**Commands**

| **Command** | **Definition** | **Pseudo code** |
| --- | --- | --- |
| AND rd, rs, rt | And | rd = rs & rt |
| ANDI rt, rs, immediate | And Immediate | rt = rs & immediate |
| OR rd, rs, rt | Or | rd = rs | rt |
| ORI rt, rs, immediate | Or Immediate | rt = rs | immediate |
| XOR rd, rs, rt | Exclusive Or | rd = rs ^ rt |
| XORI rt, rs, immediate | Exclusive Or Immediate | rd = rs ^ immediate |
| NOR rd, rs, rt | Nor | rd = ~(rs | rt) |
| SLT rd, rs, rt | Set On Less Than | rd = (rs < rt) ? 1 : 0 |
| SLTU rd, rs, rt | Set On Less Than Unsigned | rd = (rs < rt) ? 1 : 0 |
| SLTI rt, rs, immediate | Set On Less Than Immediate | rt = (rs < immediate) ? 1 : 0 |
| SLTIU rt, rs, immediate | Set On Less Than Immediate Unsigned | rt = (rs < immediate) ? 1 : 0 |
| ADD rd, rs, rt | Add | rd = rs + rt |
| ADDU rd, rs, rt | Add Unsigned | rd = rs + rt |
| ADDI rt, rs, immediate[1)](https://hack64.net/wiki/doku.php?id=r4300#fn__1) | Add Immediate | rd = rs + immediate |
| ADDIU rt, rs, immediate | Add Immediate Unsigned | rd = rs + immediate |
| SUB rd, rs, rt | Subtract | rd = rs - rt |
| SUBU rd, rs, rt | Subtract Unsigned | rd = rs - rt |
| MULT rs, rt | Multiply | lo = rs \* rt |
| MULTU rs, rt | Multiply Unsigned | lo = rs \* rt |
| DIV rs, rt | Divide | lo = rs / rt; hi = rs % rt |
| DIVU rs, rt | Divide Unsigned | lo = rs / rt; hi = rs % rt |
| SLL rd, rt, sa[2)](https://hack64.net/wiki/doku.php?id=r4300#fn__2) | Shift Left Logical | rd = rt << sa |
| SLLV rd, rt, rs | Shift Left Logical Variable | rd = rt << rs |
| SRA rd, rt, sa | Shift Right Arithmetic | rd = (int32)rt >> sa |
| SRAV rd, rt, rs | Shift Right Arithmetic Variable | rd = (int32)rt >> rs |
| SRL rd, rt, sa | Shift Right Logical | rd = (uint32)rt >> sa |
| SRLV rd, rt, rs | Shift Right Logical Variable | rd = (uint32)rt >> rs |
| DADD rd, rs, rt | Doubleword Add | rd = rs + rt |
| DADDU rd, rs, rt | Doubleword Add Unsigned | rd = rs + rt |
| DADDI rt, rs, immediate | Doubleword Add Immediate | rd = rs + immediate |
| DADDIU rt, rs, immediate | Doubleword Add Immediate Unsigned | rd = rs + immediate |
| DSUB rd, rs, rt | Doubleword Subtract | rd = rs - rt |
| DSUBU rd, rs, rt | Doubleword Subtract Unsigned | rd = rs - rt |
| DMULT rs, rt | Doubleword Multiply | lo = rs \* rt |
| DMULTU rs, rt | Doubleword Multiply Unsigned | lo = rs \* rt |
| DDIV rs, rt | Doubleword Divide | lo = rs / rt; hi = rs % rt |
| DDIVU rs, rt | Doubleword Divide Unsigned | lo = rs / rt; hi = rs % rt |
| DSLL rd, rt, sa | Doubleword Shift Left Logical | rd = rt << sa |
| DSLL32 rd, rt, sa | Doubleword Shift Left Logical + 32 | rd = rt << (sa + 32) |
| DSLLV rd, rt, rs | Doubleword Shift Left Logical Variable | rd = rt << rs |
| DSRA rd, rt, sa | Doubleword Shift Right Arithmetic | rd = (int64)rt >> sa |
| DSRA32 rd, rt, sa | Doubleword Shift Right Arithmetic + 32 | rd = (int64)rt >> (sa + 32) |
| DSRAV rd, rt, rs | Doubleword Shift Right Arithmetic Variable | rd = (int64)rt >> rs |
| DSRL rd, rt, sa | Doubleword Shift Right Logical | rd = (uint64)rt >> sa |
| DSRL32 rd, rt, sa | Doubleword Shift Right Logical + 32 | rd = (uint64)rd >> (sa + 32) |
| DSRLV rd, rt, rs | Doubleword Shift Right Logical Variable | rd = (uint64)rd >> rs |
| MFHI rd | Move From HI | rd = hi |
| MFLO rd | Move From LO | rd = lo |
| MTHI rs | Move To HI | hi = rs |
| MTLO rs | Move To LO | lo = rs |
| LUI rt, immediate | Load Upper Immediate | rt = immediate << 16 |
| LB rt, offset(rs) | Load Byte | rt = \*(int8\*)(rs + offset) |
| LBU rt, offset(rs) | Load Byte Unsigned | rt = \*(uint8\*)(rs + offset) |
| LH rt, offset(rs) | Load Halfword | rt = \*(int16\*)(rs + offset) |
| LHU rt, offset(rs) | Load Halfword Unsigned | rt = \*(uint16\*)(rs + offset) |
| LW rt, offset(rs) | Load Word | rt = \*(int32\*)(rs + offset) |
| LWU rt, offset(rs) | Load Word Unsigned | rt = \*(uint32\*)(rs + offset) |
| LWC1 ft, offset(rs) | Load Word To FPU[3)](https://hack64.net/wiki/doku.php?id=r4300#fn__3) | ft = \*(float\*)(rs + offset) |
| LWL rt, offset(rs) | Load Word Left |  |
| LWR rt, offset(rs) | Load Word Right |  |
| LD rt, offset(rs) | Load Doubleword | rt = \*(uint64\*)(rs + offset) |
| LDC1 ft, offset(rs) | Load Doubleword To FPU | ft = \*(double\*)(rs + offset) |
| LDL rt, offset(rs) | Load Doubleword Left |  |
| LDR rt, offset(rs) | Load Doubleword Right |  |
| LL rt, offset(rs) | Load Linked |  |
| LLD rt, offset(rs) | Load Linked Doubleword |  |
| SB rt, offset(rs) | Store Byte | \*(int8\*)(rs + offset) = rt |
| SH rt, offset(rs) | Store Halfword | \*(int16\*)(rs + offset) = rt |
| SW rt, offset(rs) | Store Word | \*(int32\*)(rs + offset) = rt |
| SWC1 ft, offset(rs) | Store Word From FPU | \*(float\*)(rs + offset) = ft |
| SWL rt, offset(rs) | Store Word Left |  |
| SWR rt, offset(rs) | Store Word Right |  |
| SD rt, offset(rs) | Store Doubleword | \*(int64\*)(rs + offset) = rt |
| SDC1 ft, offset(rs) | Store Doubleword From FPU | \*(double\*)(rs + offset) = rt |
| SDL rt, offset(rs) | Store Doubleword Left |  |
| SDR rt, offset(rs) | Store Doubleword Right |  |
| SC rt, offset(rs) | Store Conditional |  |
| SCD rt, offset(rs) | Store Conditional Doubleword |  |
| J target | Jump | PC = target |
| JR rs | Jump Register | PC = rs |
| JAL target | Jump And Link | RA = PC + 8; PC = target |
| JALR rs | Jump And Link Register | rd = PC + 8; PC = rs |
| JALR rd, rs | Jump And Link Register | rd = PC + 8; PC = rs |
| BEQ rs, rt, target | Branch On Equal | if(rs == rt) PC = target |
| BEQL rs, rt, target | Branch On Equal Likely | if(rs == rt) PC = target |
| BNE rs, rt, target | Branch On Not Equal | if(rs != rt) PC = target |
| BNEL rs, rt, target | Branch On Not Equal Likely | if(rs != rt) PC = target |
| BGTZ rs, target | Branch On Greater Than Zero | if(rs > 0) PC = target |
| BGTZL rs, target | Branch On Greater Than Zero Likely | if(rs > 0) PC = target |
| BLEZ rs, target | Branch On Less Than Or Equal To Zero | if(rs <= 0) PC = target |
| BLEZL rs, target | Branch On Less Than Or Equal To Zero Likely | if(rs <= 0) PC = target |
| BGEZ rs, target | Branch On Greater Than Or Equal To Zero | if(rs >= 0) PC = target |
| BGEZL rs, target | Branch On Greater Than Or Equal To Zero Likely | if(rs >= 0) PC = target |
| BGEZAL rs, target | Branch On Greater Than Or Equal To Zero And Link | if(rs >= 0) { RA = PC + 8; PC = target } |
| BGEZALL rs, target | Branch On Greater Than Or Equal To Zero And Link Likely | if(rs >= 0) { RA = PC + 8; PC = target } |
| BLTZ rs, target | Branch On Less Than Zero | if(rs < 0) PC = target |
| BLTZL rs, target | Branch On Less Than Zero Likely | if(rs < 0) PC = target |
| BLTZAL rs, target | Branch On Less Than Zero And Link | if(rs < 0) { RA = PC + 8; PC = target } |
| BLTZALL rs, target | Branch On Less Than Zero And Link Likely | if(rs < 0) { RA = PC + 8; PC = target } |
| MFC1 rt, fs | Move Word From FPU | rt = fs |
| DMFC1 rt, fs | Doubleword Move From FPU | rt = fs |
| MTC1 rt, fs | Move To FPU | fs = rt |
| DMTC1 rt, fs | Doubleword Move To FPU | fs = rt |
| CFC1 rt, fcr | Move Control Word From FPU | rt = fcr |
| CTC1 rt, fcr | Move Control Word To FPU | rt = fcr |
| MOV.S fd, fs | FPU Move Float | fd = fs |
| ABS.S fd, fs | FPU Absolute Value Float | fd = abs(fs) |
| NEG.S fd, fs | FPU Negate Float | fd = -fs |
| SQRT.S fd, fs | FPU Square Root Float | fd = sqrt(fs) |
| ADD.S fd, fs, ft | FPU Add Floats | fd = fs + ft |
| SUB.S fd, fs, ft | FPU Subtract Floats | fd = fs - ft |
| MUL.S fd, fs, ft | FPU Multiply Floats | fd = fs \* ft |
| DIV.S fd, fs, ft | FPU Divide Floats | fd = fs / ft |
| MOV.D fd, fs | FPU Move Double | fd = fs |
| ABS.D fd, fs | FPU Absolute Value Double | fd = abs(fs) |
| NEG.D fd, fs | FPU Negate Double | fd = -fs |
| SQRT.D fd, fs | FPU Square Root Double | fd = sqrt(fs) |
| ADD.D fd, fs, ft | FPU Add Doubles | fd = fs + ft |
| SUB.D fd, fs, ft | FPU Subtract Doubles | fd = fs - ft |
| MUL.D fd, fs, ft | FPU Multiply Doubles | fd = fs \* ft |
| DIV.D fd, fs, ft | FPU Divide Doubles | fd = fs / ft |
| FLOOR.L.S fd, fs | FPU Floor Float To Fixed-point Long | fd = (int64)floor(fs) |
| FLOOR.W.S fd, fs | FPU Floor Float To Fixed-point Word | fd = (int32)floor(fs) |
| ROUND.L.S fd, fs | FPU Round Float To Fixed-point Long | fd = (int64)round(fs) |
| ROUND.W.S fd, fs | FPU Round Float To Fixed-point Word | fd = (int32)round(fs) |
| TRUNC.L.S fd, fs | FPU Truncate Float To Fixed-point Long | fd = (int64)trunc(fs) |
| TRUNC.W.S fd, fs | FPU Truncate Float To Fixed-point Word | fd = (int32)trunc(fs) |
| CEIL.L.S fd, fs | FPU Ceiling Float To Fixed-point Long | fd = (int64)ceil(fs) |
| CEIL.W.S fd, fs | FPU Ceiling Float To Fixed-point Word | fd = (int32)ceil(fs) |
| FLOOR.L.D fd, fs | FPU Floor Double To Fixed-point Long | fd = (int64)floor(fs) |
| FLOOR.W.D fd, fs | FPU Floor Double To Fixed-point Word | fd = (int32)floor(fs) |
| ROUND.L.D fd, fs | FPU Round Double To Fixed-point Long | fd = (int64)round(fs) |
| ROUND.W.D fd, fs | FPU Round Double To Fixed-point Word | fd = (int32)round(fs) |
| TRUNC.L.D fd, fs | FPU Truncate Double To Fixed-point Long | fd = (int64)trunc(fs) |
| TRUNC.W.D fd, fs | FPU Truncate Double To Fixed-point Word | fd = (int32)trunc(fs) |
| CEIL.L.D fd, fs | FPU Ceiling Double To Fixed-point Long | fd = (int64)ceil(fs) |
| CEIL.W.D fd, fs | FPU Ceiling Double To Fixed-point Word | fd = (int32)ceil(fs) |
| CVT.D.S fd, fs | FPU Convert Float To Double | fd = (double)fs |
| CVT.W.S fd, fs | FPU Convert Float To Fixed-point Word | fd = (int32)fs |
| CVT.L.S fd, fs | FPU Convert Float To Fixed-Point Long | fd = (int64)fs |
| CVT.S.D fd, fs | FPU Convert Double To Float | fd = (float)fs |
| CVT.W.D fd, fs | FPU Convert Double To Fixed-point Word | fd = (int32)fs |
| CVT.L.D fd, fs | FPU Convert Double To Fixed-Point Long | fd = (int64)fs |
| CVT.S.W fd, fs | FPU Convert Fixed-Point Word To Float | fd = (float)fs |
| CVT.S.L fd, fs | FPU Convert Fixed-Point Long To Float | fd = (float)fs |
| CVT.D.W fd, fs | FPU Convert Fixed-Point Word To Double | fd = (double)fs |
| CVT.D.L fd, fs | FPU Convert Fixed-Point Long To Double | fd = (double)fs |
| C.EQ.S fs, ft | FPU Compare Equal Float | FCR31.C = (fs == ft) |
| C.LT.S fs, ft | FPU Compare Less Than Float | FCR31.C = (fs < ft) |
| C.LE.S fs, ft | FPU Compare Less Than or Equal Float | FCR31.C = (fs <= ft) |
| C.EQ.D fs, ft | FPU Compare Equal Double | FCR31.C = (fs == ft) |
| C.LT.D fs, ft | FPU Compare Less Than Double | FCR31.C = (fs < ft) |
| C.LE.D fs, ft | FPU Compare Less Than or Equal Double | FCR31.C = (fs <= ft) |
| C.F.S fs, ft | FPU Compare False Float | FCR31.C = FALSE |
| C.UN.S fs, ft | FPU Compare Unordered Float | FCR31.C = (isNaN(fs) || isNaN(ft)) |
| C.UEQ.S fs, ft | FPU Compare Unordered or Equal Float | FCR31.C = (isNaN(fs) || isNaN(ft) || (fs == ft)) |
| C.OLT.S fs, ft | FPU Compare Ordered Less Than Float | FCR31.C = (!isNaN(fs) && !isNaN(ft) && (fs < ft)) |
| C.ULT.S fs, ft | FPU Compare Unordered or Less Than Float | FCR31.C = (isNaN(fs) || isNaN(ft) || (fs < ft)) |
| C.OLE.S fs, ft | FPU Compare Ordered Less Than Or Equal Float | FCR31.C = (!isNaN(fs) && !isNaN(ft) && (fs <= ft)) |
| C.ULE.S fs, ft | FPU Compare Unordered or Less Than or Equal Float | FCR31.C = (isNaN(fs) || isNaN(ft) || (fs <= ft)) |
| C.SF.S fs, ft | FPU Compare Signaling False Float | FCR31.C = FALSE |
| C.NGLE.S fs, ft | FPU Compare Not Greater Or Less Than or Equal Float | FCR31.C = (isNaN(fs) || isNaN(ft)) |
| C.SEQ.S fs, ft | FPU Compare Signaling Equal Float | FCR31.C = (fs == ft) |
| C.NGL.S fs, ft | FPU Compare Not Greater Or Less Than Float | FCR31.C = !((fs > ft) || (fs < ft)) |
| C.NGE.S fs, ft | FPU Compare Not Greater Than or Equal Float | FCR31.C = !(fs >= ft) |
| C.NGT.S fs, ft | FPU Compare Not Greater Than Float | FCR31.C = !(fs > ft) |
| C.F.D fs, ft | FPU Compare False Double | FCR31.C = FALSE |
| C.UN.D fs, ft | FPU Compare Unordered Double | FCR31.C = (isNaN(fs) || isNaN(ft)) |
| C.UEQ.D fs, ft | FPU Compare Unordered or Equal Double | FCR31.C = (isNaN(fs) || isNaN(ft) || (fs == ft)) |
| C.OLT.D fs, ft | FPU Compare Ordered Less Than Double | FCR31.C = (!isNaN(fs) && !isNaN(ft) && (fs < ft)) |
| C.ULT.D fs, ft | FPU Compare Unordered or Less Than Double | FCR31.C = (isNaN(fs) || isNaN(ft) || (fs < ft)) |
| C.OLE.D fs, ft | FPU Compare Ordered Less Than Or Equal Double | FCR31.C = (!isNaN(fs) && !isNaN(ft) && (fs <= ft)) |
| C.ULE.D fs, ft | FPU Compare Unordered or Less Than or Equal Double | FCR31.C = (isNaN(fs) || isNaN(ft) || (fs <= ft)) |
| C.SF.D fs, ft | FPU Compare Signaling False Double | FCR31.C = FALSE |
| C.NGLE.D fs, ft | FPU Compare Not Greater Or Less Than or Equal Double | FCR31.C = (isNaN(fs) || isNaN(ft)) |
| C.SEQ.D fs, ft | FPU Compare Signaling Equal Double | FCR31.C = (fs == ft) |
| C.NGL.D fs, ft | FPU Compare Not Greater Or Less Than Double | FCR31.C = !((fs > ft) || (fs < ft)) |
| C.NGE.D fs, ft | FPU Compare Not Greater Than or Equal Double | FCR31.C = !(fs >= ft) |
| C.NGT.D fs, ft | FPU Compare Not Greater Than Double | FCR31.C = !(fs > ft) |
| BC1F offset | Branch On FPU False | if(FCR31.C == 0) PC = offset |
| BC1FL offset | Branch On FPU False Likely | if(FCR31.C == 0) PC = offset |
| BC1T offset | Branch On FPU True | if(FCR31.C == 1) PC = offset |
| BC1TL offset | Branch On FPU True Likely | if(FCR31.C == 1) PC = offset |
| MFC0 rt, rd | Move From Coprocessor 0 | rt = rd |
| MTC0 rt, rd | Move To Coprocessor 0 | rd = rt |
| CACHE op, offset(base) | Cache Operation |  |
| TEQ rs, rt | Trap If Equal | if(rs == rt) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TEQI rs, immediate | Trap If Equal Immediate | if(rs == immediate) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TGE rs, rt | Trap If Greater Than Or Equal | if(rs >= rt) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TGEI rs, immediate | Trap If Greater Than Or Equal Immedate | if(rs >= immediate) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TGEIU rs, immediate | Trap If Greater Than Or Equal Immediate Unsigned | if(rs >= immediate) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TGEU rs, rt | Trap If Greater Than Or Equal Unsigned | if(rs >= rt) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TLT rs, rt | Trap If Less Than | if(rs < rt) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TLTI rs, immediate | Trap If Less Than Immediate | if(rs < immediate) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TLTIU rs, immediate | Trap If Less Than Immediate Unsigned | if(rs < immediate) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TLTU rs, rt | Trap If Less Than Unsigned | if(rs < rt) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TNE rs, rt | Trap If Not Equal | if(rs != rt) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| TNEI rs, immediate | Trap If Not Equal Immediate | if(rs != immediate) { COP0\_CAUSE |= (13 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 } |
| SYNC | Synchronize | (No operation on R4300) |
| SYSCALL | System Call | COP0\_CAUSE |= (8 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 |
| BREAK | Breakpoint | COP0\_CAUSE |= (9 << 2); COP0\_EPC = PC; PC = exc\_vector\_base + 0x180 |
| ERET | Return From Exception | PC = COP0\_EPC (or COP0\_ERROREPC) LLBit = 0 |
| TLBP | Probe TLB For Matching Entry |  |
| TLBR | Read Indexed TLB Entry |  |
| TLBWI | Write Indexed TLB Entry |  |
| TLBWR | Write Random TLB Entry |  |

**General Purpose Registers**

| **Number** | **Name** | **Preserved** | **Purpose** |
| --- | --- | --- | --- |
| 0 | R0 | n/a | Hardwired zero |
| 1 | AT | no | Assembler temporary value |
| 2:3 | V0:V1 | no | Subroutine return value |
| 4:7 | A0:A3 | no | Subroutine arguments |
| 8:15 | T0:T7 | no | Temporary values |
| 16:23 | S0:S7 | yes | Saved values |
| 24:25 | T8:T9 | no | Temporary values |
| 26:27 | K0:K1 | n/a | Reserved by the kernel |
| 28 | GP | yes | Global pointer |
| 29 | SP | yes | Stack pointer |
| 30 | S8 or FP | yes | Saved value or frame pointer |
| 31 | RA | yes | Return address |

* Excluding R0 and RA, the register purposes noted above are conventional; they are not enforced by the processor.
* By convention if a subroutine uses a “preserved” register, the subroutine must restore that register to its original value before returning.

**Floating-point Registers (COP1)**

| **Number** | **Name** | **Preserved** | **Purpose** |
| --- | --- | --- | --- |
| 0:2 | F0:F2 | no | Subroutine return value |
| 4:10 | F4:F10 | no | Temporary values |
| 12:14 | F12:F14 | no | Subroutine arguments |
| 16:18 | F16:F18 | no | Temporary values |
| 20:30 | F20:F30 | yes | Saved values |

* When the [COP0 Status Register](https://hack64.net/wiki/doku.php?id=r4300#cop0_status_register)'s FR bit is 0, only even numbered registers should be used.
* All register purposes noted above are conventional; they are not enforced by the processor.
* By convention if a subroutine uses a “preserved” register, the subroutine must restore that register to its original value before returning.

**Floating-point FCR31 (Control/Status) Register (COP1)**

| 25|24|23| 18|17|16|15|14|13|12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| 0|

|0000000| .| .|00000| .| .| .| .| .| .| .| .| .| .| .| .| .| .| .| .|..|

|-------|FS|C |-----|CE|CV|CZ|CO|CU|CI|EV|EZ|EO|EU|EI|FV|FZ|FO|FU|FI|RM|

| 7 |1 |1 | 5 | 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|2 |

| **Field** | **Description** |
| --- | --- |
| FS | Enables flashing of denormalized numbers (See [Page 213 of the manual](https://hack64.net/docs/VR43XX.pdf#page=213)) |
| C | Compare bit (1: TRUE, 0: FALSE) Result of the last compare (C.COND.FMT) operation |
| CE | Cause bit: Unimplemented operation |
| CV | Cause bit: Invalid operation |
| CZ | Cause bit: Division by Zero |
| CO | Cause bit: Overflow |
| CU | Cause bit: Underflow |
| CI | Cause bit: Inexact operation |
| EV | Enable bit: Invalid operation |
| EZ | Enable bit: Division by Zero |
| EO | Enable bit: Overflow |
| EU | Enable bit: Underflow |
| EI | Enable bit: Inexact operation |
| FV | Flag bit: Invalid operation |
| FZ | Flag bit: Division by Zero |
| FO | Flag bit: Overflow |
| FU | Flag bit: Underflow |
| FI | Flag bit: Inexact operation |
| RM | [Rounding Mode](https://hack64.net/wiki/doku.php?id=r4300#rounding_mode) |

* Cause bits indicate the last exception - more than one may be set at a time.
* Enable bits mask exceptions when set to 0.
* Like the Cause bits, the Flag bits indicate the last exceptions, however they can only be cleared by writing a new value to FCR31 using the CTC1 instruction.

**Rounding modes**

| **Value** | **Mnemonic** | **Description** |
| --- | --- | --- |
| 0 | RN | Round to nearest representable value |
| 1 | RZ | Round towards zero |
| 2 | RP | Round towards positive infinity |
| 3 | RM | Round towards negative infinity |

**COP0 Status Register**

| 31| 30| 29| 28|27|26|25| 24|23| 22|21|20|19|18|17|16| 8| 7| 6| 5| 4| 2| 1| 0|

| .| .| .| .| .| .| .| .| 0| .| .| .| .| .| 0| 1|........| .| .| .| ..| .| .| .|

|CU3|CU2|CU1|CU0|RP|FR|RE|ITS| -|BEV|TS|SR| -|CH|CE|DE| IM |KX|SX|UX|KSU|ERL|EXL|IE|

| 1 | 1 | 1 | 1 |1 |1 |1 |1 | 1|1 |1 |1 | 1|1 |1 |1 | 8 |1 |1 |1 |2 |1 |1 |1 |

| **Field** | **Description** |
| --- | --- |
| CU3 | (Reserved) |
| CU2 | (Reserved) |
| CU1 | Coprocessor 1 (FPU) Usability (1: Usable, 0: Unusable) |
| CU0 | Coprocessor 0 Usability (1: Usable, 0: Unusable) COP0 is always usable while in kernel mode, regardless of setting |
| RP | Reduce Power (0: Normal, 1: Low Power Mode) Clock frequency is reduced to one-quarter speed when enabled |
| FR | Additional floating point registers (0: 16 registers, 1: 32 registers) |
| RE | Reverse Endian in User mode (0: Disabled, 1: Reversed) |
| ITS | Enable instruction trace support (0: False, 1: True) |
| BEV | Bootstrap exception vector (0: Normal, 1: Bootstrap) Controls [Exception Vector locations](https://hack64.net/wiki/doku.php?id=r4300#exception_vector_locations) |
| TS | TLB shutdown has occurred (0: False, 1: True) On R4300i, the TLB does not shutdown and the processor will continue execution, but the TS bit is still set. |
| SR | Soft reset or NMI has occurred (0: False, 1: True) |
| CH | CP0 Condition bit (0: False, 1: True) |
| CE | (Unused on R4300i) |
| DE | (Unused on R4300i) |
| IM | Interrupt Mask (0: Disabled, 1: Enabled) |
| KX | Enable 64-bit addressing in Kernel mode (0: Disabled, 1: Enabled) |
| SX | Enable 64-bit addressing and operations in Supervisor mode (0: Disabled, 1: Enabled) |
| UX | Enable 64-bit addressing and operations in User mode (0: Disabled, 1: Enabled) |
| KSU | Mode (10: User, 01: Supervisor, 00: Kernel) |
| ERL | Error level (0: Normal, 1: Error) |
| EXL | Exception level (0: Normal, 1: Exception) |
| IE | Global interrupt enable (0: Disabled, 1: Enabled) |

**Exception Vector Locations**

| **Exception** | **Base (BEV=0)** | **Base (BEV=1)** | **Offset** |
| --- | --- | --- | --- |
| Reset & NMI | n/a | 0xBFC000000 | 0x000 |
| TLB Miss | 0x80000000 | 0xBFC000200 | 0x000 |
| XTLB Miss | 0x80000000 | 0xBFC000200 | 0x080 |
| Other | 0x80000000 | 0xBFC000200 | 0x180 |

**COP0 Cause Register**

|31|30|28| 16| 8|7| 2| 0|

| .| 0|..|............|........|0|.....|00|

|BD| -|CE|------------| IP |-| EXC |--|

|1 |1 |2 | 12 | 8 |1| 5 |2 |

| **Field** | **Description** |
| --- | --- |
| BD | Branch delay (1: Last exception occurred in delay slot, 0: Normal) |
| CE | Coprocessor number for coprocessor unusable exception |
| IP | [Interrupt](https://hack64.net/wiki/doku.php?id=r4300#interrupts) pending (1: Interrupt, 0: No Interrupt) IP7 Timer interrupt IP6:2 External normal interrupts IP1:0 Software interrupt |
| EXC | Exception code |

**Exception Codes**

| **Code** | **Mnemonic** | **Description** | **Generated…** |
| --- | --- | --- | --- |
| 0 | Int | [Interrupt](https://hack64.net/wiki/doku.php?id=r4300#interrupts) | - When one of the eight interrupt conditions are asserted |
| 1 | Mod | TLB Modification exception | - When the TLB entry that matches the the virtual address referenced by the store instruction is marked as read-only (the D bit = 0) |
| 2 | TLBL | TLB Invalid exception (load or instruction fetch) | - When an attempt is made to read from an mapped area in a TLB segment that is marked invalid |
| 3 | TLBS | TLB Invalid exception (store) | - When an attempt is made to write to a mapped area in a TLB segment that is marked invalid |
| 2 | TLBL | TLB Miss exception (load or instruction fetch) | - When an attempt is made to read from an unmapped area in a TLB segment (Uses special TLB miss exception vector) |
| 3 | TLBS | TLB Miss exception (store) | - When an attempt is made to write to an unmapped area in a TLB segment (Uses special TLB miss exception vector) |
| 4 | AdEL | Address Error exception (load or instruction fetch) | - When an attempt is made to read from an address whose boundary alignment is incompatible with the instruction - When an attempt is made to read from an address that is not accessible in the current operating mode |
| 5 | AdES | Address Error exception (store) | - When an attempt is made to write to an address whose boundary alignment is incompatible with the opcode - When an attempt is made to write to an address that is not accessible in the current operating mode |
| 6 | IBE | Bus Error exception (instruction fetch) | |
| 7 | DBE | Bus Error exception (data reference: load or store) | |
| 8 | Sys | Syscall exception | - When a SYSCALL instruction is executed |
| 9 | Bp | Breakpoint exception | - When a BREAK instruction is executed |
| 10 | RI | Reserved instruction exception | - When an attempt is made to execute a reserved/nonexistant command |
| 11 | CpU | Coprocessor unusable exception | - When an attempt is made to use a coprocessor instruction and the corresponding coprocessor is marked unusable (Status CU bit = 0) - When an attempt is made to use a COP0 instruction while operating in user or supervisor mode |
| 12 | Ov | Arithmetic overflow exception | - When an ADD, ADDI, SUB, DADD, DADDI or DSUB instruction results in a 2's complement overflow |
| 13 | Tr | Trap exception | - When a trap instruction results in a TRUE condition |
| 14 | - | (Reserved) | |
| 15 | FPE | Floating-point exception | (Generated by the floating-point coprocessor (COP1); contents of the Floating-Point Control/Status register (FCSR31) indicate the cause of the exception) |
| 16:22 | - | (Reserved) | |
| 23 | WATCH | Watch exception | - When a load/store instruction references the address specified in the WatchLo/WatchHi registers |
| 24:31 | - | (Reserved) | |

**Interrupts**

| **Cause bit** | **Source** |
| --- | --- |
| IP0 | Set by software |
| IP1 | Set by software |
| IP2 | Int0 pin (N64: RCP) |
| IP3 | Int1 pin (N64: Cartridge) |
| IP4 | Int2 pin (N64: Pre-NMI (Reset button)) |
| IP5 | Int3 pin (N64: RDB Read) |
| IP6 | Int4 pin (N64: RDB Write) |
| IP7 | Timer interrupt |

* Software interrupts (IP1:IP0) are generated when software manually sets the Cause register's IP1 or IP0 bit to 1 using an MTC0 instruction.
* The Timer interrupt (IP7) is generated when the Count and Compare registers are equal.
* Interrupts may be masked by setting the respective IM bits of the Status register to 0.

On the Nintendo 64, IP2 represents an RCP (Reality Coprocessor) interrupt. When an RCP interrupt occurs, a flag representing the specific RCP interface may be read from MI\_INTR\_REG (0x04300008).

| **Bit** | **Name** | **Description** |
| --- | --- | --- |
| 0x01 | MI\_INTR\_SP | Signal Processor - Task Done/Task Yield |
| 0x02 | MI\_INTR\_SI | Serial Interface - Controller input available |
| 0x04 | MI\_INTR\_AI | Audio Interface - Audio buffer swap |
| 0x08 | MI\_INTR\_VI | Video Interface - Vertical retrace |
| 0x10 | MI\_INTR\_PI | Peripheral Interface - ROM to RAM DMA done |
| 0x20 | MI\_INTR\_DP | Display Processor - RDP processing done (gDPFullSync) |

RCP-specific interrupts may be enabled or disabled by writing one or more of the following values to MI\_INTR\_MASK\_REG (0x0430000C).

| **Bit** | **Name** | **Description** |
| --- | --- | --- |
| 0x0001 | MI\_INTR\_MASK\_CLR\_SP | Disable SP interrupts |
| 0x0002 | MI\_INTR\_MASK\_SET\_SP | Enable SP interrupts |
| 0x0004 | MI\_INTR\_MASK\_CLR\_SI | Disable SI interrupts |
| 0x0008 | MI\_INTR\_MASK\_SET\_SI | Enable SI interrupts |
| 0x0010 | MI\_INTR\_MASK\_CLR\_AI | Disable AI interrupts |
| 0x0020 | MI\_INTR\_MASK\_SET\_AI | Enable AI interrupts |
| 0x0040 | MI\_INTR\_MASK\_CLR\_VI | Disable VI interrupts |
| 0x0080 | MI\_INTR\_MASK\_SET\_VI | Enable VI interrupts |
| 0x0100 | MI\_INTR\_MASK\_CLR\_PI | Disable PI interrupts |
| 0x0200 | MI\_INTR\_MASK\_SET\_PI | Enable PI interrupts |
| 0x0400 | MI\_INTR\_MASK\_CLR\_DP | Disable DP interrupts |
| 0x0800 | MI\_INTR\_MASK\_SET\_DP | Enable DP interrupts |

**Memory Segments (32-bit)**

| **Virtual address range** | **Segment** | **Description** |
| --- | --- | --- |
| 00000000:7FFFFFFF | useg | TLB mapped to physical memory Accessible in all operating modes |
| 80000000:9FFFFFFF | kseg0 | Cached, translated to physical address by subtracting 80000000 Accessible in kernel mode |
| A0000000:BFFFFFFF | kseg1 | Uncached, translated to physical address by subtracting A0000000 Accessible in kernel mode |
| C0000000:DFFFFFFF | sseg | TLB mapped to physical memory Accessible in kernel and supervisor mode |
| E0000000:FFFFFFFF | kseg3 | TLB mapped to physical memory Accessible in kernel mode |

Note: Commercial Nintendo 64 games operate in kernel mode at all times.

**References**

* <http://hack64.net/docs/VR43XX.pdf>
* <http://datasheets.chipdb.org/MIPS/R4300i_datasheet.pdf>
* <https://level42.ca/projects/ultra64/Documentation/man/pro-man/pro07/index7.7.html>
* <https://level42.ca/projects/ultra64/Documentation/man/pro-man/pro07/index7.8.html>
* <https://level42.ca/projects/ultra64/Documentation/man/n64man/u64/u64.html>
* <https://level42.ca/projects/ultra64/Documentation/man/header/rcp.htm>
* <https://github.com/n64dev/cen64/blob/72c778c3bfb25262498af6a21e8dec828a28be19/vr4300/interface.h#L16>
* <http://ti.ira.uka.de/TI-2/Mips/Befehlssatz.pdf>

[**1)**](https://hack64.net/wiki/doku.php?id=r4300#fnt__1)

“immediate”: 16-bit signed immediate value

[**2)**](https://hack64.net/wiki/doku.php?id=r4300#fnt__2)

“sa”: 5-bit immediate shift amount

[**3)**](https://hack64.net/wiki/doku.php?id=r4300#fnt__3)

“FPU”: Floating-point unit, another name for Coprocessor 1