BGEZAL — Branch On Greater Than Or Equal To Zero And Link — BGEZAL

31 — 26	25 — 21	20 — 16	15 — 0
REGIMM 0 0 0 0 0 1	rs	BGEZAL 1 0 0 0 1	offset
6	5	5	16

Format:

BGEZAL rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit cleared, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *31*, because such an instruction is not restartable. An attempt to execute this instruction is not trapped, however.

Operation:

32	T:	$\text{target} \leftarrow (\text{offset}_{15})^{14} \,\|\|\, \text{offset} \,\|\|\, 0^2$
		$\text{condition} \leftarrow (\text{GPR}[\text{rs}]_{31} = 0)$
		$\text{GPR}[31] \leftarrow \text{PC} + 8$
	T+1:	if condition then
		$\text{PC} \leftarrow \text{PC} + \text{target}$
		endif
64	T:	$\text{target} \leftarrow (\text{offset}_{15})^{46} \,\|\|\, \text{offset} \,\|\|\, 0^2$
		$\text{condition} \leftarrow (\text{GPR}[\text{rs}]_{63} = 0)$
		$\text{GPR}[31] \leftarrow \text{PC} + 8$
	T+1:	if condition then
		$\text{PC} \leftarrow \text{PC} + \text{target}$
		endif

Exceptions:

None

BGEZALL Branch On Greater Than Or Equal To Zero And Link Likely BGEZALL

31 … 26	25 … 21	20 … 16	15 … 0
REGIMM 0 0 0 0 0 1	rs	BGEZALL 1 0 0 1 1	offset
6	5	5	16

Format:

BGEZALL rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset,* shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit cleared, then the program branches to the target address, with a delay of one instruction. General register *rs* may not be general register *31*, because such an instruction is not restartable. An attempt to execute this instruction is not trapped, however. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Operation:

32 T: target $\leftarrow$ $(\text{offset}_{15})^{14}$ || offset || 0^2
condition $\leftarrow$ $(\text{GPR[rs]}_{31} = 0)$
GPR[31] $\leftarrow$ PC + 8
T+1: if condition then
PC $\leftarrow$ PC + target
else
NullifyCurrentInstruction
endif

64 T: target $\leftarrow$ $(\text{offset}_{15})^{46}$ || offset || 0^2
condition $\leftarrow$ $(\text{GPR[rs]}_{63} = 0)$
GPR[31] $\leftarrow$ PC + 8
T+1: if condition then
PC $\leftarrow$ PC + target
else
NullifyCurrentInstruction
endif

Exceptions:

None

BGEZL — Branch On Greater Than Or Equal To Zero Likely — BGEZL

31–26	25–21	20–16	15–0
REGIMM 0 0 0 0 0 1	rs	BGEZL 0 0 0 1 1	offset
6	5	5	16

Format:

BGEZL rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* have the sign bit cleared, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Operation:

32 T: target $\leftarrow$ $(\text{offset}_{15})^{14}$ || offset || 0^2
condition $\leftarrow$ $(\text{GPR[rs]}_{31} = 0)$
T+1: if condition then
PC $\leftarrow$ PC + target
else
NullifyCurrentInstruction
endif

64 T: target $\leftarrow$ $(\text{offset}_{15})^{46}$ || offset || 0^2
condition $\leftarrow$ $(\text{GPR[rs]}_{63} = 0)$
T+1: if condition then
PC $\leftarrow$ PC + target
else
NullifyCurrentInstruction
endif

Exceptions:

None

BGTZ Branch On Greater Than Zero BGTZ

31–26	25–21	20–16	15–0
BGTZ 0 0 0 1 1 1	rs	0 0 0 0 0 0	offset
6	5	5	16

Format:

BGTZ rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit cleared and are not equal to zero, then the program branches to the target address, with a delay of one instruction.

Operation:

```
32   T:     target ← (offset15)^14 || offset || 0^2
            condition ← (GPR[rs]31 = 0) and (GPR[rs] ≠ 0^32)
     T+1:   if condition then
                PC ← PC + target
            endif
64   T:     target ← (offset15)^46 || offset || 0^2
            condition ← (GPR[rs]63 = 0) and (GPR[rs] ≠ 0^64)
     T+1:   if condition then
                PC ← PC + target
            endif
```

Exceptions:

None

BGTZL — Branch On Greater Than Zero Likely — BGTZL

31–26	25–21	20–16	15–0
BGTZL 0 1 0 1 1 1	rs	0 0 0 0 0 0	offset
6	5	5	16

Format:

BGTZL rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit cleared and are not equal to zero, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Operation:

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR[rs]31 = 0) and (GPR[rs] ≠ 0^32)
      T+1:  if condition then
                 PC ← PC + target
            else
                 NullifyCurrentInstruction
            endif
64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR[rs]63 = 0) and (GPR[rs] ≠ 0^64)
      T+1:  if condition then
                 PC ← PC + target
            else
                 NullifyCurrentInstruction
            endif
```

Exceptions:

None

BLEZ — Branch on Less Than Or Equal To Zero — BLEZ

31–26	25–21	20–16	15–0
BLEZ 0 0 0 1 1 0	rs	0 0 0 0 0 0	offset
6	5	5	16

Format:

BLEZ rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* are compared to zero. If the contents of general register *rs* have the sign bit set, or are equal to zero, then the program branches to the target address, with a delay of one instruction.

Operation:

32	T:	target $\leftarrow$ $(\text{offset}_{15})^{14}$ || offset || 0^2
		condition $\leftarrow$ $(\text{GPR[rs]}_{31} = 1)$ or $(\text{GPR[rs]} = 0^{32})$
	T+1:	if condition then
		PC $\leftarrow$ PC + target
		endif
64	T:	target $\leftarrow$ $(\text{offset}_{15})^{46}$ || offset || 0^2
		condition $\leftarrow$ $(\text{GPR[rs]}_{63} = 1)$ or $(\text{GPR[rs]} = 0^{64})$
	T+1:	if condition then
		PC $\leftarrow$ PC + target
		endif

Exceptions:

None

BLEZL — Branch on Less Than Or Equal To Zero Likely — BLEZL

31–26	25–21	20–16	15–0
BLEZL 0 1 0 1 1 0	rs	0 0 0 0 0 0	offset
6	5	5	16

Format:

BLEZL rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* is compared to zero. If the contents of general register *rs* have the sign bit set, or are equal to zero, then the program branches to the target address, with a delay of one instruction.

If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Operation:

```
32   T:    target ← (offset15)^14 || offset || 0^2
           condition ← (GPR[rs]31 = 1) or  (GPR[rs] = 0^32)
     T+1:  if condition then
                PC ← PC + target
           else
                NullifyCurrentInstruction
           endif
64   T:    target ← (offset15)^46 || offset || 0^2
           condition ← (GPR[rs]63 = 1) or  (GPR[rs] = 0^64)
     T+1:  if condition then
                PC ← PC + target
           else
                NullifyCurrentInstruction
           endif
```

Exceptions:

None

BLTZ — Branch On Less Than Zero — BLTZ

31 … 26	25 … 21	20 … 16	15 … 0
REGIMM 0 0 0 0 0 1	rs	BLTZ 0 0 0 0 0	offset
6	5	5	16

Format:

BLTZ rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction.

Operation:

32 T: target $\leftarrow$ $(\text{offset}_{15})^{14}$ || offset || 0^2
condition $\leftarrow$ $(\text{GPR[rs]}_{31} = 1)$
T+1: if condition then
PC $\leftarrow$ PC + target
endif

64 T: target $\leftarrow$ $(\text{offset}_{15})^{46}$ || offset || 0^2
condition $\leftarrow$ $(\text{GPR[rs]}_{63} = 1)$
T+1: if condition then
PC $\leftarrow$ PC + target
endif

Exceptions:

None

BLTZAL — Branch On Less Than Zero And Link — BLTZAL

31–26	25–21	20–16	15–0
REGIMM 000001	rs	BLTZAL 10000	offset
6	5	5	16

Format:

BLTZAL rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *31*, because such an instruction is not restartable. An attempt to execute this instruction with register *31* specified as *rs* is not trapped, however.

Operation:

```
32    T:    target ← (offset_15)^14 || offset || 0^2
            condition ← (GPR[rs]_31 = 1)
            GPR[31] ← PC + 8
      T+1:  if condition then
                 PC ← PC + target
            endif
64    T:    target ← (offset_15)^46 || offset || 0^2
            condition ← (GPR[rs]_63 = 1)
            GPR[31] ← PC + 8
      T+1:  if condition then
                 PC ← PC + target
            endif
```

Exceptions:

None

BLTZALL Branch On Less Than Zero And Link Likely BLTZALL

31–26	25–21	20–16	15–0
REGIMM 0 0 0 0 0 1	rs	BLTZALL 1 0 0 1 0	offset
6	5	5	16

Format:

BLTZALL rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. Unconditionally, the address of the instruction after the delay slot is placed in the link register, *r31*. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction.

General register *rs* may not be general register *31*, because such an instruction is not restartable. An attempt to execute this instruction with register *31* specified as *rs* is not trapped, however. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Operation:

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR[rs]31 = 1)
            GPR[31] ← PC + 8
      T+1:  if condition then
                 PC ← PC + target
            else
                 NullifyCurrentInstruction
            endif
64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR[rs]63 = 1)
            GPR[31] ← PC + 8
      T+1:  if condition then
                 PC ← PC + target
            else
                 NullifyCurrentInstruction
            endif
```

Exceptions:

None

BLTZL Branch On Less Than Zero Likely BLTZL

31–26	25–21	20–16	15–0
REGIMM 0 0 0 0 0 1	rs	BLTZL 0 0 0 1 0	offset
6	5	5	16

Format:

BLTZ rs, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. If the contents of general register *rs* have the sign bit set, then the program branches to the target address, with a delay of one instruction. If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Operation:

```
32    T:    target ← (offset15)^14 || offset || 0^2
            condition ← (GPR[rs]31 = 1)
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
64    T:    target ← (offset15)^46 || offset || 0^2
            condition ← (GPR[rs]63 = 1)
      T+1:  if condition then
                  PC ← PC + target
            else
                  NullifyCurrentInstruction
            endif
```

Exceptions:

None

BNE — Branch On Not Equal — BNE

31–26	25–21	20–16	15–0
BNE 0 0 0 1 0 1	rs	rt	offset
6	5	5	16

Format:

BNE rs, rt, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

Operation:

```
32    T:     target ← (offset15)^14 || offset || 0^2
             condition ← (GPR[rs] ≠ GPR[rt])
      T+1:   if condition then
                  PC ← PC + target
             endif
64    T:     target ← (offset15)^46 || offset || 0^2
             condition ← (GPR[rs] ≠ GPR[rt])
      T+1:   if condition then
                  PC ← PC + target
             endif
```

Exceptions:

None

BNEL Branch On Not Equal Likely BNEL

31 26	25 21	20 16	15 0
BNEL 0 1 0 1 0 1	rs	rt	offset
6	5	5	16

Format:

BNEL rs, rt, offset

Description:

A branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit *offset*, shifted left two bits and sign-extended. The contents of general register *rs* and the contents of general register *rt* are compared. If the two registers are not equal, then the program branches to the target address, with a delay of one instruction.

If the conditional branch is not taken, the instruction in the branch delay slot is nullified.

Operation:

```
32   T:    target ← (offset15)^14 || offset || 0^2
           condition ← (GPR[rs] ≠ GPR[rt])
     T+1:  if condition then
                PC ← PC + target
           else
                NullifyCurrentInstruction
           endif
64   T:    target ← (offset15)^46 || offset || 0^2
           condition ← (GPR[rs] ≠ GPR[rt])
     T+1:  if condition then
                PC ← PC + target
           else
                NullifyCurrentInstruction
           endif
```

Exceptions:

None

BREAK Breakpoint BREAK

31 26	25 6	5 0
SPECIAL 0 0 0 0 0 0	code	BREAK 0 0 1 1 0 1
6	20	6

Format:

BREAK

Description:

A breakpoint trap occurs, immediately and unconditionally transferring control to the exception handler.

The code field is available for use as software parameters, but is retrieved by the exception handler only by loading the contents of the memory word containing the instruction.

Operation:

32, 64	T:	BreakpointException

Exceptions:

Breakpoint exception

CACHE — Cache — CACHE

31 … 26	25 … 21	20 … 16	15 … 0
CACHE 1 0 1 1 1 1	base	op	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 virtual address. The virtual address is translated to a physical address using the TLB, and the 5-bit sub-opcode specifies a cache operation for that address.

If CP0 is not usable (User or Supervisor mode) the CP0 enable bit in the *Status* register is clear, and a coprocessor unusable exception is taken. The operation of this instruction on any operation/cache combination not listed below, or on a secondary cache when none is present, is undefined. The operation of this instruction on uncached addresses is also undefined.

The Index operation uses part of the virtual address to specify a cache block.

For a primary cache of $2^{CACHEBITS}$ bytes with $2^{LINEBITS}$ bytes per tag, $vAddr_{CACHEBITS\ ...\ LINEBITS}$ specifies the block.

For a secondary cache of $2^{CACHEBITS}$ bytes with $2^{LINEBITS}$ bytes per tag, $pAddr_{CACHEBITS\ ...\ LINEBITS}$ specifies the block.

Index Load Tag also uses $vAddr_{LINEBITS...\ 3}$ to select the doubleword for reading ECC or parity. When the *CE* bit of the *Status* register is set, Hit WriteBack, Hit WriteBack Invalidate, Index WriteBack Invalidate, and Fill also use $vAddr_{LINEBITS\ ...\ 3}$ to select the doubleword that has its ECC or parity modified. This operation is performed unconditionally.

The Hit operation accesses the specified cache as normal data references, and performs the specified operation if the cache block contains valid data with the specified physical address (a hit). If the cache block is invalid or contains a different address (a miss), no operation is performed.

CACHE — Cache (continued) — CACHE

Write back from a primary cache goes to the secondary cache (if there is one), otherwise to memory. Write back from a secondary cache always goes to memory. A secondary write back always writes the most recent data; the data comes from the primary data cache, if present, and modified (the *W* bit is set). Otherwise the data comes from the specified secondary cache. The address to be written is specified by the cache tag and not the translated physical address.

TLB Refill and TLB Invalid exceptions can occur on any operation. For Index operations (where the physical address is used to index the cache but need not match the cache tag) unmapped addresses may be used to avoid TLB exceptions. This operation never causes TLB Modified or Virtual Coherency exceptions.

Bits 17...16 of the instruction specify the cache as follows:

Code	Name	Cache
0	I	primary instruction
1	D	primary data
2	SI	secondary instruction
3	SD	secondary data (or combined instruction/data)

CACHE Cache (continued) CACHE

Bits 20...18 (this value is listed under the **Code** column) of the instruction specify the operation as follows:

Code	Caches	Name	Operation
0	I, SI	Index Invalidate	Set the cache state of the cache block to Invalid.
0	D	Index Writeback Invalidate	Examine the cache state and Writeback bit (*W* bit) of the primary data cache block at the index specified by the virtual address. If the state is not Invalid and the *W* bit is set, write the block back to the secondary cache (if present) or to memory (if no secondary cache). The address to write is taken from the primary cache tag. When a secondary cache is present, and the *CE* bit of the *Status* register is set, the contents of the *ECC* register is XOR'd into the computed check bits during the write to the secondary cache for the addressed doubleword. Set the cache state of primary cache block to Invalid. The *W* bit is unchanged (and irrelevant because the state is Invalid).
O	SD	Index Writeback Invalidate	Examine the cache state of the secondary data cache block at the index specified by the physical address. If the block is dirty (the state is Dirty Exclusive or Dirty Shared), write the data back to memory. Like all secondary writebacks, the operation writes any modified data for the addresses from the primary data cache. The address to write is taken from the secondary cache tag. The *PIdx* field of the secondary tag is used to determine the locations in the primaries to check for matching primary blocks. In all cases, set the state of the secondary cache block and all matching primary subblocks to Invalid. No Invalidate is sent on the R4000's system interface.
1	All	Index Load Tag	Read the tag for the cache block at the specified index and place it iinto the *TagLo* and *TagHi* CP0 registers, ignoring any ECC or parity errors. Also load the data ECC or parity bits into the ECC register.
2	All	Index Store Tag	Write the tag for the cache block at the specified index from the *TagLo* and *TagHi* CP0 registers. The processor uses computed parity for the primary caches and the *TagLo* register in the case of the secondary cache.

CACHE — Cache (continued) — CACHE

Code	Caches	Name	Operation
3	SD	Create Dirty Exclusive	This operation is used to avoid loading data needlessly from memory when writing new contents into an entire cache block. If the cache block is valid but does not contain the specified address (a valid miss) the secondary block is vacated. The data is written back to memory if dirty and all matching blocks in both primary caches are invalidated. As usual during a secondary writeback, if the primary data cache contains modified data (matching blocks with *W* bit set) that modified data is written to memory. If the cache block is valid and contains the specified physical address (a hit), the operation cleans up the primary caches to avoid virtual aliases: all blocks in both primary caches that match the secondary line are invalidated without writeback. Note that the search for matching primary blocks uses the virtual index of the *PIdx* field of the secondary cache tag (the virtual index when the location was last used) and not the virtual index of the virtual address used in the operation (the virtual index where the location will now be used). If the secondary tag and address do not match (miss), or the tag and address do match (hit) and the block is in a shared state, an invalidate for the specified address is sent over the System interface. In all cases, the cache block tag must be set to the specified physical address, the cache state must be set to Dirty Exclusive, and the virtual index field set from the virtual address. The *CH* bit in the *Status* register is set or cleared to indicate a hit or miss.
3	D	Create Dirty Exclusive	This operation is used to avoid loading data needlessly from secondary cache or memory when writing new contents into an entire cache block. If the cache block does not contain the specified address, and the block is dirty, write it back to the secondary cache (if present) or otherwise to memory. In all cases, set the cache block tag to the specified physical address, set the cache state to Dirty Exclusive.
4	I,D	Hit Invalidate	If the cache block contains the specified address, mark the cache block invalid.
4	SI, SD	Hit Invalidate	If the cache block contains the specified address, mark the cache block invalid and also invalidate all matching blocks, if present, in the primary caches (the *PIdx* field of the secondary tag is used to determine the locations in the primaries to search). The *CH* bit in the *Status* register is set or cleared to indicate a hit or miss.
5	D	Hit Writeback Invalidate	If the cache block contains the specified address, write the data back if it is dirty, and mark the cache block invalid. When a secondary cache is present, and the *CE* bit of the *Status* register is set, the contents of the *ECC* register is XOR'd into the computed check bits during the write to the secondary cache for the addressed doubleword.

CACHE Cache (continued) CACHE

Code	Caches	Name	Operation
5	SD	Hit Writeback Invalidate	If the cache block contains the specified address, write back the data (if dirty), and mark the secondary cache block and all matching blocks in both primary caches invalid. As usual with secondary writebacks, modified data in the primary data cache (matching block with the *W* bit set) is used during the writeback. The *PIdx* field of the secondary tag is used to determine the locations in the primaries to check for matching primary blocks. The *CH* bit in the *Status* register is set or cleared to indicate a hit or miss.
5	I	Fill	Fill the primary instruction cache block from secondary cache or memory. If the *CE* bit of the Status register is set, the content of the *ECC* register is used instead of the computed parity bits for addressed doubleword when written to the instruction cache. For the R4000PC, the cache is filled from memory. For the R4000SC and R4000MC, the cache is filled from the secondary cache whether or not the secondary cache block is valid or contains the specified address.
6	D	Hit Writeback	If the cache block contains the specified address, and the *W* bit is set, write back the data. The *W* bit is not cleared; a subsequent miss to the block will write it back again. This second writeback is redundant, but not incorrect. When a secondary cache is present, and the *CE* bit of the *Status* register is set, the content of the *ECC* register is XOR'd into the computed check bits during the write to the secondary cache for the addressed doubleword. Note: The *W* bit is not cleared during this operation due to an artifact of the implementation; the *W* bit is implemented as part of the data side of the cache array so that it can be written during a data write.
6	SD	Hit Writeback	If the cache block contains the specified address, and the cache state is Dirty Exclusive or Dirty Shared, data is written back to memory. The cache state is unchanged; a subsequent miss to the block causes it to be written back again. This second writeback is redundant, but not incorrect. The *CH* bit in the *Status* register is set or cleared to indicate a hit or miss. The writeback looks in the primary data cache for modified data, but does not invalidate or clear the Writeback bit in the primary data cache. Note: The state of the secondary block is not changed to clean during this operation because the *W* bit of matching sub-blocks cannot be cleared to put the primary block in a clean state.
6	I	Hit Writeback	If the cache block contains the specified address, data is written back unconditionally. When a secondary cache is present, and the *CE* bit of the *Status* register is set, the contents of the *ECC* register is XOR'd into the computed check bits during the write to the secondary cache for the addressed doubleword.

CACHE — Cache (continued) — CACHE

Code	Caches	Name	Operation
7	SI,SD	Hit Set Virtual	This operation is used to change the virtual index of secondary cache contents, avoiding unnecessary memory operations. If the cache block contains the specified address, invalidate matching blocks in the primary caches at the index formed by concatenating *PIdx* in the secondary cache tag (not the virtual address of the operation) and $vAddr_{11..4}$, and then set the virtual index field of the secondary cache tag from the specified virtual address. Modified data in the primary data cache is not preserved by the operation and should be explicitly written back before this operation. The *CH* bit in the *Status* register is set or cleared to indicate a hit or miss.

Operation:

32, 64	T:	$vAddr \leftarrow ((offset_{15})^{48} \mid\mid offset_{15...0}) + GPR[base]$
		$(pAddr, uncached) \leftarrow AddressTranslation (vAddr, DATA)$
		CacheOp (op, vAddr, pAddr)

Exceptions:

Coprocessor unusable exception

CFCz — Move Control From Coprocessor — CFCz

31 … 26	25 … 21	20 … 16	15 … 11	10 … 0
COPz 0 1 0 0 x x*	CF 0 0 0 1 0	rt	rd	0 0 0 0 0 0
6	5	5	5	11

Format:

CFCz rt, rd

Description:

The contents of coprocessor control register *rd* of coprocessor unit *z* are loaded into general register *rt*.

This instruction is not valid for CP0.

Operation:

32	T:	data ← CCR[z,rd]
	T+1:	GPR[rt] ← data
64	T:	data ← $(CCR[z,rd]_{31})^{32}$ || CCR[z,rd]
	T+1:	GPR[rt] ← data

Exceptions:

Coprocessor unusable exception

***Opcode Bit Encoding:**

CFCz

	31	30	29	28	27	26	25	24	23	22	21 … 0
CFC1	0	1	0	0	0	1	0	0	0	1	0
CFC2	0	1	0	0	1	0	0	0	0	1	0

Opcode (bits 31–26, first four) | Coprocessor Unit Number (bits 27–26) | Coprocessor Suboperation (bits 25–21)

COPz Coprocessor Operation COPz

31–26	25	24–0
COPz 0 1 0 0 x x*	CO 1	cofun
6	1	25

Format:

COPz cofun

Description:

A coprocessor operation is performed. The operation may specify and reference internal coprocessor registers, and may change the state of the coprocessor condition line, but does not modify state within the processor or the cache/memory system. Details of coprocessor operations are contained in Appendix B.

Operation:

32, 64	T:	CoprocessorOperation (z, cofun)

Exceptions:

Coprocessor unusable exception
Coprocessor interrupt or Floating-Point Exception (R4000 CP1 only)

***Opcode Bit Encoding:**

COPz

Bit #	31	30	29	28	27	26	25	0
C0P0	0	1	0	0	0	0	1	

Bit #	31	30	29	28	27	26	25	0
C0P1	0	1	0	0	0	1	1	

Bit #	31	30	29	28	27	26	25	0
C0P2	0	1	0	0	1	0	1	

Opcode (bits 31–28)
Coprocessor Unit Number (bits 27–26)
CO sub-opcode (see end of Appendix A) (bit 25)

CTCz Move Control to Coprocessor CTCz

31 26	25 21	20 16	15 11	10 0
COPz 0 1 0 0 x x *	CT 0 0 1 1 0	rt	rd	0 0 0 0 0 0 0 0 0 0 0 0
6	5	5	5	11

Format:

CTCz rt, rd

Description:

The contents of general register *rt* are loaded into control register *rd* of coprocessor unit *z*.

This instruction is not valid for CP0.

Operation:

32,64	T:	data ← GPR[rt]
	T + 1:	CCR[z,rd] ← data

Exceptions:

Coprocessor unusable

*See "CPU Instruction Opcode Bit Encoding" at the end of Appendix A.

DADD Doubleword Add DADD

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL 0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0	DADD 1 0 1 1 0 0
6	5	5	5	5	6

Format:

DADD rd, rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*.

An overflow exception occurs if the carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	GPR[rd] ←GPR[rs] + GPR[rt]

Exceptions:

Integer overflow exception
Reserved instruction exception (R4000 in 32-bit mode)

DADDI Doubleword Add Immediate DADDI

31–26	25–21	20–16	15–0
DADDI 0 1 1 0 0 0	rs	rt	immediate
6	5	5	16

Format:

DADDI rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.*

An overflow exception occurs if carries out of bits 62 and 63 differ (2's complement overflow). The destination register *rt* is not modified when an integer overflow exception occurs.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$GPR[rt] \leftarrow GPR[rs] + (immediate_{15})^{48} \; || \; immediate_{15...0}$

Exceptions:

Integer overflow exception
Reserved instruction exception (R4000 in 32-bit mode)

DADDIU — Doubleword Add Immediate Unsigned — DADDIU

31 … 26	25 … 21	20 … 16	15 … 0
DADDIU 0 1 1 0 0 1	rs	rt	immediate
6	5	5	16

Format:

DADDIU rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and added to the contents of general register *rs* to form the result. The result is placed into general register *rt.* No integer overflow exception occurs under any circumstances.

The only difference between this instruction and the DADDI instruction is that DADDIU never causes an overflow exception.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	$\mathrm{GPR}\,[rt] \leftarrow \mathrm{GPR}[rs] + (\mathrm{immediate}_{15})^{48} \mid\mid \mathrm{immediate}_{15...0}$

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DADDU Doubleword Add Unsigned DADDU

31 — 26	25 — 21	20 — 16	15 — 11	10 — 6	5 — 0
SPECIAL 0 0 0 0 0 0	rs	rt	rd	0 0 0 0 0 0	DADDU 1 0 1 1 0 1
6	5	5	5	5	6

Format:

DADDU rd, rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are added to form the result. The result is placed into general register *rd*.

No overflow exception occurs under any circumstances.

The only difference between this instruction and the DADD instruction is that DADDU never causes an overflow exception.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

64	T:	GPR[rd] ← GPR[rs] + GPR[rt]

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)

DDIV Doubleword Divide DDIV

31–26	25–21	20–16	15–6	5–0
SPECIAL 0 0 0 0 0 0	rs	rt	0 0 0 0 0 0 0 0 0 0 0	DDIV 0 1 1 1 1 0
6	5	5	10	6

Format:

DDIV rs, rt

Description:

The contents of general register *rs* are divided by the contents of general register *rt*, treating both operands as 2's complement values. No overflow exception occurs under any circumstances, and the result of this operation is undefined when the divisor is zero.

This instruction is typically followed by additional instructions to check for a zero divisor and for overflow.

When the operation completes, the quotient word of the double result is loaded into special register *LO*, and the remainder word of the double result is loaded into special register *HI*.

If either of the two preceding instructions is MFHI or MFLO, the results of those instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by two or more instructions.

This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

```
64   T–2:   LO   ← undefined
            HI   ← undefined
     T–1:   LO   ← undefined
            HI   ← undefined
     T:     LO   ← GPR[rs] div GPR[rt]
            HI   ← GPR[rs] mod GPR[rt]
```

Exceptions:

Reserved instruction exception (R4000 in 32-bit mode)