hazard.

In the R4400 processor, a TLB write instruction is used to write the whole page frame number from the *EntryLo* registers to the TLB entry. Depending on the page size specified in the corresponding *PageMask* register, the lower bits of PFN may not be used for address translation. In the R10000 processor, the lower bits not used for address translation are forced to zeroes during a TLB write. This does not affect TLB address translation, however a TLB read may not retrieve what was originally written.

Operation:

32, 64 T: $\text{TLB[Index}_{5...0}] \leftarrow$
PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0

Exceptions:

Coprocessor unusable exception

TLBWR Instruction

TLBWR — Write Random TLB Entry — TLBWR

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

Format: TLBWR

Description:

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

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

In the R4400, this instruction had to be executed in unmapped spaces, and in the R10000 processor it can be executed in unmapped spaces without any hazard.

There is no hazard to executing a TLB write in mapped space unless the write affects those instructions that have been fetched and buffered by the processor. If necessary, a flush to the instruction-fetch pipeline, such as execution of a jump register instruction, after a TLB write can avoid this hazard.

In the R4400 processor, a TLB write instruction is used to write the whole page frame number from the *EntryLo* registers to the TLB entry. Depending on the page size specified in the corresponding *PageMask* register, the lower bits of PFN may not be used for address translation. In the R10000 processor, the lower bits not used for address translation are forced to zeroes during a TLB write. This does not affect TLB address translation, however a TLB read may not retrieve what was originally written.

Operation:

32, 64 T: $\text{TLB[Random}_{5...0}] \leftarrow$
PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0

Exceptions:

Coprocessor unusable exception