The MIPS R2000 Instruction Set

Arithmetic and Logical Instructions

In all instructions below, Src2 can either be a register or an immediate value (a 16 bit integer). The immediate forms of the instructions are only included for reference. The assembler will translate the more general form of an instruction (e.g., add) into the immediate form (e.g., addi) if the second argument is constant. Instructions marked with a dagger (†) are pseudoinstructions.

abs Rdest, Rsrc

Put the absolute value of the integer from register Rsrc in register Rdest.

add Rdest, Rsrc1, Src2

Addition (with overflow)

addi Rdest, Rsrc1, Imm

Addition Immediate (with overflow)

addu Rdest, Rsrc1, Src2

Addition (without overflow)

addiu Rdest, Rsrc1, Imm

Addition Immediate (without overflow)

Put the sum of the integers from register Rsrc1 and Src2 (or Imm) into register Rdest.

and Rdest, Rsrc1, Src2

AND

andi Rdest, Rsrc1, Imm

AND Immediate

Put the logical AND of the integers from register Rsrc1 and Src2 (or Imm) into register Rdest.

div Rsrc1, Rsrc2

Divide (with overflow)

divu Rsrc1, Rsrc2

Divide (without overflow)

Divide the contents of the two registers. Leave the quotient in register lo and the remainder in register hi. Note that if an operand is negative, the remainder is unspecified by the MIPS architecture and depends on the conventions of the machine on which SPIM is run.

div Rdest, Rsrc1, Src2

Divide (with overflow) †

divu Rdest, Rsrc1, Src2

Divide (without overflow) †

Put the quotient of the integers from register Rsrc1 and Src2 into register Rdest.

mul Rdest, Rsrc1, Src2

Multiply (without overflow) †

mulo Rdest, Rsrc1, Src2

Multiply (with overflow) †

mulou Rdest, Rsrc1, Src2

Unsigned Multiply (with overflow) †

Put the product of the integers from register Rsrc1 and Src2 into register Rdest.

mult Rsrc1, Rsrc2

Multiply

multu Rsrc1, Rsrc2

Unsigned Multiply

Multiply the contents of the two registers. Leave the low-order word of the product in register lo and the high-word in register hi.
neg Rdest, Rsr
   Negate Value (with overflow) †

negu Rdest, Rsr
   Negate Value (without overflow) †

Put the negative of the integer from register Rsr into register Rdest.

nor Rdest, Rsr1, Src2
   NOR

Put the logical NOR of the integers from register Rsr1 and Src2 into register Rdest.

not Rdest, Rsr
   NOT †

Put the bitwise logical negation of the integer from register Rsr into register Rdest.

or Rdest, Rsr1, Src2
   OR

ori Rdest, Rsr1, Imm
   OR Immediate

Put the logical OR of the integers from register Rsr1 and Src2 (or Imm) into register Rdest.

rem Rdest, Rsr1, Src2
   Remainder †

remu Rdest, Rsr1, Src2
   Unsigned Remainder †

Put the remainder from dividing the integer in register Rsr1 by the integer in Src2 into register Rdest. Note that if an operand is negative, the remainder is unspecified by the MIPS architecture and depends on the conventions of the machine on which SPIM is run.

rol Rdest, Rsr1, Src2
   Rotate Left †

ror Rdest, Rsr1, Src2
   Rotate Right †

Rotate the contents of register Rsr1 left (right) by the distance indicated by Src2 and put the result in register Rdest.

sll Rdest, Rsr1, Src2
   Shift Left Logical

sllv Rdest, Rsr1, Src2
   Shift Left Logical Variable

sra Rdest, Rsr1, Src2
   Shift Right Arithmetic

srav Rdest, Rsr1, Src2
   Shift Right Arithmetic Variable

srl Rdest, Rsr1, Src2
   Shift Right Logical

sr lv Rdest, Rsr1, Src2
   Shift Right Logical Variable

Shift the contents of register Rsr1 left (right) by the distance indicated by Src2 (Rs r2) and put the result in register Rdest.

sub Rdest, Rsr1, Src2
   Subtract (with overflow)

subu Rdest, Rsr1, Src2
   Subtract (without overflow)

Put the difference of the integers from register Rsr1 and Src2 into register Rdest.

xor Rdest, Rsr1, Src2
   XOR

xori Rdest, Rsr1, Imm
   XOR Immediate

Put the logical XOR of the integers from register Rsr1 and Src2 (or Imm) into register Rdest.
Constant-Manipulating Instructions

```plaintext
li Rdest, Imm 
Move the immediate imm into register Rdest.

lui Rdest, Imm 
Load the lower halfword of the immediate imm into the upper halfword of register Rdest. The lower bits of the register are set to 0.
```

Comparison Instructions

In all instructions below, Src2 can either be a register or an immediate value (a 16 bit integer).

```plaintext
seq Rdest, Rsrc1, Src2 
Set register Rdest to 1 if register Rsrc1 equals Src2 and to be 0 otherwise.

sge Rdest, Rsrc1, Src2 
Set Greater Than Equal

sgeu Rdest, Rsrc1, Src2 
Set Greater Than Equal Unsigned

sgt Rdest, Rsrc1, Src2 
Set Greater Than

sgtu Rdest, Rsrc1, Src2 
Set Greater Than Unsigned

sle Rdest, Rsrc1, Src2 
Set Less Than Equal

sleu Rdest, Rsrc1, Src2 
Set Less Than Equal Unsigned

slt Rdest, Rsrc1, Src2 
Set Less Than

slti Rdest, Rsrc1, Imm 
Set Less Than Immediate

sltu Rdest, Rsrc1, Src2 
Set Less Than Unsigned

sltiu Rdest, Rsrc1, Imm 
Set Less Than Unsigned Immediate

sne Rdest, Rsrc1, Src2 
Set Not Equal
```

Branch and Jump Instructions

In all instructions below, Src2 can either be a register or an immediate value (integer). Branch instructions use a signed 16-bit offset field; hence they can jump $2^{15} - 1$ instructions (not bytes) forward or $2^{15}$ instructions backwards. The jump instruction contains a 26 bit address field.
b label
Unconditionally branch to the instruction at the label.

bal label
Unconditionally branch to the instruction at the label. Save the address of the next instruction in register $ra.

bczt label
Branch Coprocessor z True

bczf label
Branch Coprocessor z False
Conditionally branch to the instruction at the label if coprocessor z condition flag is true (false).

beg Rsrc1, Src2, label
Conditionally branch to the instruction at the label if the contents of register Rsrc1 equals Src2.

begz Rsrc, label
Conditionally branch to the instruction at the label if the contents of Rsrc equals 0.

bge Rsrc1, Src2, label
Branch on Greater Than Equal

bgeu Rsrc1, Src2, label
Branch on GTE Unsigned
Conditionally branch to the instruction at the label if the contents of register Rsrc1 are greater than or equal to Src2.

bgez Rsrc, label
Branch on Greater Than Equal Zero
Conditionally branch to the instruction at the label if the contents of Rsrc are greater than or equal to 0.

bgezal Rsrc, label
Branch on Greater Than Equal Zero And Link
Conditionally branch to the instruction at the label if the contents of Rsrc are greater than or equal to 0. Save the address of the next instruction in register $ra.

bgt Rsrc1, Src2, label
Branch on Greater Than

bgtu Rsrc1, Src2, label
Branch on Greater Than Unsigned
Conditionally branch to the instruction at the label if the contents of register Rsrc1 are greater than Src2.

bgtz Rsrc, label
Branch on Greater Than Zero
Conditionally branch to the instruction at the label if the contents of Rsrc are greater than 0.

ble Rsrc1, Src2, label
Branch on Less Than Equal

bleu Rsrc1, Src2, label
Branch on LTE Unsigned
Conditionally branch to the instruction at the label if the contents of register Rsrc1 are less than or equal to Src2.
blez Rsrc, label  
Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are less than or equal to 0.

bgezal Rsrc, label  
\textit{Branch on Greater Than Equal Zero And Link}

bltzal Rsrc, label  
\textit{Branch on Less Than And Link}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are greater or equal to 0 or less than 0, respectively. Save the \texttt{address} of the next instruction in register 31.

blt Rsrc1, Src2, label  
\textit{Branch on Less Than} \footnote{†}

bltu Rsrc1, Src2, label  
\textit{Branch on Less Than Unsigned} \footnote{†}

Conditionally branch to the instruction at the label if the contents of register \texttt{Rsrc1} are less than \texttt{Src2}.

bltz Rsrc, label  
\textit{Branch on Less Than Zero}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are less than 0.

bne Rsrc1, Src2, label  
\textit{Branch on Not Equal}

Conditionally branch to the instruction at the label if the contents of register \texttt{Rsrc1} are not equal to \texttt{Src2}.

bnez Rsrc, label  
\textit{Branch on Not Equal Zero} \footnote{†}

Conditionally branch to the instruction at the label if the contents of \texttt{Rsrc} are not equal to 0.

j label  
\textit{Jump}

Unconditionally jump to the instruction at the label.

jal label  
\textit{Jump and Link}

jalr Rsrc  
\textit{Jump and Link Register}

Unconditionally jump to the instruction at the label or whose \texttt{address} is in register \texttt{Rsrc}. Save the \texttt{address} of the next instruction in register \texttt{$ra$}.

jr Rsrc  
\textit{Jump Register}

Unconditionally jump to the instruction whose \texttt{address} is in register \texttt{Rsrc}.

\textbf{Load Instructions}

la Rdest, address  
\textit{Load Address} \footnote{†}

Load computed \texttt{address}, not the contents of the location, into register \texttt{Rdest}. 
Load Byte
Load the byte at address into register Rdest. The byte is sign-extended by the 1b, but not the 1bu, instruction.

Load Double-Word †
Load the 64-bit quantity at address into registers Rdest and Rdest + 1.

Load Word †
Load the 32-bit quantity (word) at address into register Rdest.

Load Word Coprocessor
Load the word at address into register Rdest of coprocessor z (0-3).

Load Word Left
Load the left (right) bytes from the word at the possibly-unaligned address into register Rdest.

Unaligned Load Halfword †
Load the 16-bit quantity (halfword) at the possibly-unaligned address into register Rdest. The halfword is sign-extended by the ulh, but not the ulhu, instruction

Unaligned Load Word †
Load the 32-bit quantity (word) at the possibly-unaligned address into register Rdest.

Store Byte
Store the low byte from register Rsrck at address.

Store Double-Word †
Store the 64-bit quantity in registers Rsrck and Rsrck + 1 at address.

Store Halfword
Store the low halfword from register Rsrck at address.
sw Rsrc, address  
Store the word from register Rsrc at address.

swcz Rsrc, address  
Store the word from register Rsrc of coprocessor z at address.

swl Rsrc, address  
Store the left (right) bytes from register Rsrc at the possibly-unaligned address.

swr Rsrc, address  
Store Word Right

ush Rsrc, address  
Unaligned Store Halfword

usw Rsrc, address  
Unaligned Store Word

Data Movement Instructions

move Rdest, Rsrc  
Move the contents of Rsrc to Rdest.

The multiply and divide unit produces its result in two additional registers, hi and lo. These instructions move values to and from these registers. The multiply, divide, and remainder instructions described above are pseudoinstructions that make it appear as if this unit operates on the general registers and detect error conditions such as divide by zero or overflow.

mfhi Rdest  
Move From hi

mflo Rdest  
Move From lo

mthi Rsns  
Move To hi

mtlo Rsns  
Move To lo

Coprocessors have their own register sets. These instructions move values between these registers and the CPU's registers.

mfcz Rdest, CPsrc  
Move From Coprocessor z

Move the contents of coprocessor z's register CPsrc to CPU register Rdest.
mfc1.d Rdest, FRsrc1
Move Double From Coprocessor 1 †
Move the contents of floating point registers FRsrc1 and FRsrc1 + 1 to CPU registers Rdest and Rdest + 1.

mtc1.d Rs, FRdest1
Move Double To Coprocessor 1 †
Move the contents of CPU registers Rs and Rs + 1 to floating point registers FRdest1 and FRdest1 + 1.

mtcz Rs, CPdest
Move To Coprocessor z
Move the contents of CPU register Rs to coprocessor z's register CPdest.

Floating Point Instructions

The MIPS has a floating point coprocessor (numbered 1) that operates on single precision (32-bit) and double precision (64-bit) floating point numbers. This coprocessor has its own registers, which are numbered $f0 - f31. Because these registers are only 32-bits wide, two of them are required to hold doubles. To simplify matters, floating point operations only use even-numbered registers - including instructions that operate on single floats.

Values are moved in or out of these registers a word (32-bits) at a time by lwcl, swcl, mtc1, and mfc1 instructions described above or by the l.s, l.d, s.s, and s.d pseudoinstructions described below. The flag set by floating point comparison operations is read by the CPU with its bc1t and bc1f instructions. In all instructions below, FRdest, FRsrc1, FRsrc2, and FRsrc are floating point registers (e.g., $f2).

abs.d FRdest, FRsrc
Floating Point Absolute Value Double
Compute the absolute value of the floating point double in register FRsrc and put it in register FRdest.

abs.s FRdest, FRsrc
Floating Point Absolute Value Single

add.d FRdest, FRsrc1, FRsrc2
Floating Point Addition Double
Compute the sum of the floating point doubles in registers FRsrc1 and FRsrc2 and put it in register FRdest.

add.s FRdest, FRsrc1, FRsrc2
Floating Point Addition Single

compare.d FRsrc1, FRsrc2
Compare Equal Double
Compare the floating point double in register FRsrc1 against the one in FRsrc2 and set the floating point condition flag true if they are equal.

c.eq.s FRsrc1, FRsrc2
Compare Equal Single
c.le.d FRsrc1, FRsrc2  
**Compare Less Than Equal Double (w/ exept.)**

Compare the floating point double (single) in register FRsrc1 against the one in FRsrc2 and set the floating point condition flag true if the first is less than or equal to the second. Cause exeption 14 if FRsrc1 or FRsrc2 contains an invalid floating point number.

---

c.le.s FRsrc1, FRsrc2  
**Compare Less Than Equal Single (w/ exept.)**

---

c.lt.d FRsrc1, FRsrc2  
**Compare Less Than Double (w/ exept.)**

---

c.lt.s FRsrc1, FRsrc2  
**Compare Less Than Single (w/ exept.)**

---

c.nge.d FRsrc1, FRsrc2  
**Compare Not Greater Than Equal Double (w/ exept.)**

---

c.nge.s FRsrc1, FRsrc2  
**Compare Not Greater Than Equal Single (w/ exept.)**

---

c.ngle.d FRsrc1, FRsrc2  
**Compare Not Greater or Less Than Equal Double (w/ exept.)**

---

c.ngle.s FRsrc1, FRsrc2  
**Compare Not Greater or Less Than Equal Single (w/ exept.)**

---

c.ngt.d FRsrc1, FRsrc2  
**Compare Not Greater Than Double (w/ exept.)**

---

c.ngt.s FRsrc1, FRsrc2  
**Compare Not Greater Than Single (w/ exept.)**

---

c.ngt.d FRsrc1, FRsrc2  
**Compare Not Greater Than Double (w/ exept.)**

---

c.ngt.s FRsrc1, FRsrc2  
**Compare Not Greater Than Single (w/ exept.)**

---
c.ole.d FRsrc1, FRsrc2  \textit{Compare Less Than Equal Double}
c.ole.s FRsrc1, FRsrc2  \textit{Compare Less Than Equal Single}

Compare the floating point double (single) in register \texttt{FRsrc1} against the one in \texttt{FRsrc2} and set the floating point condition flag true if the first is less than or equal to the second.

c.olt.d FRsrc1, FRsrc2  \textit{Compare Less Than Double}
c.olt.s FRsrc1, FRsrc2  \textit{Compare Less Than Single}

Compare the floating point double (single) in register \texttt{FRsrc1} against the one in \texttt{FRsrc2} and set the condition flag true if the first is less than the second.

c.seq.d FRsrc1, FRsrc2  \textit{Compare Equal Double (w/ exept.)}
c.seq.s FRsrc1, FRsrc2  \textit{Compare Equal Single (w/ exept.)}

Compare the floating point double (single) in register \texttt{FRsrc1} against the one in \texttt{FRsrc2} and set the floating point condition flag true if they are equal. Cause exception 14 if \texttt{FRsrc1} or \texttt{FRsrc2} contains an invalid floating point number.

c.sf.d FRsrc1, FRsrc2  \textit{Set False Double (w/ exept.)}
c.sf.s FRsrc1, FRsrc2  \textit{Set False Single (w/ exept.)}

Set the condition flag false. Cause exception 14 if \texttt{FRsrc1} or \texttt{FRsrc2} contains an invalid floating point number.

c.f.d FRsrc1, FRsrc2  \textit{Set False Double}
c.f.s FRsrc1, FRsrc2  \textit{Set False Single}

Set the condition flag false. The contents of \texttt{FRsrc1} or \texttt{FRsrc2} are meaningless.

c.ueq.d FRsrc1, FRsrc2  \textit{Compare Unordered or Equal Double}
c.ueq.s FRsrc1, FRsrc2  \textit{Compare Unordered or Equal Single}

Compare the floating point double (single) in register \texttt{FRsrc1} against the one in \texttt{FRsrc2} and set the floating point condition flag true if they are equal or if \texttt{FRsrc1} or \texttt{FRsrc2} contains an invalid floating point number.

c.ule.d FRsrc1, FRsrc2  \textit{Compare Unordered or Less Than Equal Double}
c.ule.s FRsrc1, FRsrc2  \textit{Compare Unordered or Less Than Equal Single}

Compare the floating point double (single) in register \texttt{FRsrc1} against the one in \texttt{FRsrc2} and set the floating point condition flag true if the first is less than or equal to the second or if \texttt{FRsrc1} or \texttt{FRsrc2} contains an invalid floating point number.

c.ult.d FRsrc1, FRsrc2  \textit{Compare Unordered or Less Than Double}
c.ult.s FRsrc1, FRsrc2  \textit{Compare Unordered or Less Than Single}

Compare the floating point double (single) in register \texttt{FRsrc1} against the one in \texttt{FRsrc2} and set the condition flag true if the first is less than the second or if \texttt{FRsrc1} or \texttt{FRsrc2} contains an invalid floating point number.
c.un.d FRsrc1, FRsrc2  
\textit{Compare Unordered Double}

c.un.s FRsrc1, FRsrc2  
\textit{Compare Unordered Single}

Set the condition flag true if FRsrc1 or FRsrc2 contains an invalid floating point number.

cvt.d.s FRdest, FRsrc  
\textit{Convert Single to Double}

cvt.d.w FRdest, FRsrc  
\textit{Convert Integer to Double}

Convert the single precision floating point number or integer in register FRsrc to a double precision number and put it in register FRdest.

cvt.s.d FRdest, FRsrc  
\textit{Convert Double to Single}

cvt.s.w FRdest, FRsrc  
\textit{Convert Integer to Single}

Convert the double precision floating point number or integer in register FRsrc to a single precision number and put it in register FRdest.

cvt.w.d FRdest, FRsrc  
\textit{Convert Double to Integer}

cvt.w.s FRdest, FRsrc  
\textit{Convert Single to Integer}

Convert the double or single precision floating point number in register FRsrc to an integer and put it in register FRdest.

div.d FRdest, FRsrc1, FRsrc2  
\textit{Floating Point Divide Double}

div.s FRdest, FRsrc1, FRsrc2  
\textit{Floating Point Divide Single}

Compute the quotient of the floating float doubles (singles) in registers FRsrc1 and FRsrc2 and put it in register FRdest.

l.d FRdest, address  
\textit{Load Floating Point Double} \dagger

l.s FRdest, address  
\textit{Load Floating Point Single} \dagger

Load the floating float double (single) at address into register FRdest.

li.d FRdest, Imm  
\textit{Load Floating Point Double Immediate} \dagger

li.s FRdest, Imm  
\textit{Load Floating Point Single Immediate} \dagger

Load the floating float double (single) immediate Imm (e.g. 3.141) into register FRdest.

mov.d FRdest, FRsrc  
\textit{Move Floating Point Double}

mov.s FRdest, FRsrc  
\textit{Move Floating Point Single}

Move the floating float double (single) from register FRsrc to register FRdest.

mul.d FRdest, FRsrc1, FRsrc2  
\textit{Floating Point Multiply Double}

mul.s FRdest, FRsrc1, FRsrc2  
\textit{Floating Point Multiply Single}

Compute the product of the floating float doubles (singles) in registers FRsrc1 and FRsrc2 and put it in register FRdest.
neg.d FRdest, FRsrc  \hspace{1cm} \textit{Negate Double}

neg.s FRdest, FRsrc  \hspace{1cm} \textit{Negate Single}

Negate the floating point double (single) in register FRsrc and put it in register FRdest.

s.d FRdest, address  \hspace{1cm} \textit{Store Floating Point Double} †

s.s FRdest, address  \hspace{1cm} \textit{Store Floating Point Single} †

Store the floating float double (single) in register FRdest at address.

sub.d FRdest, FRsrc1, FRsrc2  \hspace{1cm} \textit{Floating Point Subtract Double}

sub.s FRdest, FRsrc1, FRsrc2  \hspace{1cm} \textit{Floating Point Subtract Single}

Compute the difference of the floating float doubles (singles) in registers FRsrc1 and FRsrc2 and put it in register FRdest.

\section*{Exception and Trap Instructions}

rfe  \hspace{1cm} \textit{Return From Exception}

Restore the Status register.

syscall  \hspace{1cm} \textit{System Call}

Register $v0 contains the number of the system call provided by SPIM.

break n  \hspace{1cm} \textit{Break}

Cause exception n. Exception 1 is reserved for the debugger.

nop  \hspace{1cm} \textit{No operation} †

Do nothing.