10. CACHE Instructions

This chapter describes the CacheOps (CACHE†) used in the R10000 processor.

The format of the CACHE instruction is:

```
CACHE op, offset(base)
```

In a CACHE instruction, the 16-bit offset is sign-extended and added to the contents of the general register *base* to form a Virtual Address (VA). The VA is translated to a Physical Address (PA) using the TLB. The 5-bit sub-opcode specifies a cache instruction variation for that address.

†*CacheOp* and *CACHE instruction* are used interchangeably in this text.

10.1 Notes on CACHE Instruction Operations

This section describes the operations of the CACHE instructions in the R10000 processor.

> **NOTE:** The operation of any operation/cache combination not listed below is undefined, and the operation of this instruction on uncached addresses is also undefined.

Virtual Address

The CACHE instruction uses the following portions of the VA to specify a primary cache block and way:

- **VA[13:5]** defines a 32-byte block in the primary data cache array.
- **VA[13:6]** defines a 64-byte block in the primary instruction cache array.
- In both cases, **VA[0]** defines the way needed by Index operations.

Since **VA[0]** is used to indicate the way, it does not cause alignment errors.

When accessing data in the primary caches, **VA[Blocksize-1]** is also used to read or write a specific word.

Physical Address

The CACHE instruction uses the following portions of the PA to specify a secondary cache block and way**:**

- **PA[Size of secondary cache - 2:Blocksize of secondary cache]** is used to access the secondary cache.
- **PA[0]** is used to specify the way needed by Index operations.

Since **PA[0]** is used to indicate the way during CACHE Index operations, alignment errors are suppressed.

When accessing data in the secondary cache, **PA[Blocksize-1:3]** is also used to read or write a specific doubleword.

CP0 Not Usable

If the CP0 is not usable (if not in Kernel mode, *CU0* must be set in the *Status* register for CP0 to be usable), a Coprocessor Unusable exception is taken.

TLB Refill and TLB Invalid Exceptions on CacheOps

TLB Refill and TLB Invalid exceptions can occur on any operation. For Index operations, where the address (virtual address for the primary caches, physical address for the secondary cache) is used to index the cache but need not match the cache tag, unmapped addresses may be used to avoid TLB exceptions. The operation never causes TLB Modified exceptions.

Hit Operation Accesses

A Hit operation accesses the specified cache as a normal data reference, and performs the specified operation if the cache block contains valid data at the specified physical address (a hit).

The operation is undefined if a CacheOp hit occurs in both ways of the cache.

Watch Exception

There is no Watch exception for CacheOps.

Address Error Exception

During an Index CacheOp, bit 0 is not checked for an Address Error exception since this bit is used as the *Way* indicator bit, and may be non-zero. Bit 1 of an Index CacheOp can still generate an Address Error exception if it is not set to zero.

For all remaining CacheOps, the low-order two bits of the instruction must be set to zero, or else they will generate an Address Error exception.

A CacheOp is never checked for alignment Address Error exceptions, only for privilege-type Address Error exceptions.

Write Back

Write back from the primary data cache goes to the secondary cache. Write back from a secondary cache always goes to the System interface unit.

A secondary write back always writes the most recent data; the primary data cache must be interrogated, and any dirty inconsistent data written back to the secondary cache before the secondary block is written back to the system interface unit. The address to be written is specified by the cache tag and not the translated PA.

Invalidation

When a block is invalidated in the secondary cache, all subset blocks in the primary cache are also invalidated. The *StateMod* bits on invalidated block in the primary data cache are set to “001” (*Normal*) during any invalidation.

CE Bit

The R10000 processor does not support the *CE* bit. The functionality of the *CE* bit has been replaced by the Index Load Data and Index Store Data instructions.

CH Bit

The *CH* bit is supported in the R10000 processor. It is modified by a Hit Invalidate (S) or Hit WriteBack Invalidate (S) CACHE instruction. *CH* is set if there is a hit in the secondary cache, and cleared if there is a miss. The *CH* bit can also be modified by a MTC0 instruction.

Serial Operation of CACHE Instructions

All CACHE instruction variations are performed serially. From the aspect of the primary cache, this means CACHE instructions can impede the instruction stream. For this reason, load/store speculation is not allowed beyond a CACHE instruction until the CACHE instruction has graduated. All load/store accesses, including writebacks to the external agent, must be complete before the CACHE instruction can graduate, and any load/store following a CACHE instruction cannot be issued speculatively until the CACHE instruction graduates. Uncached operations and instruction fetches are not affected.

Instructions Not Supported

The processor does not support the following CACHE instructions:

- Create DirtyExclusive
- Hit WriteBack
- Fill (I)
- Hit Set Virtual variations

Op Field Encoding

Table 10-1 presents the Op field encoding for the CACHE instruction. Encodings not listed in this table are undefined.

Table 10-1 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)

10.2 Index Invalidate (I)

Index Invalidate (I) sets a block in the primary instruction cache to *Invalid*. **VA[13:6]** defines the address and **VA[0]** defines the way to be invalidated.

The invalidation takes place by writing the primary instruction cache state bit to 0 (*Invalid*). This also sets the instruction cache state parity bit to 0.

The **LRU** bit does not change.

Parity check is suppressed.

10.3 Index WriteBack Invalidate (D)

Index WriteBack Invalidate (D) sets a block in the primary data cache to *Invalid*. **VA[13:5]** defines the address and **VA[0]** defines the way to be invalidated.

The invalidation takes place by writing the following bits:

- primary data cache state bits are set to 00 (*Invalid*)
- the **SCWay** bit is set to 0
- the **StateMod** bits = 001 (*Normal*)
- the state parity is set to 0.

The **LRU** bit does not change.

If the **StateMod** of the block to be invalidated = 010_2 (*Inconsistent*), the block in the primary data cache must be written back to the secondary cache.

The address and way in the secondary cache to be written back to are read out of the primary data cache tag address and secondary way fields and all 32 bytes are written back.

Only the data field of the secondary cache is modified by this instruction since the processor follows state and data subset rules.

Since the *CE* bit is not defined in the R10000 processor, this instruction no longer has a CP0 *ECC* register mode.

10.4 Index WriteBack Invalidate (S)

The Index WriteBack Invalidate (S) instruction sets a block in the secondary cache to *Invalid* and writes back any dirty data to the System interface unit. This operation extends to any blocks in the primary data or instruction caches which are subsets of the secondary cache block.

The CACHE instruction physical address, **PA[Cachesize-2..Blocksize]**, defines the address and **PA[0]** defines the way to be invalidated.

The invalidation occurs in the following sequence:

1. The processor reads the **STag**, **PIdx**, and **State** bits from the secondary cache tag array. If **State** = 00 (*Invalid*) no further activity takes place. If there is a valid entry, then the STag is used to interrogate the primary instruction and data caches.
2. The processor reads each subset block from the primary instruction cache. If **ITag** = **STag** and **IState** = 1 (*Valid*) then the block is invalidated by writing the **IState** bit to 0 (*Invalid*) and the **IState** parity bit to 0.
3. Read each subset block from the primary data cache. If **DTag** = **STag** and **DState** is not equal to 00 (*Invalid*), then write the **DState** bits = 00 (*Invalid*), the **StateMod** bits = 001 (*Normal*), the **SCWay** bit = 0, and the **DState** parity bit = 0. If the original block is **DState** = 11_2 (*Dirty*) and **StateMod** = 010_2 (*Inconsistent*), also write this block back to the secondary cache using the **DTag** and the **SCWay** bit from the primary data tag array.
4. Set the state of the secondary cache block to 00 (*Invalid*). Since the secondary cache is designed so all tag bits must be written at once, the Tag, VA, and ECC bits are also written. The tag is written with the PA and **VA[13:12]** (virtual index) of the original CACHE instruction address. The ECC is generated.
5. If the secondary cache block's original **State** bits were 11_2 (*Dirty*), the block is written back to the system interface unit. If the block's **State** was *Shared* or *CleanExclusive* the system interface unit is notified with a Tag Invalidation request that the block has been deleted.

The MRU bit is set to point away from the block invalidated unless the line was already invalid.

10.5 Index Load Tag (I)

Index Load Tag (I) reads the primary instruction cache tag fields into the CP0 *TagLo* and *TagHi* registers. **VA[13:6]** defines the address and **VA[0]** defines the way of the tag to be read.

All parity errors caused by Index Load Tag (I) are ignored.

The following mapping defines the operation:

TagLo[0]	= Tag parity bit
TagLo[2]	= State parity bit
TagLo[3]	= LRU bit
TagLo[6]	= State bit
TagLo[31:8]	= Tag[35:12]
TagHi[3:0]	= Tag[39:36]

All other CP0 *TagLo* and *TagHi* bits are set to 0.

10.6 Index Load Tag (D)

Index Load Tag (D) reads the primary data cache tag fields into the CP0 *TagLo* and *TagHi* registers. **VA[13:5]** defines the address and **VA[0]** defines the way of the tag to be read.

All parity errors caused by Index Load Tag (D) are ignored. The following mapping defines the operation:

TagLo[0]	= Tag parity bit
TagLo[1]	= SCWay
TagLo[2]	= State parity bit
TagLo[3]	= LRU bit
TagLo[7:6]	= State bits
TagLo[31:8]	= Tag[35:12]
TagHi[3:0]	= Tag[39:36]
TagHi[31:29]	= StateMod bits

All other CP0 *TagLo* and *TagHi* bits are set to 0.

10.7 Index Load Tag (S)

Index Load Tag (S) reads the secondary cache tag fields into the CP0 *TagLo* and *TagHi* registers. The **PA[Cachesize-2..Blocksize]** defines the address and **PA[0]** defines the way to be read.

All parity and ECC errors caused by Index Load Tag (D) are ignored.

The following mapping defines the operation:

TagLo[6:0]	= Tag ECC bits
TagLo[8:7]	= Virtual index bits
TagLo[11:10]	= State bits
TagLo[31:14]	= Tag[35:18]
TagHi[3:0]	= Tag[39:36]
TagHi[31]	= MRU Bit

All other CP0 *TagLo* and *TagHi* register bits are set to 0.

10.8 Index Store Tag (I)

Index Store Tag (I) stores the CP0 *TagLo* and *TagHi* registers into the primary instruction cache tag array. **VA[13:6]** defines the address and **VA[0]** defines the way of the tag to be written.

The following mapping defines the operation:

Tag parity bit	= **TagLo[0]**
State parity bit	= **TagLo[2]**
LRU bit	= **TagLo[3]**
State bit	= **TagLo[6]**
Tag[35:12]	= **TagLo[31:8]**
Tag[39:36]	= **TagHi[3:0]**

All the Tag fields, including parity, are directly written.

Parity check is suppressed for all Index Store Tags.

10.9 Index Store Tag (D)

Index Store Tag (D) stores the CP0 *TagLo* and *TagHi* registers into the primary data cache tag array. **VA[13:5]** defines the address and **VA[0]** defines the way of the tag to be written.

The following mapping defines the operation:

Tag parity bit = **TagLo[0]**

SCWay = **TagLo[1]**

State parity bit = **TagLo[2]**

LRU bit = **TagLo[3]**

State bits = **TagLo[7:6]**

Tag[35:12] = **TagLo[31:8]**

Tag[39:36] = **TagHi[3:0]**

StateMod bits = **TagHi[31:29]**

All Tag fields, including parity, are directly written.

Parity check is suppressed for all Index Store Tags.

10.10 Index Store Tag (S)

Index Store Tag (S) stores fields from the CP0 *TagLo* and *TagHi* registers into the secondary cache tag and MRU array fields. The **PA[Cachesize-2..Blocksize]** defines the address and **PA[0]** defines the way to be read.

The following mapping defines the operation:

Tag ECC bits = **TagLo[6:0]**

Virtual index bits = **TagLo[8:7]**

State bits = **TagLo[11:10]**

Tag[35:18] = **TagLo[31:14]**

Tag[39:36] = **TagHi[3:0]**

MRU bit = **TagHi[31]**

All Tag fields, including ECC, are directly written.

Parity check is suppressed for all Index Store Tags.

10.11 Hit Invalidate (I)

Hit Invalidate (I) invalidates an entry in the instruction cache which matches the PA of the CACHE instruction. Both way tags at **VA[13:6]** are read from the instruction cache.

If the **IState** is 1 (*Valid*), and the PA of the CACHE instruction matches the Tag from the instruction cache tag array, the **IState** bit of the entry is written to 0 (*Invalid*) and the **IState** parity bit is written to 0.

The **LRU** bit does not change.

Parity error is checked.

Hit CacheOps can cause cache error exceptions if they check ECC or parity bits.

10.12 Hit Invalidate (D)

Hit Invalidate (D) invalidates an entry in the data cache which matches the PA of the CACHE instruction. Both ways tags at **VA[13:5]** are read from the data cache.

If the **DState** is not equal to 00 (*Invalid*) and the PA of the CACHE instruction matches the DTag from the data cache tag array, then the **State** bits are written to 00 (*Invalid*), the **SCWay** bit = 0, the **StateMod** bits = 001_2 (*Normal*), and the **DState** parity = 0.

The **LRU** bit is left unchanged.

Parity check is enabled.

Hit CacheOps can cause cache error exceptions if they check ECC or parity bits.

10.13 Hit Invalidate (S)

Hit Invalidate (S) invalidates all entries in the secondary, primary instruction, and primary data caches which match the PA of the CACHE instruction. The following sequence takes place:

1. The processor reads the Tags from both ways of the secondary cache at the address pointed to by the PA of the CACHE instruction. If the tag entry's STag matches the CACHE instruction PA, and the **State** of the entry is not equal to 00 (*Invalid*), then a Hit has occurred in that entry. If there is no Hit, the CACHE instruction completes.
2. The processor checks each entry in the primary caches to determine which corresponds to the CACHE instruction PA and the *PIdx* read from the secondary cache tag array. Any entry which matches is invalidated. No write back is required by Hit Invalidate (S).
3. The processor sets the tag array entry of the secondary cache block which was hit to **State** = 00 (*Invalid*), **Tag** = PA of CACHE instruction, and **PIdx** = **VA[13:12]** of CACHE instruction.
4. ECC is generated.
5. The **MRU** bit is written to point to the way opposite to that being invalidated.
6. If the processor Eliminate Request mode bit, **PrcElmReq**, is set, a processor eliminate request is sent to notify the external agent that a block in the secondary cache has been invalidated.
7. Hit Invalidate (S) sets the **CH** bit if it hits in the secondary cache.
8. Once the **CH** bit is set it stays set until cleared by a MTC0 instruction, or the next CacheOp that can change the **CH** bit.

Hit CacheOps can cause cache error exceptions if they check ECC or parity bits.

10.14 Cache Barrier

Cache Barrier does not change any cache fields. It is used when serialization of a CACHE instruction is needed without unwanted side effects. For more information, see the section titled the section titled "Serial Operation of CACHE Instructions," in this chapter.

10.15 Hit Writeback Invalidate (D)

Hit Writeback Invalidate (D) invalidates an entry in the primary data cache which matches the PA of the CACHE instruction. In addition, it writes back to the secondary cache any *DirtyExclusive* or *Inconsistent* data found in the primary data cache. Both way DTags at **VA[13:5]** are read from the data cache.

If the **DState** is not equal to 00 (*Invalid*) and PA of the CACHE instruction matches the DTag, then the **DState** bits of the entry are set to 00 (*Invalid*), the **SCWay** is set to 0, the **DState** parity is set to 0, and the **StateMod** bits are set to 001_2 (*Normal*).

The **LRU** bit is left unchanged.

If the state of the block to be invalidated was found to be **StateMod** = 010_2 (*Inconsistent*), the block in the primary data cache must be written back to the secondary cache. The address and way in the secondary cache to be written back to are read out of the primary data cache Tag Address and secondary way fields, and all 32 bytes are written back.

Only the data field of the secondary cache is modified by this instruction since the processor obeys State and data subset rules.

Since the *CE* bit is not defined in the R10000 processor, this instruction no longer has an *ECC* register mode.

Hit CacheOps can cause cache error exceptions if they check ECC or parity bits.

10.16 Hit WriteBack Invalidate (S)

Hit Writeback Invalidate (S) checks for a block which matches the CACHE instruction PA in the secondary cache, invalidates it, and writes back any dirty data to the System interface unit. This operation extends to any blocks in the primary data or instruction caches which are subsets of the secondary cache block. The operation takes place in the following sequence:

1. The processor reads the **STag**, **PIdx**, and **State** bits from both ways of the secondary tag array.
2. If the PA of the CACHE instruction matches the **STag**, and the **State** does not equal 00 (*Invalid*), a hit has occurred. If there is a hit, the **STag** is used to interrogate the primary caches. If there is not a hit, the instruction ends.
3. The processor reads each subset block from the primary instruction cache. If there is a match then invalidate the block by writing the **IState** bit to 0 (*Invalid*) and the **IState** parity bit to 0.
4. Read each subset block from the primary data cache. If there is a match then write the **DState** bits = 00 (*Invalid*), the **StateMod** bits = 001 (*Normal*), the **SCWay** bit = 0, and the **DState** parity bit = 0. If the original State of any subset block is **StateMod** = 010_2 (*Inconsistent*), also write it back to the secondary cache using the DTag and the secondary way bit from the primary data tag array.
5. Write the **State** of the secondary cache block = 00 (*Invalid*). Since the secondary cache is designed so all tag bits must be written at once, the STag, PIdx, and ECC bits are also written. The STag is written with whatever the PA and **VA[13:12]** of the original CACHE instruction were. The Tag ECC is generated.
6. If the secondary block's original **State** bits were 11_2 (*Dirty*) then the block is written back to the system interface unit. If the block's State was *Shared* or *CleanExclusive* the system interface unit is simply notified that the block has been deleted with a "Tag Invalidation" request.
7. The **MRU** bit is set to point away from the block invalidated.

Hit WriteBack Invalidate (S) set the *CH* bit if it hits in the secondary cache. Once the *CH* bit is set it stays set until cleared by a MTC0 Instruction.

Hit CacheOps can cause cache error exceptions if they check ECC or parity bits.

10.17 Index Load Data (I)

Index Load Data (I) loads a single instruction from the primary instruction cache into the CP0 *TagHi*, *TagLo*, and *ECC* registers. A predecoded instruction in R10000 is 36 bits of data and one bit of parity. The address of the target instruction is **VA[13:2]** of the CACHE instruction. The way of the target instruction is **VA[0]** of the CACHE instruction. The instruction itself is loaded into CP0 *TagHi[3:0]* and *TagLo[31:0]*. The parity bit is loaded into CP0 *ECC[0]*. The tag field is not read.

Parity checking is suppressed during operation of Index Load Data (I).

10.18 Index Load Data (D)

Index Load Data (D) loads a singleword of data and the corresponding four bits of byte parity into CP0 *TagLo* and *ECC*. The address of the target singleword is **VA[13:2]** of the CACHE instruction. The way of the target singleword is **VA[0]** of the CACHE instruction. The singleword of data will be loaded into the CP0 *TagLo* register. The byte parity will be loaded into CP0 *ECC*[3:0] register. The tag field is not read.

Parity checking is suppressed during operation of Index Load Data (D).

10.19 Index Load Data (S)

Index Load Data (S) loads a doubleword of data and all 10 check bits into the CP0 *TagHi*, *TagLo*, and *ECC* registers. The address of the target doublewords comes from the PA of the CACHE instruction. The way comes from **PA[0]** of the CACHE instruction. The high word will be loaded into CP0 *TagHi* and the low word of data will be loaded into CP0 *TagLo*. The check bits will be loaded into CP0 *ECC[9:0]*. The MRU field is unmodified.

ECC correction and checking is suppressed during Index Load Data (S).

10.20 Index Store Data (I)

Index Store Data (I) stores a single instruction into the primary instruction cache. The address where this instruction will be written comes from **VA[13:2]** of the CACHE instruction. The way where the data will be written comes from **VA[0]** of the CACHE instruction. The instruction itself comes from CP0 *TagHi[3:0]* and *TagLo[31:0]*. The parity bit is also stored. This comes from CP0 *ECC[0]*. The data to be stored bypasses the predecode and is written directly into the instruction cache. The tag field is unmodified.

10.21 Index Store Data (D)

Index Store Data (D) stores a word of data and its byte parity into the data cache from the CP0 *TagLo* and *ECC* registers. The address where this word will be written is defined by **VA[13:2]** of the CACHE instruction. The way is defined by **VA[0]**. The data word comes from CP0 *TagLo*. The parity bits come from CP0 *ECC[3:0]*. The data cache tag array including the LRU bit is left unchanged.

10.22 Index Store Data (S)

Index Store Data (S) stores a quadword of data and 10 check bits into the secondary cache data array. It stores a doubleword of data from CP0 *TagHi* and *TagLo* and pads the remaining doubleword with zeroes. This allows the ECC and parity, which are based on the quadword, to be valid for the doubleword of data stored. The address of the quadword stored is defined by the PA of the CACHE instruction, and the way is defined by **PA[0]**. The data stored in the non-padded doubleword comes from CP0 *TagHi* and *TagLo*. The check bits are stored from *ECC[9:0]*. The tag array including the MRU bit is left unchanged.