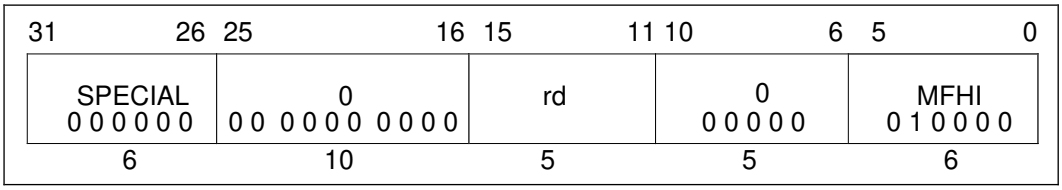
MFHI — Move From HI — MFHI



Format:

MFHI rd

Description:

The contents of special register *HI* are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions which follow a MFHI instruction may not be any of the instructions which modify the *HI* register: MULT, MULTU, DIV, DIVU, MTHI, DMULT, DMULTU, DDIV, DDIVU.

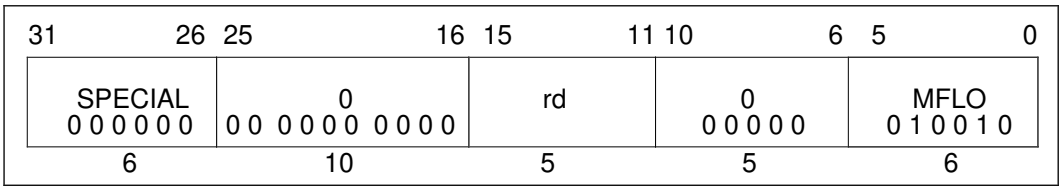
Operation:

```
32, 64      T:      GPR[rd] ← HI
```

Exceptions:

None

MFLO Move From Lo MFLO



Format:

MFLO rd

Description:

The contents of special register *LO* are loaded into general register *rd*.

To ensure proper operation in the event of interruptions, the two instructions which follow a MFLO instruction may not be any of the instructions which modify the *LO* register: MULT, MULTU, DIV, DIVU, MTLO, DMULT, DMULTU, DDIV, DDIVU.

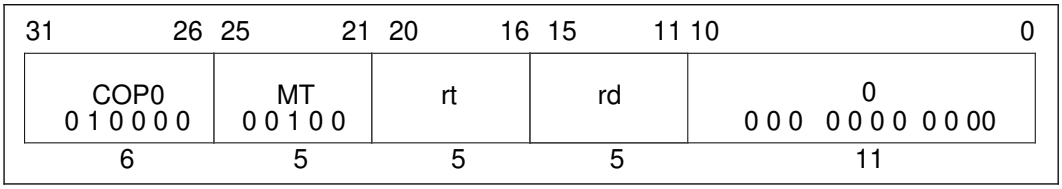
Operation:

```
32, 64      T:      GPR[rd] ← LO
```

Exceptions:

None

MTC0 Move To System Control Coprocessor MTC0



Format:

MTC0 rt, rd

Description:

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

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.

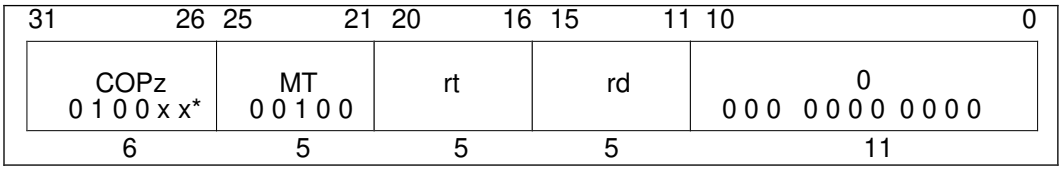
Operation:

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

Exceptions:

Coprocessor unusable exception

MTCz Move To Coprocessor MTCz



Format:

MTCz rt, rd

Description:

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

Operation:

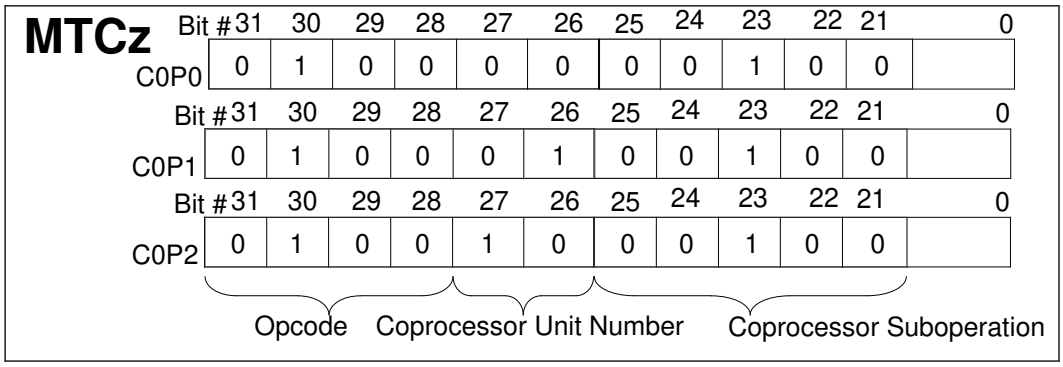
```
32    T:    data ← GPR[rt]
      T+1:  CPR[z,rd] ← data

64    T:    data ← GPR[rt]31...0
      T+1:  if rd0 = 0
                CPR[z,rd4...1 || 0] ← CPR[z, rd4...1 || 0]63...32 || data
            else
                CPR[z,rd4...1 || 0] ← data || CPR[z,rd4...1 || 0]31...0
            endif
```

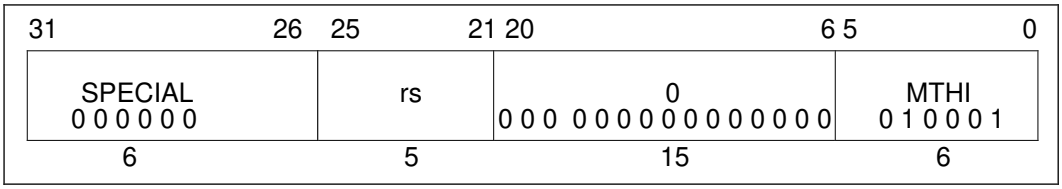
Exceptions:

Coprocessor unusable exception

***Opcode Bit Encoding:**



MTHI Move To HI MTHI



Format:

MTHI rs

Description:

The contents of general register *rs* are loaded into special register *HI*.

If a MTHI operation is executed following a MULT, MULTU, DIV, or DIVU instruction, but before any MFLO, MFHI, MTLO, or MTHI instructions, the contents of special register *LO* are undefined.

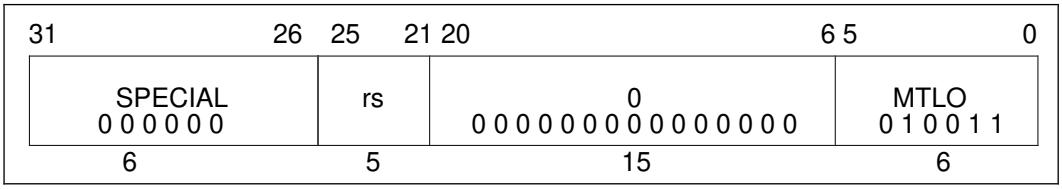
Operation:

```
32,64     T–2:  HI ← undefined
          T–1:  HI ← undefined
          T:    HI ← GPR[rs]
```

Exceptions:

None

MTLO Move To LO MTLO



Format:

MTLO rs

Description:

The contents of general register *rs* are loaded into special register *LO.*

If a MTLO operation is executed following a MULT, MULTU, DIV, or DIVU instruction, but before any MFLO, MFHI, MTLO, or MTHI instructions, the contents of special register *HI* are undefined.

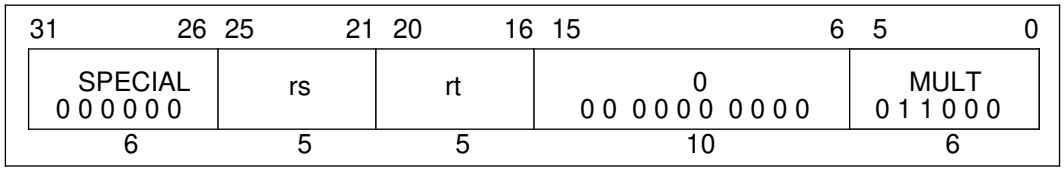
Operation:

```
32,64    T–2:  LO ← undefined
          T–1:  LO ← undefined
          T:    LO ← GPR[rs]
```

Exceptions:

None

MULT Multiply MULT



Format:

MULT rs, rt

Description:

The contents of general registers *rs* and *rt* are multiplied, treating both operands as 32-bit 2's complement values. No integer overflow exception occurs under any circumstances. In 64-bit mode, the operands must be valid 32-bit, sign-extended values.

When the operation completes, the low-order word of the double result is loaded into special register *LO*, and the high-order 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 these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two other instructions.

MULT — Multiply (continued) — MULT

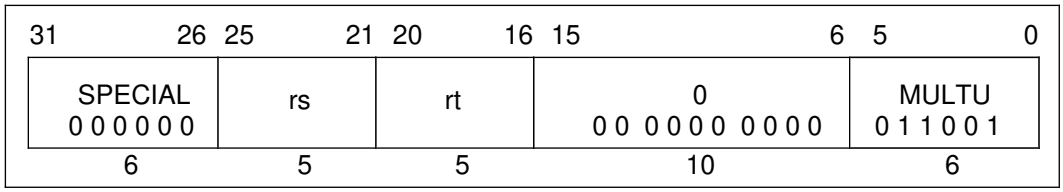
Operation:

```
32    T–2: LO   ← undefined
           HI   ← undefined
      T–1: LO   ← undefined
           HI   ← undefined
      T:   t    ← GPR[rs] * GPR[rt]
           LO   ← t31...0
           H I  ← t63...32
64    T–2: LO   ← undefined
           HI   ← undefined
      T–1: LO   ← undefined
           HI   ← undefined
      T:   t    ← GPR[rs]31...0 * GPR[rt]31...0
           LO   ← (t31)32 || t31...0
           HI   ← (t63)32 || t63...32
```

Exceptions:

None

MULTU Multiply Unsigned MULTU



Format:

MULTU rs, rt

Description:

The contents of general register *rs* and the contents of general register *rt* are multiplied, treating both operands as unsigned values. No overflow exception occurs under any circumstances. In 64-bit mode, the operands must be valid 32-bit, sign-extended values.

When the operation completes, the low-order word of the double result is loaded into special register *LO*, and the high-order 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 these instructions are undefined. Correct operation requires separating reads of *HI* or *LO* from writes by a minimum of two instructions.

MULTU — Multiply Unsigned (continued) — MULTU

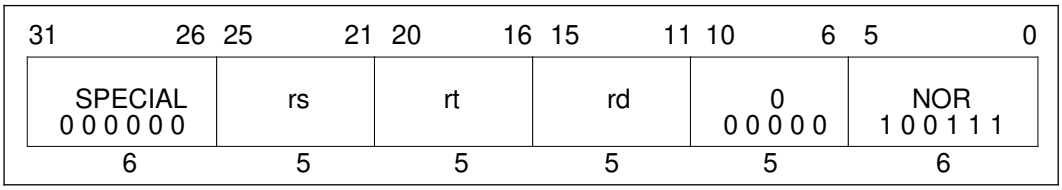
Operation:

```
32   T–2: LO   ← undefined
          HI   ← undefined
     T–1: LO   ← undefined
          HI   ← undefined
     T:   t    ← (0 || GPR[rs]) * (0 || GPR[rt])
          LO   ← t31...0
          HI   ← t63...32
64   T–2: LO   ← undefined
          HI   ← undefined
     T–1: LO   ← undefined
          HI   ← undefined
     T:   t    ← (0 || GPR[rs]31...0) * (0 || GPR[rt]31...0)
          LO   ← (t31)32 || t31...0
          HI   ← (t63)32 || t63...32
```

Exceptions:

None

NOR Nor NOR



Format:

NOR rd, rs, rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical NOR operation. The result is placed into general register *rd*.

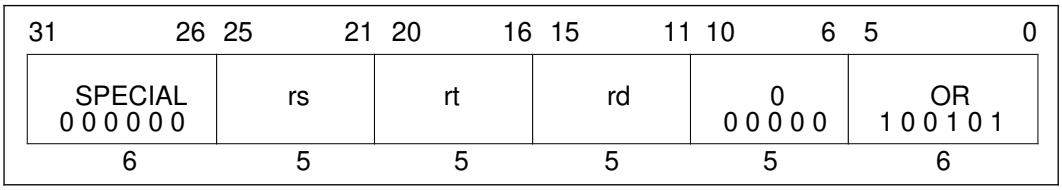
Operation:

```
32, 64      T:    GPR[rd] ← GPR[rs] nor GPR[rt]
```

Exceptions:

None

OR Or OR



Format:

OR rd, rs, rt

Description:

The contents of general register *rs* are combined with the contents of general register *rt* in a bit-wise logical OR operation. The result is placed into general register *rd*.

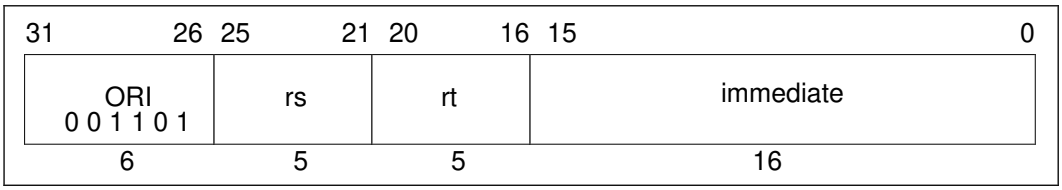
Operation:

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

Exceptions:

None

ORI — Or Immediate — ORI



Format:

ORI 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 OR operation. The result is placed into general register *rt*.

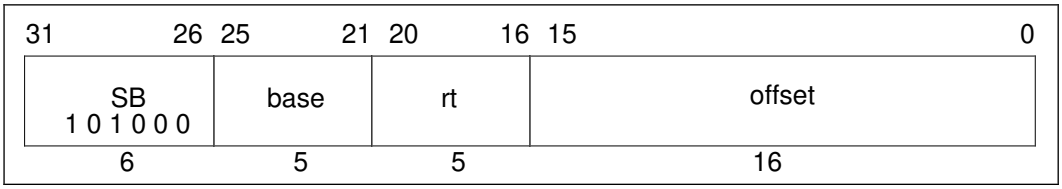
Operation:

```
32    T:    GPR[rt] ← GPR[rs]31...16 || (immediate or GPR[rs]15...0)
64    T:    GPR[rt] ← GPR[rs]63...16 || (immediate or GPR[rs]15...0)
```

Exceptions:

None

SB Store Byte SB



Format:

SB 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 least-significant byte of register *rt* is stored at the effective address.

Operation:

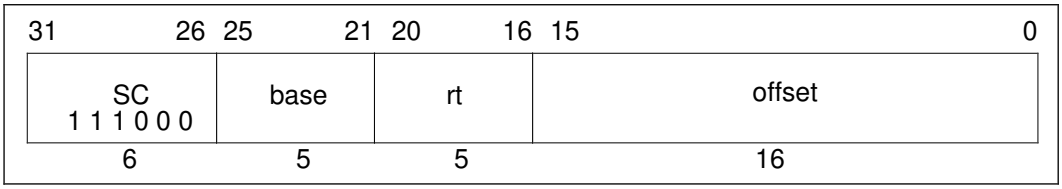
```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE-1...3 || (pAddr2...0 xor ReverseEndian^3)
           byte ← vAddr2...0 xor BigEndianCPU^3
           data ← GPR[rt]63–8*byte...0 || 0^8*byte
           StoreMemory (uncached, BYTE, data, pAddr, vAddr, DATA)

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE-1...3 || (pAddr2...0 xor ReverseEndian^3)
           byte ← vAddr2...0 xor BigEndianCPU^3
           data ← GPR[rt]63–8*byte...0 || 0^8*byte
           StoreMemory (uncached, BYTE, data, pAddr, vAddr, DATA)
```

Exceptions:

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

SC Store Conditional SC



Format:

SC 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 conditionally stored at the memory location specified by the effective address.

If any other processor or device has modified the physical address since the time of the previous Load Linked instruction, or if an ERET instruction occurs between the Load Linked instruction and this store instruction, the store fails and is inhibited from taking place.

The success or failure of the store operation (as defined above) is indicated by the contents of general register *rt* after execution of the instruction. A successful store sets the contents of general register *rt* to 1; an unsuccessful store sets it to 0.

The operation of Store Conditional is undefined when the address is different from the address used in the last Load Linked.

This instruction is available in User mode; it is not necessary for CP0 to be enabled.

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

If this instruction should both fail and take an exception, the exception takes precedence.

SC — Store Conditional (continued) — SC

Operation:

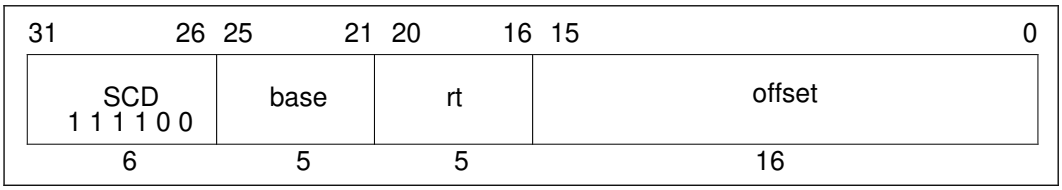
```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE-1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
           data ← GPR[rt]63-8*byte...0 || 0^8*byte
           if LLbit then
               StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)
           endif
           GPR[rt] ← 0^31 || LLbit

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE-1...3 || (pAddr2...0 xor (ReverseEndian || 0^2))
           data ← GPR[rt]63-8*byte...0 || 0^8*byte
           if LLbit then
               StoreMemory (uncached, WORD, data, pAddr, vAddr, DATA)
           endif
           GPR[rt] ← 0^63 || LLbit
```

Exceptions:

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

SCD Store Conditional Doubleword SCD



Format:

SCD 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 conditionally stored at the memory location specified by the effective address.

If any other processor or device has modified the physical address since the time of the previous Load Linked Doubleword instruction, or if an ERET instruction occurs between the Load Linked Doubleword instruction and this store instruction, the store fails and is inhibited from taking place.

The success or failure of the store operation (as defined above) is indicated by the contents of general register *rt* after execution of the instruction. A successful store sets the contents of general register *rt* to 1; an unsuccessful store sets it to 0.

The operation of Store Conditional Doubleword is undefined when the address is different from the address used in the last Load Linked Doubleword.

This instruction is available in User mode; it is not necessary for CP0 to be enabled.

If either of the three least-significant bits of the effective address is non-zero, an address error exception takes place.

SCD Store Conditional Doubleword (continued) SCD

If this instruction should both fail and take an exception, the exception takes precedence.

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.

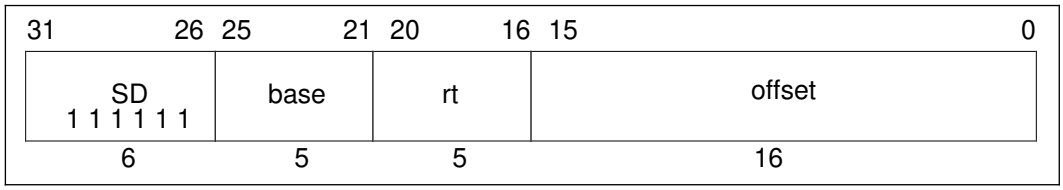
Operation:

```
64    T:    vAddr ← ((offset15)^48 || offset15...0) + GPR[base]
            (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
            data  ← GPR[rt]
            if LLbit then
                StoreMemory (uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)
            endif
            GPR[rt] ←  0^63 || LLbit
```

Exceptions:

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception
Reserved instruction exception (R4000 in 32-bit mode)

SD Store Doubleword SD



Format:

SD 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 three least-significant bits of the effective address are non-zero, an address error 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.

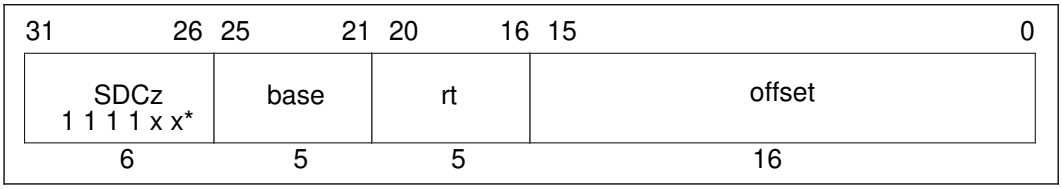
Operation:

```
64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           data ← GPR[rt]
           StoreMemory (uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)
```

Exceptions:

TLB refill exception
TLB invalid exception
TLB modification exception
Bus error exception
Address error exception
Reserved instruction exception (R4000 in 32-bit user mode
R4000 in 32-bit supervisor mode)

SDCz — Store Doubleword From Coprocessor — SDCz



Format:

SDCz 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 doubleword, which the processor writes to the addressed memory location. The data to be stored is defined by individual coprocessor specifications.

If any of the three least-significant bits of the effective address are non-zero, an address error exception takes place.

This instruction is not valid for use with CP0.

This instruction is undefined when the least-significant bit of the *rt* field is non-zero.

Operation:

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           data ← COPzSD(rt),
           StoreMemory (uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)

64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           data ← COPzSD(rt),
           StoreMemory (uncached, DOUBLEWORD, data, pAddr, vAddr, DATA)
```

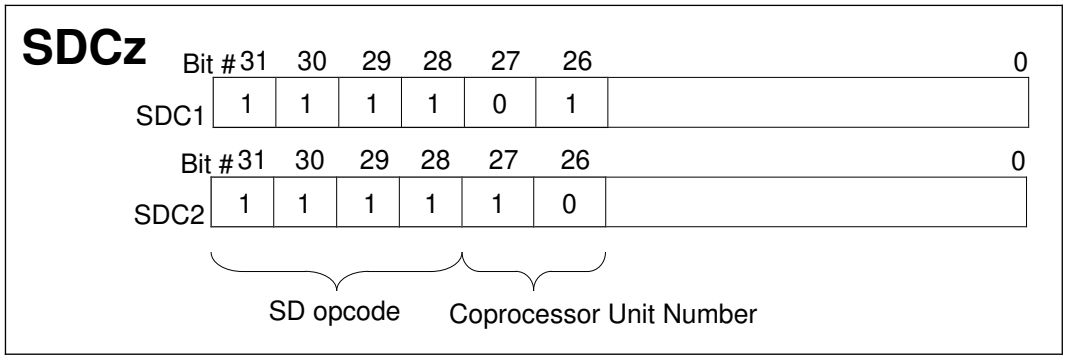
*See the table, "Opcode Bit Encoding" on next page, or "CPU Instruction Opcode Bit Encoding" at the end of Appendix A.

SDCz Store Doubleword From Coprocessor (continued) SDCz

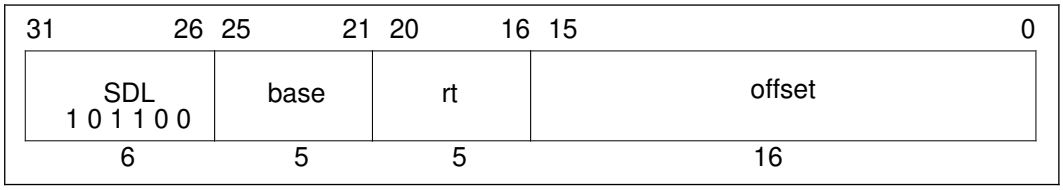
Exceptions:

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

Opcode Bit Encoding:



SDL Store Doubleword Left SDL



Format:

SDL rt, offset(base)

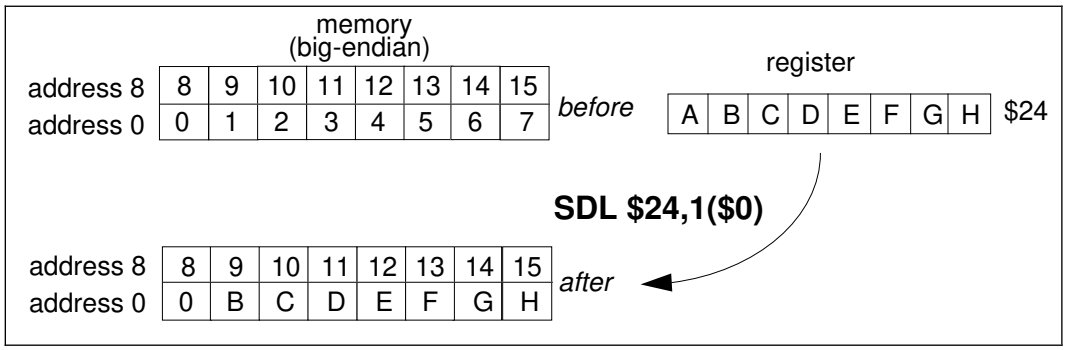
Description:

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

The SDL 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.



SDL Store Doubleword Left (continued) SDL

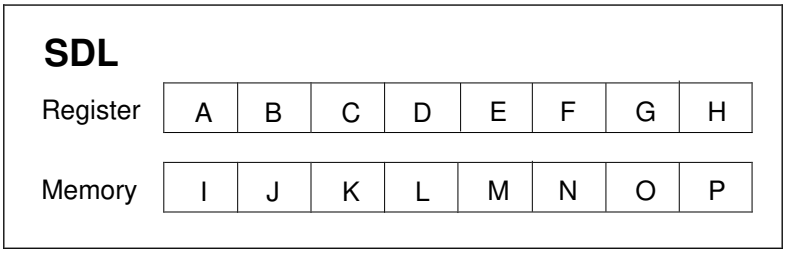
This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

```
64    T:   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...3 || 0^3
           endif
           byte ← vAddr2...0 xor BigEndianCPU^3
           data ← 0^(56–8*byte) || GPR[rt]63...56–8*byte
           Storememory (uncached, byte, data, pAddr, vAddr, DATA)
```

SDL Store Doubleword Left (continued) SDL

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



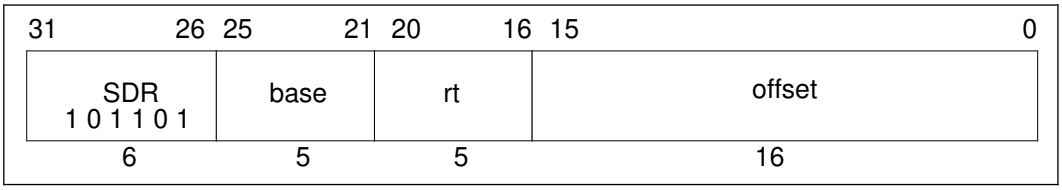
$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1			
	destination	type	offset		destination	type	offset	
			LEM	BEM			LEM	BEM
0	I J K L M N O A	0	0	7	A B C D E F G H	7	0	0
1	I J K L M N A B	1	0	6	I A B C D E F G	6	0	1
2	I J K L M A B C	2	0	5	I J A B C D E F	5	0	2
3	I J K L A B C D	3	0	4	I J K A B C D E	4	0	3
4	I J K A B C D E	4	0	3	I J K L A B C D	3	0	4
5	I J A B C D E F	5	0	2	I J K L M A B C	2	0	5
6	I A B C D E F G	6	0	1	I J K L M N A B	1	0	6
7	A B C D E F G H	7	0	0	I J K L M N O A	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
Reserved instruction exception (R4000 in 32-bit mode)

SDR Store Doubleword Right SDR



Format:

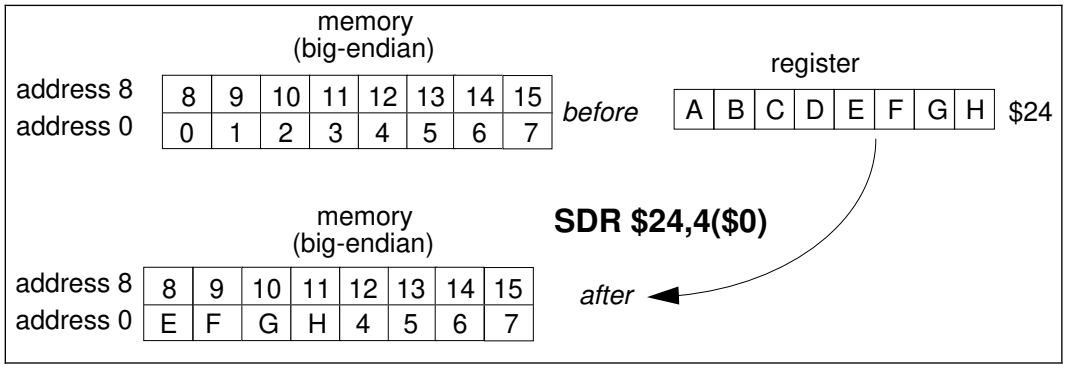
SDR rt, offset(base)

Description:

This instruction can be used with the SDL instruction to store the contents of a register into eight consecutive bytes of memory, when the bytes cross a boundary between two doublewords. SDR stores the right portion of the register into the appropriate part of the low-order doubleword; SDL stores the left portion of the register into the appropriate part of the low-order doubleword of memory.

The SDR 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 eight 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 it 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.



SDR Store Doubleword Right (continued) SDR

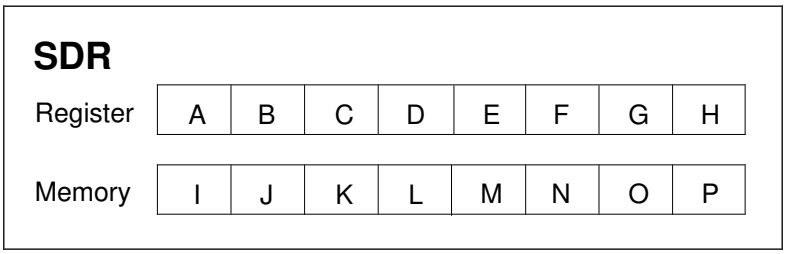
This operation is only defined for the R4000 operating in 64-bit mode. Execution of this instruction in 32-bit mode causes a reserved instruction exception.

Operation:

```
64    T:  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 ← pAddrPSIZE – 31...3 || 0^3
          endif
          byte ← vAddr1...0 xor BigEndianCPU^3
          data ← GPR[rt]63–8*byte || 0^8*byte
          StoreMemory (uncached, DOUBLEWORD-byte, data, pAddr, vAddr, DATA)
```

SDR — Store Doubleword Right (continued) — SDR

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



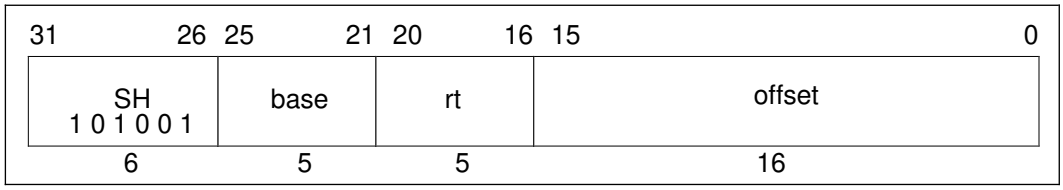
$vAddr_{2..0}$	BigEndianCPU = 0				BigEndianCPU = 1			
	destination	type	offset LEM	offset BEM	destination	type	offset LEM	offset BEM
0	A B C D E F G H	7	0	0	H J K L M N O P	0	7	0
1	B C D E F G H P	6	1	0	G H K L M N O P	1	6	0
2	C D E F G H O P	5	2	0	F G H L M N O P	2	5	0
3	D E F G H N O P	4	3	0	E F G H M N O P	3	4	0
4	E F G H M N O P	3	4	0	D E F G H N O P	4	3	0
5	F G H L M N O P	2	5	0	C D E F G H O P	5	2	0
6	G H K L M N O P	1	6	0	B C D E F G H P	6	1	0
7	H J K L M N O P	0	7	0	A B C D E F G H	7	0	0

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
Reserved instruction exception (R4000 in 32-bit mode)

SH Store Halfword SH



Format:

SH rt, offset(base)

Description:

The 16-bit *offset* is sign-extended and added to the contents of general register *base* to form an unsigned effective address. The least-significant halfword of register *rt* is stored at the effective address. If the least-significant bit of the effective address is non-zero, an address error exception occurs.

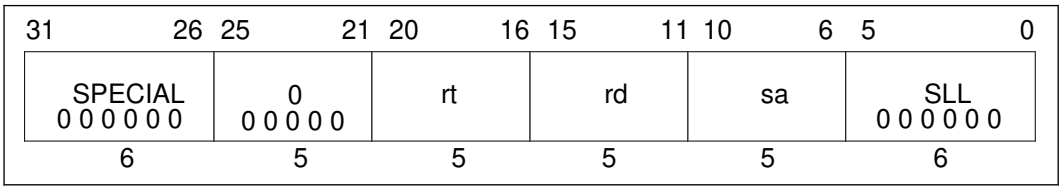
Operation:

```
32    T:   vAddr ← ((offset15)^16 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE-1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           data ← GPR[rt]63–8*byte...0 || 0^8*byte
           StoreMemory (uncached, HALFWORD, data, pAddr, vAddr, DATA)
64    T:   vAddr ← ((offset15)^48 || offset15...0) + GPR[base]
           (pAddr, uncached) ← AddressTranslation (vAddr, DATA)
           pAddr ← pAddrPSIZE-1...3 || (pAddr2...0 xor (ReverseEndian^2 || 0))
           byte ← vAddr2...0 xor (BigEndianCPU^2 || 0)
           data ← GPR[rt]63–8*byte...0 || 0^8*byte
           StoreMemory (uncached, HALFWORD, data, pAddr, vAddr, DATA)
```

Exceptions:

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

SLL — Shift Left Logical — SLL



Format:

SLL rd, rt, sa

Description:

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

The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign extended when placed in the destination register. It is sign extended for all shift amounts, including zero; SLL with a zero shift amount truncates a 64-bit value to 32 bits and then sign extends this 32-bit value. SLL, unlike nearly all other word operations, does not require an operand to be a properly sign-extended word value to produce a valid sign-extended word result.

NOTE: SLL with a shift amount of zero may be treated as a NOP by some assemblers, at some optimization levels. If using SLL with a zero shift to truncate 64-bit values, check the assembler you are using.

Operation:

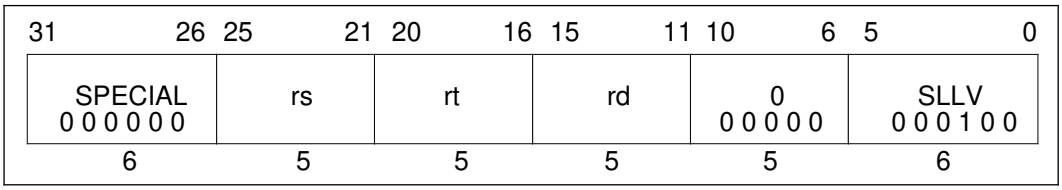
```
32    T:   GPR[rd] ← GPR[rt]31– sa...0 || 0sa

64    T:   s ← 0 || sa
           temp ← GPR[rt]31-s...0 || 0s
           GPR[rd] ← (temp31)32 || temp
```

Exceptions:

None

SLLV Shift Left Logical Variable SLLV



Format:

SLLV rd, rt, rs

Description:

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

The result is placed in register *rd*.

In 64-bit mode, the 32-bit result is sign extended when placed in the destination register. It is sign extended for all shift amounts, including zero; SLLV with a zero shift amount truncates a 64-bit value to 32 bits and then sign extends this 32-bit value. SLLV, unlike nearly all other word operations, does not require an operand to be a properly sign-extended word value to produce a valid sign-extended word result.

NOTE: SLLV with a shift amount of zero may be treated as a NOP by some assemblers, at some optimization levels. If using SLLV with a zero shift to truncate 64-bit values, check the assembler you are using.

Operation:

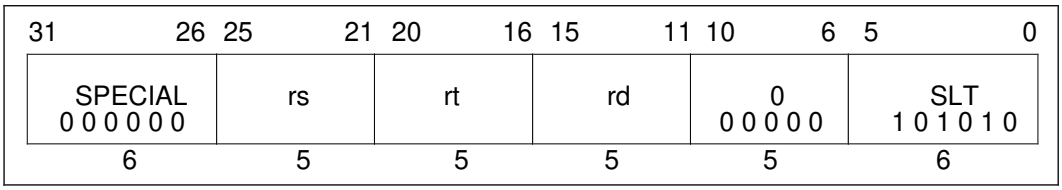
```
32    T:   s ← GP[rs]4...0
           GPR[rd]← GPR[rt](31–s)...0 || 0^s

64    T:   s ← 0 || GP[rs]4...0
           temp ← GPR[rt](31-s)...0 || 0^s
           GPR[rd] ← (temp31)^32 || temp
```

Exceptions:

None

SLT — Set On Less Than — SLT



Format:

SLT rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of 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*, the result is set to one; otherwise the result is set to zero.

The result is placed into general register *rd*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

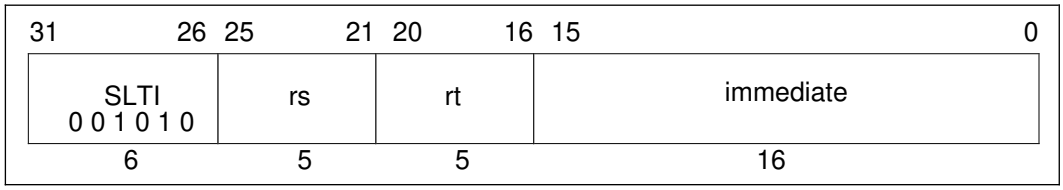
Operation:

```
32    T:    if GPR[rs] < GPR[rt] then
                GPR[rd] ← 0^31 || 1
            else
                GPR[rd] ← 0^32
            endif
64    T:    if GPR[rs] < GPR[rt] then
                GPR[rd] ← 0^63 || 1
            else
                GPR[rd] ← 0^64
            endif
```

Exceptions:

None

SLTI Set On Less Than Immediate SLTI



Format:

SLTI rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and subtracted from the contents of general register *rs*. Considering both quantities as signed integers, if *rs* is less than the sign-extended immediate, the result is set to one; otherwise the result is set to zero.

The result is placed into general register *rt*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

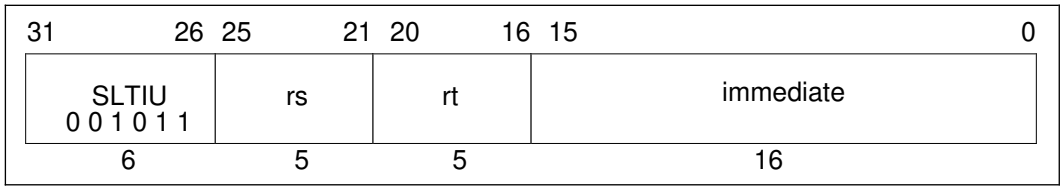
Operation:

```
32    T:    if GPR[rs] < (immediate15)^16 || immediate15...0 then
                    GPR[rd] ← 0^31 || 1
            else
                    GPR[rd] ← 0^32
            endif
64    T:    if GPR[rs] < (immediate15)^48 || immediate15...0 then
                    GPR[rd] ← 0^63 || 1
            else
                    GPR[rd] ← 0^64
            endif
```

Exceptions:

None

SLTIU — Set On Less Than Immediate Unsigned — SLTIU



Format:

SLTIU rt, rs, immediate

Description:

The 16-bit *immediate* is sign-extended and subtracted from the contents of general register *rs*. Considering both quantities as unsigned integers, if *rs* is less than the sign-extended immediate, the result is set to one; otherwise the result is set to zero.

The result is placed into general register *rt*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

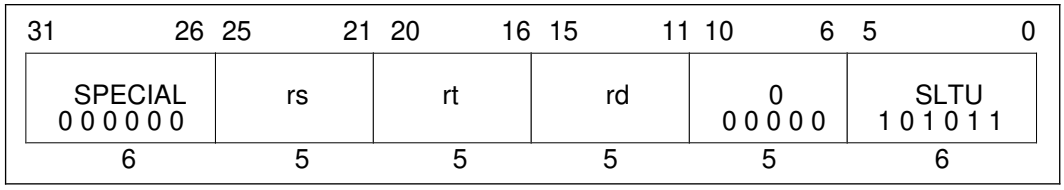
Operation:

```
32    T:   if (0 || GPR[rs]) < (immediate15)^16 || immediate15...0 then
                    GPR[rd] ← 0^31 || 1
           else
                    GPR[rd] ← 0^32
           endif
64    T:   if (0 || GPR[rs]) < (immediate15)^48 || immediate15...0 then
                    GPR[rd] ← 0^63 || 1
           else
                    GPR[rd] ← 0^64
           endif
```

Exceptions:

None

SLTU Set On Less Than Unsigned SLTU



Format:

SLTU rd, rs, rt

Description:

The contents of general register *rt* are subtracted from the contents of 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*, the result is set to one; otherwise the result is set to zero.

The result is placed into general register *rd*.

No integer overflow exception occurs under any circumstances. The comparison is valid even if the subtraction used during the comparison overflows.

Operation:

```
32    T:    if (0 || GPR[rs]) < 0 || GPR[rt] then
                    GPR[rd] ← 0^31 || 1
            else
                    GPR[rd] ← 0^32
            endif
64    T:    if (0 || GPR[rs]) < 0 || GPR[rt] then
                    GPR[rd] ← 0^63 || 1
            else
                    GPR[rd] ← 0^64
            endif
```

Exceptions:

None