SRA Shift Right Arithmetic SRA

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 000000	0 00000	rt	rd	sa	SRA 000011
6	5	5	5	5	6

Format:

SRA rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, sign-extending the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

Operation:

32	T:	$GPR[rd] \leftarrow (GPR[rt]_{31})^{sa} \,\|\, GPR[rt]_{31...sa}$
64	T:	$s \leftarrow 0 \,\|\, sa$
		$temp \leftarrow (GPR[rt]_{31})^{s} \,\|\, GPR[rt]_{31...s}$
		$GPR[rd] \leftarrow (temp_{31})^{32} \,\|\, temp$

Exceptions:

None

SRAV — Shift Right Arithmetic Variable — SRAV

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	SRAV 0 0 0 1 1 1
6	5	5	5	5	6

Format:

SRAV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of general register *rs*, sign-extending the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

Operation:

32	T:	$s \leftarrow GPR[rs]_{4...0}$
		$GPR[rd] \leftarrow (GPR[rt]_{31})^{s} \parallel GPR[rt]_{31...s}$
64	T:	$s \leftarrow GPR[rs]_{4...0}$
		$temp \leftarrow (GPR[rt]_{31})^{s} \parallel GPR[rt]_{31...s}$
		$GPR[rd] \leftarrow (temp_{31})^{32} \parallel temp$

Exceptions:

None

SRL Shift Right Logical SRL

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 000000	0 00000	rt	rd	sa	SRL 000010
6	5	5	5	5	6

Format:

SRL rd, rt, sa

Description:

The contents of general register *rt* are shifted right by *sa* bits, inserting zeros into the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

Operation:

32	T:	$GPR[rd] \leftarrow 0^{sa} \;\|\|\; GPR[rt]_{31...sa}$
64	T:	$s \leftarrow 0 \;\|\|\; sa$ $temp \leftarrow 0^{s} \;\|\|\; GPR[rt]_{31...s}$ $GPR[rd] \leftarrow (temp_{31})^{32} \;\|\|\; temp$

Exceptions:

None

SRLV Shift Right Logical Variable SRLV

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	SRLV 0 0 0 1 1 0
6	5	5	5	5	6

Format:

SRLV rd, rt, rs

Description:

The contents of general register *rt* are shifted right by the number of bits specified by the low-order five bits of general register *rs,* inserting zeros into the high-order bits.

The result is placed in register *rd*.

In 64-bit mode, the operand must be a valid sign-extended, 32-bit value.

Operation:

32	T:	$s \leftarrow GPR[rs]_{4...0}$ $GPR[rd] \leftarrow 0^s \,\|\|\, GPR[rt]_{31...s}$
64	T:	$s \leftarrow GPR[rs]_{4...0}$ $temp \leftarrow 0^s \,\|\|\, GPR[rt]_{31...s}$ $GPR[rd] \leftarrow (temp_{31})^{32} \,\|\|\, temp$

Exceptions:

None

SUB Subtract SUB

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	SUB 1 0 0 0 1 0
6	5	5	5	5	6

Format:

SUB rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result. The result is placed into general register *rd*. In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the SUBU instruction is that SUBU never traps on overflow.

An integer overflow exception takes place if the carries out of bits 30 and 31 differ (2's complement overflow). The destination register *rd* is not modified when an integer overflow exception occurs.

Operation:

32	T:	$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$
64	T:	$temp \leftarrow GPR[rs] - GPR[rt]$ $GPR[rd] \leftarrow (temp_{31})^{32} \mid\mid temp_{31...0}$

Exceptions:

Integer overflow exception

SUBU Subtract Unsigned SUBU

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	SUBU 1 0 0 0 1 1
6	5	5	5	5	6

Format:

SUBU rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of general register *rs* to form a result.

The result is placed into general register *rd*.

In 64-bit mode, the operands must be valid sign-extended, 32-bit values.

The only difference between this instruction and the SUB instruction is that SUBU never traps on overflow. No integer overflow exception occurs under any circumstances.

Operation:

32	T:	$GPR[rd] \leftarrow GPR[rs] - GPR[rt]$
64	T:	$temp \leftarrow GPR[rs] - GPR[rt]$ $GPR[rd] \leftarrow (temp_{31})^{32} \mid\mid temp_{31...0}$

Exceptions:

None

SW Store Word SW

31 ... 26	25 ... 21	20 ... 16	15 ... 0
SW 1 0 1 0 1 1	base	rt	offset
6	5	5	16

Format:

SW rt, 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 contents of general register *rt* are stored at the memory location specified by the effective address.

If either of the two least-significant bits of the effective address are non-zero, an address error exception occurs.

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ (ReverseEndian \,||\, 0^2)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,||\, 0^2)$
$data \leftarrow GPR[rt]_{63-8*byte} \,||\, 0^{8*byte}$
StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ (ReverseEndian \,||\, 0^2)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,||\, 0^2)$
$data \leftarrow GPR[rt]_{63-8*byte} \,||\, 0^{8*byte}$
StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)

Exceptions:

TLB refill exception
TLB modification exception
Address error exception
TLB invalid exception
Bus error exception

SWCz Store Word From Coprocessor SWCz

31 … 26	25 … 21	20 … 16	15 … 0
SWCz 1 1 1 0 x x*	base	rt	offset
6	5	5	16

Format:

SWCz rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form a virtual address. Coprocessor unit *z* sources a word, which the processor writes to the addressed memory location.

The data to be stored is defined by individual coprocessor specifications.

This instruction is not valid for use with CP0.

If either of the two least-significant bits of the effective address is non-zero, an address error exception occurs.

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ (ReverseEndian \,||\, 0^2)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,||\, 0^2)$
data ← COPzSW (byte, rt)
StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ (ReverseEndian \,||\, 0^2)$
$byte \leftarrow vAddr_{2...0}\ xor\ (BigEndianCPU \,||\, 0^2)$
data ← COPzSW (byte,rt)
StoreMemory (uncached, WORD, data, pAddr, vAddr DATA)

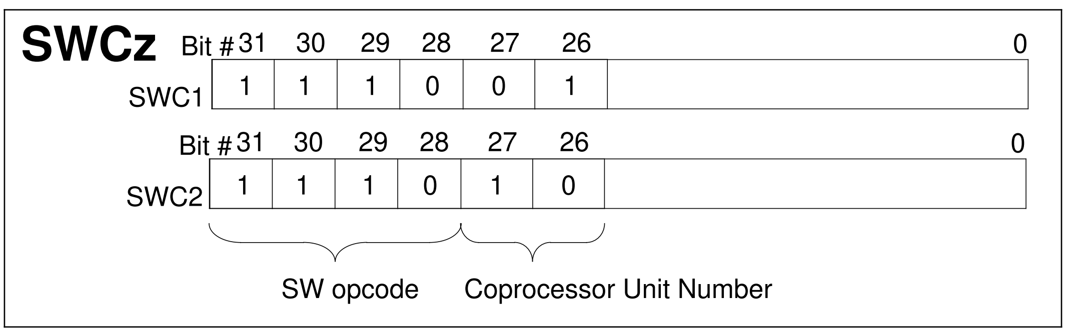
*See the table “Opcode Bit Encoding” on next page, or “CPU Instruction Opcode Bit Encoding” at the end of Appendix A.

SWCz Store Word From Coprocessor (Continued) SWCz

Exceptions:

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception
Coprocessor unusable exception

Opcode Bit Encoding:



SWL Store Word Left SWL

31 – 26	25 – 21	20 – 16	15 – 0
SWL 101010	base	rt	offset
6	5	5	16

Format:

SWL rt, offset(base)

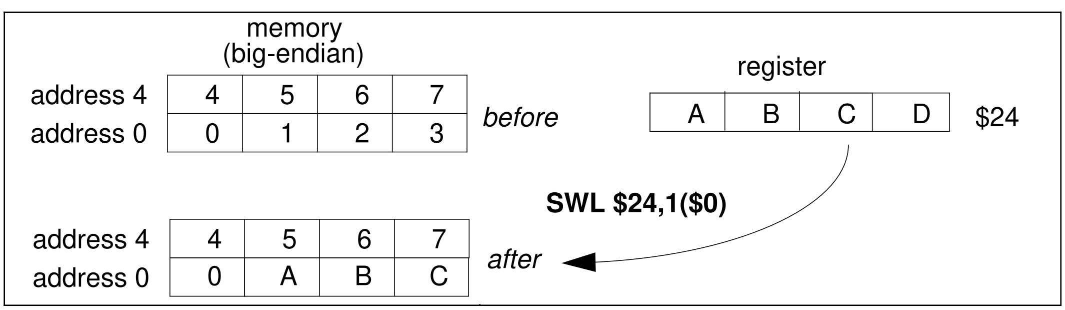
Description:

This instruction can be used with the SWR instruction to store the contents of a register into four consecutive bytes of memory, when the bytes cross a word boundary. SWL stores the left portion of the register into the appropriate part of the high-order word of memory; SWR stores the right portion of the register into the appropriate part of the low-order word.

The SWL instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address which may specify an arbitrary byte. It alters only the word in memory which contains that byte. From one to four bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the most-significant byte of the register and copies it to the specified byte in memory; then it copies bytes from register to memory until it reaches the low-order byte of the word in memory.

No address exceptions due to alignment are possible.



SWL Store Word Left (Continued) SWL

Operation:

```
32    T:  vAddr ← ((offset15)^16 || offset 15...0) + GPR[base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
          If BigEndianMem = 0 then
               pAddr ← pAddr31...2 || 0^2
          endif
          byte ← vAddr1...0 xor BigEndianCPU^2
          if (vAddr2 xor BigEndianCPU) = 0 then
               data ← 0^32 || 0^24-8*byte || GPR[rt]31...24-8*byte
          else
               data ← 0^24-8*byte || GPR[rt]31...24-8*byte || 0^32
          endif
          Storememory (uncached, byte, data, pAddr, vAddr, DATA)
64    T:  vAddr ← ((offset15)^48 || offset 15...0) + GPR[base]
          (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
          pAddr ← pAddrPSIZE – 1...3 || (pAddr2...0 xor ReverseEndian^3)
          If BigEndianMem = 0 then
               pAddr ← pAddr31...2 || 0^2
          endif
          byte ← vAddr1...0 xor BigEndianCPU^2
          if (vAddr2 xor BigEndianCPU) = 0 then
               data ← 0^32 || 0^24-8*byte || GPR[rt]31...24-8*byte
          else
               data ← 0^24-8*byte || GPR[rt]31...24-8*byte || 0^32
          endif
          StoreMemory(uncached, byte, data, pAddr, vAddr, DATA)
```

SWL — Store Word Left (Continued) — SWL

Given a doubleword in a register and a doubleword in memory, the operation of SWL is as follows:

SWL

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

vAddr$_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1			
	destination	type	offset		destination	type	offset	
			LEM	BEM			LEM	BEM
0	I J K L M N O E	0	0	7	E F G H M N O P	3	4	0
1	I J K L M N E F	1	0	6	I E F G M N O P	2	4	1
2	I J K L M E F G	2	0	5	I J E F M N O P	1	4	2
3	I J K L E F G H	3	0	4	I J K E M N O P	0	4	3
4	I J K E M N O P	0	4	3	I J K L E F G H	3	0	4
5	I J E F M N O P	1	4	2	I J K L M E F G	2	0	5
6	I E F G M N O P	2	4	1	I J K L M N E F	1	0	6
7	E F G H M N O P	3	4	0	I J K L M N O E	0	0	7

LEM Little-endian memory (BigEndianMem = 0)
BEM BigEndianMem = 1
Type AccessType (see Table 2-1) sent to memory
Offset pAddr$_{2...0}$ sent to memory

Exceptions:

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception

SWR Store Word Right SWR

31 26	25 21	20 16	15 0
SWR 1 0 1 1 1 0	base	rt	offset
6	5	5	16

Format:

SWR rt, offset(base)

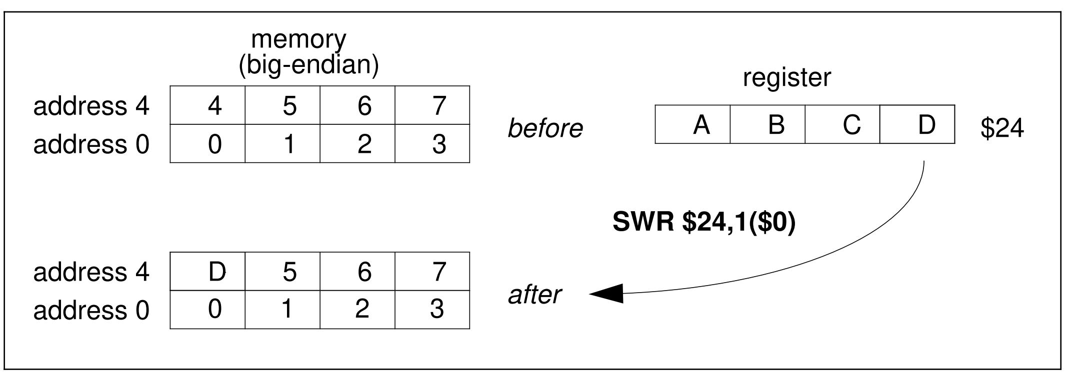
Description:

This instruction can be used with the SWL instruction to store the contents of a register into four consecutive bytes of memory, when the bytes cross a boundary between two words. SWR stores the right portion of the register into the appropriate part of the low-order word; SWL stores the left portion of the register into the appropriate part of the low-order word of memory.

The SWR instruction adds its sign-extended 16-bit *offset* to the contents of general register *base* to form a virtual address which may specify an arbitrary byte. It alters only the word in memory which contains that byte. From one to four bytes will be stored, depending on the starting byte specified.

Conceptually, it starts at the least-significant (rightmost) byte of the register and copies it to the specified byte in memory; then copies bytes from register to memory until it reaches the high-order byte of the word in memory.

No address exceptions due to alignment are possible.



SWR Store Word Right (Continued) SWR

Operation:

32 T: $vAddr \leftarrow ((offset_{15})^{16} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ ReverseEndian^3)$
If BigEndianMem = 0 then
$pAddr \leftarrow pAddr_{31...2} \,||\, 0^2$
endif
$byte \leftarrow vAddr_{1...0}\ xor\ BigEndianCPU^2$
if ($vAddr_2$ xor BigEndianCPU) = 0 then
$data \leftarrow 0^{32} \,||\, GPR[rt]_{31-8*byte...0} \,||\, 0^{8*byte}$
else
$data \leftarrow GPR[rt]_{31-8*byte...0} \,||\, 0^{8*byte} \,||\, 0^{32}$
endif
Storememory (uncached, WORD-byte, data, pAddr, vAddr, DATA)

64 T: $vAddr \leftarrow ((offset_{15})^{48} \,||\, offset_{15...0}) + GPR[base]$
$(pAddr, uncached) \leftarrow AddressTranslation\ (vAddr, DATA)$
$pAddr \leftarrow pAddr_{PSIZE-1...3} \,||\, (pAddr_{2...0}\ xor\ ReverseEndian^3)$
If BigEndianMem = 0 then
$pAddr \leftarrow pAddr_{31...2} \,||\, 0^2$
endif
$byte \leftarrow vAddr_{1...0}\ xor\ BigEndianCPU^2$
if ($vAddr_2$ xor BigEndianCPU) = 0 then
$data \leftarrow 0^{32} \,||\, GPR[rt]_{31-8*byte...0} \,||\, 0^{8*byte}$
else
$data \leftarrow GPR[rt]_{31-8*byte...0} \,||\, 0^{8*byte} \,||\, 0^{32}$
endif
StoreMemory(uncached, WORD-byte, data, pAddr, vAddr, DATA)

SWR — Store Word Right (Continued) — SWR

Given a doubleword in a register and a doubleword in memory, the operation of SWR is as follows:

SWR

Register	A	B	C	D	E	F	G	H
Memory	I	J	K	L	M	N	O	P

vAddr$_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1			
	destination	type	offset		destination	type	offset	
			LEM	BEM			LEM	BEM
0	I J K L E F G H	3	0	4	H J K L M N O P	0	7	0
1	I J K L F G H P	2	1	4	G H K L M N O P	1	6	0
2	I J K L G H O P	1	2	4	F G H L M N O P	2	5	0
3	I J K L H N O P	0	3	4	E F G H M N O P	3	4	0
4	E F G H M N O P	3	4	0	I J K L H N O P	0	3	4
5	F G H L M N O P	2	5	0	I J K L G H O P	1	2	4
6	G H K L M N O P	1	6	0	I J K L F G H P	2	1	4
7	H J K L M N O P	0	7	0	I J K L E F G H	3	0	4

LEM	Little-endian memory (BigEndianMem = 0)
BEM	BigEndianMem = 1
Type	AccessType (see Table 2-1) sent to memory
Offset	pAddr$_{2...0}$ sent to memory

Exceptions:

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception

SYNC — Synchronize — SYNC

31–26	25–6	5–0
SPECIAL 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	SYNC 0 0 1 1 1 1
6	20	6

Format:

SYNC

Description:

The SYNC instruction ensures that any loads and stores fetched *prior to* the present instruction are completed before any loads or stores *after* this instruction are allowed to start. Use of the SYNC instruction to serialize certain memory references may be required in a multiprocessor environment for proper synchronization. For example:

Processor A			Processor B	
SW	R1, DATA	1:	LW	R2, FLAG
LI	R2, 1		BEQ	R2, R0, 1B
SYNC			NOP	
SW	R2, FLAG		SYNC	
			LW	R1, DATA

The SYNC in processor A prevents DATA being written after FLAG, which could cause processor B to read stale data. The SYNC in processor B prevents DATA from being read before FLAG, which could likewise result in reading stale data. For processors which only execute loads and stores in order, with respect to shared memory, this instruction is a NOP.

LL and SC instructions implicitly perform a SYNC.

This instruction is allowed in User mode.

Operation:

32, 64	T:	SyncOperation()

Exceptions:

None

SYSCALL System Call SYSCALL

31 26	25 6	5 0
SPECIAL 0 0 0 0 0 0	Code	SYSCALL 0 0 1 1 00
6	20	6

Format:

SYSCALL

Description:

A system call exception 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:    SystemCallException
```

Exceptions:

System Call exception

TEQ — Trap If Equal — TEQ

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

Format:

TEQ rs, rt

Description:

The contents of general register *rt* are compared to general register *rs*. If the contents of general register *rs* are equal to the contents of general register *rt*, a trap exception occurs.

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:   if GPR[rs] = GPR[rt] then
                      TrapException
                 endif
```

Exceptions:

Trap exception

TEQI Trap If Equal Immediate TEQI

31 26	25 21	20 16	15 0
REGIMM 0 0 0 0 0 1	rs	TEQI 0 1 1 0 0	immediate
6	5	5	16

Format:

TEQI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. If the contents of general register *rs* are equal to the sign-extended *immediate*, a trap exception occurs.

Operation:

32	T:	if GPR[rs] = $(\text{immediate}_{15})^{16} \,\|\|\, \text{immediate}_{15...0}$ then
		TrapException
		endif

64	T:	if GPR[rs] = $(\text{immediate}_{15})^{48} \,\|\|\, \text{immediate}_{15...0}$ then
		TrapException
		endif

Exceptions:

Trap exception

TGE Trap If Greater Than Or Equal TGE

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

Format:

TGE rs, rt

Description:

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are greater than or equal to the contents of general register *rt*, a trap exception occurs.

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:   if GPR[rs] ≥ GPR[rt] then
                    TrapException
              endif
```

Exceptions:

Trap exception

TGEI Trap If Greater Than Or Equal Immediate TGEI

31–26	25–21	20–16	15–0
REGIMM 0 0 0 0 0 1	rs	TGEI 0 1 0 0 0	immediate
6	5	5	16

Format:

TGEI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are greater than or equal to the sign-extended *immediate*, a trap exception occurs.

Operation:

```
32    T:  if GPR[rs] ≥ (immediate_15)^16 || immediate_15...0 then
              TrapException
          endif
64    T:  if GPR[rs] ≥ (immediate_15)^48 || immediate_15...0 then
              TrapException
          endif
```

Exceptions:

Trap exception

TGEIU — Trap If Greater Than Or Equal Immediate Unsigned — TGEIU

31–26	25–21	20–16	15–0
REGIMM 0 0 0 0 0 1	rs	TGEIU 0 1 0 0 1	immediate
6	5	5	16

Format:

TGEIU rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are greater than or equal to the sign-extended *immediate*, a trap exception occurs.

Operation:

```
32    T:  if (0 || GPR[rs]) ≥ (0 || (immediate_15)^16 || immediate_15...0) then
              TrapException
          endif
64    T:  if (0 || GPR[rs]) ≥ (0 || (immediate_15)^48 || immediate_15...0) then
              TrapException
          endif
```

Exceptions:

Trap exception

TGEU Trap If Greater Than Or Equal Unsigned TGEU

31–26	25–21	20–16	15–6	5–0
SPECIAL 000000	rs	rt	code	TGEU 110001
6	5	5	10	6

Format:

TGEU rs, rt

Description:

The contents of general register *rt* are compared to the contents of general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are greater than or equal to the contents of general register *rt*, a trap exception occurs.

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:

```
T:      if (0 || GPR[rs]) ≥ (0 || GPR[rt]) then
            TrapException
        endif
```

Exceptions:

Trap exception

TLBP Probe TLB For Matching Entry TLBP

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	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.

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: $\text{Index} \leftarrow 1 \mid\mid 0^{25} \mid\mid \text{undefined}^6$
for i in 0...TLBEntries–1
  if $(\text{TLB[i]}_{95...77} = \text{EntryHi}_{31...12})$ and $(\text{TLB[i]}_{76}$ or
  $(\text{TLB[i]}_{71...64} = \text{EntryHi}_{7...0}))$ then
    $\text{Index} \leftarrow 0^{26} \mid\mid \text{i}_{5...0}$
  endif
endfor

64 T: $\text{Index} \leftarrow 1 \mid\mid 0^{25} \mid\mid \text{undefined}^6$
for i in 0...TLBEntries–1
  if $(\text{TLB[i]}_{167...141}$ and not $(0^{15} \mid\mid \text{TLB[i]}_{216...205}))$
  $= \text{EntryHi}_{39...13})$ and not $(0^{15} \mid\mid \text{TLB[i]}_{216...205}))$ and
  $(\text{TLB[i]}_{140}$ *or* $(\text{TLB[i]}_{135...128} = \text{EntryHi}_{7...0}))$ then
    $\text{Index} \leftarrow 0^{26} \mid\mid \text{i}_{5...0}$
  endif
endfor

Exceptions:

Coprocessor unusable exception

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. 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.

Operation:

32	T:	PageMask $\leftarrow$ TLB[Index$_{5...0}$]$_{127...96}$ EntryHi $\leftarrow$ TLB[Index$_{5...0}$]$_{95...64}$ and not TLB[Index$_{5...0}$]$_{127...96}$ EntryLo1 $\leftarrow$TLB[Index$_{5...0}$]$_{63...32}$ EntryLo0 $\leftarrow$ TLB[Index$_{5...0}$]$_{31...0}$
64	T:	PageMask $\leftarrow$ TLB[Index$_{5...0}$]$_{255...192}$ EntryHi $\leftarrow$ TLB[Index$_{5...0}$]$_{191...128}$ and not TLB[Index$_{5...0}$]$_{255...192}$ EntryLo1 $\leftarrow$TLB[Index$_{5...0}$]$_{127...65}$ || TLB[Index$_{5...0}$]$_{140}$ EntryLo0 $\leftarrow$ TLB[Index$_{5...0}$]$_{63...1}$ || TLB[Index$_{5...0}$]$_{140}$

Exceptions:

Coprocessor unusable exception

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.

Operation:

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

Exceptions:

Coprocessor unusable exception

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.

Operation:

32, 64 T: $\text{TLB}[\text{Random}_{5\ldots0}] \leftarrow$
PageMask || (EntryHi and not PageMask) || EntryLo1 || EntryLo0

Exceptions:

Coprocessor unusable exception

TLT — Trap If Less Than — TLT

31 … 26	25 … 21	20 … 16	15 … 6	5 … 0
SPECIAL 0 0 0 0 0 0	rs	rt	code	TLT 1 1 0 0 1 0
6	5	5	10	6

Format:

TLT rs, rt

Description:

The contents of general register *rt* are compared to general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the contents of general register *rt*, a trap exception occurs.

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:   if GPR[rs] < GPR[rt] then
                   TrapException
              endif
```

Exceptions:

Trap exception

TLTI Trap If Less Than Immediate TLTI

31 26	25 21	20 16	15 0
REGIMM 0 0 0 0 0 1	rs	TLTI 0 1 0 1 0	immediate
6	5	5	16

Format:

TLTI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the sign-extended *immediate*, a trap exception occurs.

Operation:

```
32    T:  if GPR[rs] < (immediate_15)^16 || immediate_15...0 then
              TrapException
          endif
64    T:  if GPR[rs] < (immediate_15)^48 || immediate_15...0 then
              TrapException
          endif
```

Exceptions:

Trap exception

TLTIU Trap If Less Than Immediate Unsigned TLTIU

31 26	25 21	20 16	15 0
REGIMM 0 0 0 0 0 1	rs	TLTIU 0 1 0 1 1	immediate
6	5	5	16

Format:

TLTIU rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. Considering both quantities as signed integers, if the contents of general register *rs* are less than the sign-extended *immediate*, a trap exception occurs.

Operation:

32 T: if (0 || GPR[rs]) < (0 || $(\text{immediate}_{15})^{16}$ || $\text{immediate}_{15...0}$) then
TrapException
endif

64 T: if (0 || GPR[rs]) < (0 || $(\text{immediate}_{15})^{48}$ || $\text{immediate}_{15...0}$) then
TrapException
endif

Exceptions:

Trap exception

TLTU Trap If Less Than Unsigned TLTU

31–26	25–21	20–16	15–6	5–0
SPECIAL 000000	rs	rt	code	TLTU 110011
6	5	5	10	6

Format:

TLTU rs, rt

Description:

The contents of general register *rt* are compared to general register *rs*. Considering both quantities as unsigned integers, if the contents of general register *rs* are less than the contents of general register *rt*, a trap exception occurs.

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:   if (0 || GPR[rs]) < (0 || GPR[rt]) then
                TrapException
            endif
```

Exceptions:

Trap exception

TNE Trap If Not Equal TNE

31 ... 26	25 ... 21	20 ... 16	15 ... 6	5 ... 0
SPECIAL 0 0 0 0 0 0	rs	rt	code	TNE 1 1 0 1 1 0
6	5	5	10	6

Format:

TNE rs, rt

Description:

The contents of general register *rt* are compared to general register *rs*. If the contents of general register *rs* are not equal to the contents of general register *rt*, a trap exception occurs.

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:   if GPR[rs] ≠ GPR[rt] then
                TrapException
            endif
```

Exceptions:

Trap exception

TNEI Trap If Not Equal Immediate TNEI

31 — 26	25 — 21	20 — 16	15 — 0
REGIMM 0 0 0 0 0 1	rs	TNEI 0 1 1 1 0	immediate
6	5	5	16

Format:

TNEI rs, immediate

Description:

The 16-bit *immediate* is sign-extended and compared to the contents of general register *rs*. If the contents of general register *rs* are not equal to the sign-extended *immediate*, a trap exception occurs.

Operation:

```
32    T:  if GPR[rs] ≠ (immediate_15)^16 || immediate_15...0 then
              TrapException
          endif
64    T:  if GPR[rs] ≠ (immediate_15)^48 || immediate_15...0 then
              TrapException
          endif
```

Exceptions:

Trap exception

XOR　　　Exclusive Or　　　XOR

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	XOR 1 0 0 1 1 0
6	5	5	5	5	6

Format:

XOR rd, rs, rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical exclusive OR operation.

The result is placed into general register *rd.*

Operation:

32, 64	T:	GPR[rd] ← GPR[rs] xor GPR[rt]

Exceptions:

None

XORI Exclusive OR Immediate XORI

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

Format:

XORI rt, rs, immediate

Description:

The 16-bit *immediate* is zero-extended and combined with the contents of general register *rs* in a bit-wise logical exclusive OR operation.

The result is placed into general register *rt*.

Operation:

32	T:	$GPR[rt] \leftarrow GPR[rs]$ xor $(0^{16} \,\|\|\,$ immediate$)$
64	T:	$GPR[rt] \leftarrow GPR[rs]$ xor $(0^{48} \,\|\|\,$ immediate$)$

Exceptions:

None

CPU Instruction Opcode Bit Encoding

The remainder of this Appendix presents the opcode bit encoding for the CPU instruction set (ISA and extensions), as implemented by the R4000. Figure A-2 lists the R4000 Opcode Bit Encoding.

Opcode

31...29 \ 28...26	0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	COP1	COP2	*	BEQL	BNEL	BLEZL	BGTZL
3	DADDIε	DADDIUε	LDLε	LDRε	*	*	*	*
4	LB	LH	LWL	LW	LBU	LHU	LWR	LWUε
5	SB	SH	SWL	SW	SDLε	SDRε	SWR	CACHE δ
6	LL	LWC1	LWC2	*	LLDε	LDC1	LDC2	LDε
7	SC	SWC1	SWC2	*	SCDε	SDC1	SDC2	SDε

SPECIAL function

5...3 \ 2...0	0	1	2	3	4	5	6	7
0	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1	JR	JALR	*	*	SYSCALL	BREAK	*	SYNC
2	MFHI	MTHI	MFLO	MTLO	DSLLVε	*	DSRLVε	DSRAVε
3	MULT	MULTU	DIV	DIVU	DMULTε	DMULTUε	DDIVε	DDIVUε
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	*	*	SLT	SLTU	DADDε	DADDUε	DSUBε	DSUBUε
6	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7	DSLLε	*	DSRLε	DSRAε	DSLL32ε	*	DSRL32ε	DSRA32ε

REGIMM rt

20...19 \ 18...16	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1	TGEI	TGEIU	TLTI	TLTIU	TEQI	*	TNEI	*
2	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3	*	*	*	*	*	*	*	*

COPz rs

25, 24 \ 23...21	0	1	2	3	4	5	6	7
0	MF	DMFε	CF	γ	MT	DMTε	CT	γ
1	BC	γ	γ	γ	γ	γ	γ	γ
2	CO							
3								

Figure A-2 R4000 Opcode Bit Encoding

COPz rt

20...19 \ 18...16	0	1	2	3	4	5	6	7
0	BCF	BCT	BCFL	BCTL	γ	γ	γ	γ
1	γ	γ	γ	γ	γ	γ	γ	γ
2	γ	γ	γ	γ	γ	γ	γ	γ
3	γ	γ	γ	γ	γ	γ	γ	γ

CP0 Function

5 ... 3 \ 2 ... 0	0	1	2	3	4	5	6	7
0	ϕ	TLBR	TLBWI	ϕ	ϕ	ϕ	TLBWR	ϕ
1	TLBP	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
2	ξ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
3	ERET χ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
2	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
3	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

Figure A-2 (cont.) R4000 Opcode Bit Encoding

Key:

- * Operation codes marked with an asterisk cause reserved instruction exceptions in all current implementations and are reserved for future versions of the architecture.
- γ Operation codes marked with a gamma cause a reserved instruction exception. They are reserved for future versions of the architecture.
- δ Operation codes marked with a delta are valid only for R4000 processors with CP0 enabled, and cause a reserved instruction exception on other processors.
- ϕ Operation codes marked with a phi are invalid but do not cause reserved instruction exceptions in R4000 implementations.
- ξ Operation codes marked with a xi cause a reserved instruction exception on R4000 processors.
- χ Operation codes marked with a chi are valid only on R4000.
- ε Operation codes marked with epsilon are valid when the processor is operating either in the Kernel mode or in the 64-bit non-Kernel (User or Supervisor) mode. These instructions cause a reserved instruction exception if 64-bit operation is not enabled in User or Supervisor mode.