

## MIPS R2000 Assembly Language

Arithmetic and Logical Instructions																		
Instruction	Format					Comment												
<b>Absolute value</b> abs rdest, rsrc	pseudoinstruction					Put the absolute value of register rsrc in register rdest												
<b>Addition (with overflow)</b> add rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x20</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x20	6	5	5	5	5	6					
0	rs	rt	rd	0	0x20													
6	5	5	5	5	6													
<b>Addition (without overflow)</b> addu rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x21</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x21	6	5	5	5	5	6	Put the sum of the register rs and rt into register rd				
0	rs	rt	rd	0	0x21													
6	5	5	5	5	6													
<b>Addition immediate (with overflow)</b> addi rt, rs, imm	<table><tr><td>8</td><td>rs</td><td>rt</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	8	rs	rt	imm			6	5	5	16							
8	rs	rt	imm															
6	5	5	16															
<b>Addition immediate (without overflow)</b> addiu rt, rs, imm	<table><tr><td>9</td><td>rs</td><td>rt</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	9	rs	rt	imm			6	5	5	16			Put the sum of register rs and the sign-extended immediate into register rt				
9	rs	rt	imm															
6	5	5	16															
<b>AND</b> and rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x24</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x24	6	5	5	5	5	6	Put the logical AND of register rs and rt into register rd				
0	rs	rt	rd	0	0x24													
6	5	5	5	5	6													
<b>AND immediate</b> andi rt, rs, imm	<table><tr><td>0xc</td><td>rs</td><td>rt</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	0xc	rs	rt	imm			6	5	5	16			Put the logical AND of register rs and the zero-extended immediate into register rt				
0xc	rs	rt	imm															
6	5	5	16															
<b>Divide (with overflow)</b> div rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td colspan="2">0</td><td>0x1a</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="2">10</td><td>6</td></tr></table>	0	rs	rt	0		0x1a	6	5	5	10		6					
0	rs	rt	0		0x1a													
6	5	5	10		6													
<b>Divide (without overflow)</b> divu rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td colspan="2">0</td><td>0x1b</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="2">10</td><td>6</td></tr></table>	0	rs	rt	0		0x1b	6	5	5	10		6	Divide register rs by register rt. Leave the quotient in register lo and the remainder in register hi. If an operand is negative, the remainder is unspecified by the MIPS architecture and depends on the convention of the machine on which SPIM is run.				
0	rs	rt	0		0x1b													
6	5	5	10		6													
<b>Divide (with overflow)</b> div rdest, rsrc1, src2	pseudoinstruction																	
<b>Divide (without overflow)</b> div rdest, rsrc1, src2	pseudoinstruction					Put the quotient of register rsrc1 and src2 into register rdest.												
<b>Multiply</b> mult rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td colspan="2">0</td><td>0x18</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="2">10</td><td>6</td></tr></table>	0	rs	rt	0		0x18	6	5	5	10		6					
0	rs	rt	0		0x18													
6	5	5	10		6													
<b>Unsigned multiply</b> multu rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td colspan="2">0</td><td>0x19</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="2">10</td><td>6</td></tr></table>	0	rs	rt	0		0x19	6	5	5	10		6	Multiply registers rs and rt. Leave the low-order word of the product in register lo and the high-order word in register hi				
0	rs	rt	0		0x19													
6	5	5	10		6													

<b>Multiply (without overflow)</b> mul rdest, rsrc1, src2	pseudoinstruction													
<b>Multiply (with overflow)</b> mulo rdest, rsrc1, src2	pseudoinstruction													
<b>Unsigned multiply (with overflow)</b> mulou rdest, rsrc1, src2	pseudoinstruction	Put the product of register rsrc1 and src2 into register rdest.												
<b>Negate value (with overflow)</b> neg rdest, rsrc	pseudoinstruction													
<b>Negate value (without overflow)</b> negu rdest, rsrc	pseudoinstruction	Put the negative of register rsrc into register rd.												
<b>NOR</b> nor rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x27</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x27	6	5	5	5	5	6	Put the logical NOR of registers rs and rt into register rd
0	rs	rt	rd	0	0x27									
6	5	5	5	5	6									
<b>NOT</b> not rdest, rsrc	pseudoinstruction	Put the bitwise logical negation of register rsrc into register rdest.												
<b>OR</b> or rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x25</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x25	6	5	5	5	5	6	Put the logical OR of registers rs and rt into register rd.
0	rs	rt	rd	0	0x25									
6	5	5	5	5	6									
<b>OR immediate</b> ori rt, rs, imm	<table><tr><td>0xd</td><td>rs</td><td>rt</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	0xd	rs	rt	imm			6	5	5	16			Put the logical OR of register rs and the zero-extended immediate into register rt.
0xd	rs	rt	imm											
6	5	5	16											
<b>Remainder</b> rem rdest, rsrc1, rsrc2	pseudoinstruction													
<b>Unsigned remainder</b> rem rdest, rsrc1, rsrc2	pseudoinstruction	Put the remainder of register rsrc1 divided by register rsrc2 into register rdest. If an operand is negative, the remainder is unspecified by the MIPS architecture and depends on the convention of the machine on which SPIM is run.												
<b>Shift left logical</b> sll rd, rt, shamt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>shamt</td><td>0</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	shamt	0	6	5	5	5	5	6	
0	rs	rt	rd	shamt	0									
6	5	5	5	5	6									
<b>Shift left logical variable</b> sllv rd, rt, rs	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>4</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	4	6	5	5	5	5	6	
0	rs	rt	rd	0	4									
6	5	5	5	5	6									
<b>Shift right arithmetic</b> sra rd, rt, shamt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>shamt</td><td>3</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	shamt	3	6	5	5	5	5	6	
0	rs	rt	rd	shamt	3									
6	5	5	5	5	6									
<b>Shift right arithmetic variable</b> srav rd, rt, rs	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>7</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	7	6	5	5	5	5	6	
0	rs	rt	rd	0	7									
6	5	5	5	5	6									
<b>Shift right logical</b> srl rd, rt, shamt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>shamt</td><td>2</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	shamt	2	6	5	5	5	5	6	
0	rs	rt	rd	shamt	2									
6	5	5	5	5	6									

<b>Shift right logical variable</b> srlv rd, rt, rs	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>6</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	6	6	5	5	5	5	6	Shift register rt left (right) by the distance indicated by the immediate shamt or the register rs and put the result into register rd. Argument rs is ignored for sll, sra, and srl.
0	rs	rt	rd	0	6									
6	5	5	5	5	6									
<b>Rotate left</b> rol rdest, rsrc1, rsrc2	pseudo-instruction													
<b>Rotate right</b> ror rdest, rsrc1, rsrc2	pseudo-instruction	Rotate register rsrc1 left (right) by the distance indicated by rsrc2 and put the result into register rdest.												
<b>Subtract (with overflow)</b> sub rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x22</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x22	6	5	5	5	5	6	
0	rs	rt	rd	0	0x22									
6	5	5	5	5	6									
<b>Subtract (without overflow)</b> subu rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x23</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x23	6	5	5	5	5	6	Put the difference of registers rs and rt into register rd.
0	rs	rt	rd	0	0x23									
6	5	5	5	5	6									
<b>Exclusive OR</b> xor rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x26</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x26	6	5	5	5	5	6	Put the logical XOR of registers rs and rt into register rd.
0	rs	rt	rd	0	0x26									
6	5	5	5	5	6									
<b>XOR immediate</b> xori rt, rs, imm	<table><tr><td>0xe</td><td>rs</td><td>rt</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	0xe	rs	rt	imm			6	5	5	16			Put the logical XOR of register rs and the zero-extended immediate into register rt.
0xe	rs	rt	imm											
6	5	5	16											
<b>Constant-Manipulating Instructions</b>														
<b>Load upper immediate</b> lui rt, imm	<table><tr><td>0xf</td><td>0</td><td>rt</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	0xf	0	rt	imm			6	5	5	16			Load the lower halfword of the immediate imm into the upper halfword of register rt. The lower bits of the register are set to 0.
0xf	0	rt	imm											
6	5	5	16											
<b>Load immediate</b> li rdest, imm	pseudoinstruction	Move the immediate imm into register rdest.												
<b>Comparison instructions</b>														
<b>Set less than</b> slt rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x2a</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x2a	6	5	5	5	5	6	
0	rs	rt	rd	0	0x2a									
6	5	5	5	5	6									
<b>Set less than unsigned</b> sltu rd, rs, rt	<table><tr><td>0</td><td>rs</td><td>rt</td><td>rd</td><td>0</td><td>0x2b</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	rt	rd	0	0x2b	6	5	5	5	5	6	Set register rd to 1 if register rs is less than rt, and to 0 otherwise.
0	rs	rt	rd	0	0x2b									
6	5	5	5	5	6									
<b>Set less than immediate</b> slti rd, rs, imm	<table><tr><td>0xa</td><td>rs</td><td>rd</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	0xa	rs	rd	imm			6	5	5	16			
0xa	rs	rd	imm											
6	5	5	16											
<b>Set less than unsigned immediate</b> sltiu rd, rs, imm	<table><tr><td>0xb</td><td>0</td><td>rt</td><td colspan="3">imm</td></tr><tr><td>6</td><td>5</td><td>5</td><td colspan="3">16</td></tr></table>	0xb	0	rt	imm			6	5	5	16			Set register rd to 1 if register rs is less than the sign-extended immediate, and to o otherwise.
0xb	0	rt	imm											
6	5	5	16											
<b>Set equal</b> seq rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 equals rsrc2, and to 0 otherwise.												

<b>Set greater than equal</b> sge rdest, rsrc1, rsrc2	pseudoinstruction									
<b>Set greater than equal unsigned</b> sgeu rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 is greater than or equal to register rsrc2, and to 0 otherwise.								
<b>Set greater than</b> sgt rdest, rsrc1, rsrc2	pseudoinstruction									
<b>Set greater than unsigned</b> sgtu rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 is greater than register rsrc2, and to 0 otherwise.								
<b>Set less than equal</b> sle rdest, rsrc1, rsrc2	pseudoinstruction									
<b>Set less than equal unsigned</b> sleu rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 is less than or equal to rsrc2, and to 0 otherwise.								
<b>Branch instructions</b>										
<b>Branch instruction</b> b label	pseudoinstruction	Unconditionally branch to the instruction at the label.								
<b>Branch coprocessor z true</b> bczt label	<table><tr><td>0x1z</td><td>8</td><td>1</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x1z	8	1	offset	6	5	5	16	
0x1z	8	1	offset							
6	5	5	16							
<b>Branch coprocessor z false</b> bczf label	<table><tr><td>0x1z</td><td>8</td><td>0</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x1z	8	0	offset	6	5	5	16	Conditionally branch the number of instructions specified by the offset if z’s condition flag is true (false). z is 0, 1, 2, or 3. The floating point unit is z = 1.
0x1z	8	0	offset							
6	5	5	16							
<b>Branch on equal</b> beq rs, rt, label	<table><tr><td>4</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	4	rs	rt	offset	6	5	5	16	Conditionally branch the number of instructions specified by the offset if register rs equals rt.
4	rs	rt	offset							
6	5	5	16							
<b>Branch on greater than equal zero</b> bgez rs, label	<table><tr><td>1</td><td>rs</td><td>1</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	1	rs	1	offset	6	5	5	16	Conditionally branch the number of instructions specified by the offset if register rs is greater than or equal to 0.
1	rs	1	offset							
6	5	5	16							
<b>Branch on greater than equal zero and link</b> bgezal rs, label	<table><tr><td>1</td><td>rs</td><td>0x11</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	1	rs	0x11	offset	6	5	5	16	Conditionally branch the number of instructions specified by the offset if register rs is greater than or equal to 0. Save the address of the next instruction in register 31.
1	rs	0x11	offset							
6	5	5	16							
<b>Branch on greater than zero</b> bgtz rs, label	<table><tr><td>7</td><td>rs</td><td>0</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	7	rs	0	offset	6	5	5	16	Conditionally branch the instructions specified by the offset if register rs is greater than 0.
7	rs	0	offset							
6	5	5	16							

<b>Branch on less than equal zero</b> blez rs, label	<table><tr><td>6</td><td>rs</td><td>0</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	6	rs	0	offset	6	5	5	16	Conditionally branch the instructions specified by the offset if register rs is less than or equal to 0.
6	rs	0	offset							
6	5	5	16							
<b>Branch on less than zero and link</b> bltzal rs, label	<table><tr><td>1</td><td>rs</td><td>0x10</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	1	rs	0x10	offset	6	5	5	16	Conditionally branch the instructions specified by the offset if register rs is less than 0. Save the address of the next instruction in register 31.
1	rs	0x10	offset							
6	5	5	16							
<b>Branch on less than zero</b> bltz rs, label	<table><tr><td>1</td><td>rs</td><td>0</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	1	rs	0	offset	6	5	5	16	Conditionally branch the instructions specified by the offset if register rs is less than 0.
1	rs	0	offset							
6	5	5	16							
<b>Branch on not equal</b> bne rs, rt, label	<table><tr><td>5</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	5	rs	rt	offset	6	5	5	16	Conditionally branch the instructions specified by the offset if register rs is not equal to rt.
5	rs	rt	offset							
6	5	5	16							
<b>Branch on equal zero</b> beqz rsrc, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc equals 0.								
<b>Branch on greater than equal</b> bge rsrc1, rsrc2, label	pseudoinstruction									
<b>Branch on greater than equal unsigned</b> bgeu rsrc1, rsrc2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is greater than or equal to rsrc2.								
<b>Branch on greater than</b> bgt rsrc1, src2, label	pseudoinstruction									
<b>Branch on greater than unsigned</b> bgtu rsrc1, src2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is greater than src2.								
<b>Branch on less than equal</b> ble rsrc1, src2, label	pseudoinstruction									
<b>Branch on less than equal unsigned</b> bleu rsrc1, src2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is less than or equal to src2.								
<b>Branch on less than</b> blt rsrc1, src2, label	pseudoinstruction									
<b>Branch on less than unsigned</b> bltu rsrc1, src2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is less than src2.								

<b>Branch on not equal zero</b> bnez rsrc, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc is not equal to zero												
<b>Jump instructions</b>														
<b>Jump</b> j target	<table><tr><td>2</td><td>target</td></tr><tr><td>6</td><td>26</td></tr></table>	2	target	6	26	Unconditionally jump to the instruction at target.								
2	target													
6	26													
<b>Jump and link</b> jal target	<table><tr><td>3</td><td>target</td></tr><tr><td>6</td><td>26</td></tr></table>	3	target	6	26	Unconditionally jump to the instruction at target. Save the address of the next instruction in register \$ra.								
3	target													
6	26													
<b>Jump and link register</b> jalr rs, rd	<table><tr><td>0</td><td>rs</td><td>0</td><td>rd</td><td>0</td><td>9</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	0	rd	0	9	6	5	5	5	5	6	Unconditionally jump to the instruction whose address is in register rs. Save the address of the next instruction in register rd (which defaults to 31).
0	rs	0	rd	0	9									
6	5	5	5	5	6									
<b>Jump register</b> jr rs	<table><tr><td>0</td><td>rs</td><td>0</td><td>0</td><td>0</td><td>8</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	0	0	0	8	6	5	5	5	5	6	Unconditionally jump to the instruction whose address is in register rs.
0	rs	0	0	0	8									
6	5	5	5	5	6									
<b>Load instructions</b>														
<b>Load rdest, address</b> la rdest, address	pseudoinstruction	Load computed address – not the contents of the location – into register rd.												
<b>Load byte</b> lb rt, address	<table><tr><td>0x20</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x20	rs	rt	offset	6	5	5	16					
0x20	rs	rt	offset											
6	5	5	16											
<b>Load unsigned byte</b> lbu rt, address	<table><tr><td>0x24</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x24	rs	rt	offset	6	5	5	16	Load the byte at address into register rt. The byt is sign-extended by lb, but not by lbu.				
0x24	rs	rt	offset											
6	5	5	16											
<b>Load halfword</b> lh rt address	<table><tr><td>0x21</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x21	rs	rt	offset	6	5	5	16					
0x21	rs	rt	offset											
6	5	5	16											
<b>Load unsigned halfword</b> lhu rt, address	<table><tr><td>0x25</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x25	rs	rt	offset	6	5	5	16	Load the byte at address into register rt. The byt is sign-extended by lh, but not by lhu.				
0x25	rs	rt	offset											
6	5	5	16											
<b>Load word</b> lw rt, address	<table><tr><td>0x23</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x23	rs	rt	offset	6	5	5	16	Load 32-bit word at address into register rt.				
0x23	rs	rt	offset											
6	5	5	16											
<b>Load word coprocessor</b> lwcx rt, address	<table><tr><td>0x3z</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x3z	rs	rt	offset	6	5	5	16	Load the word at address into register rt of coprocessor z (0-3). The FP unit is z = 1.				
0x3z	rs	rt	offset											
6	5	5	16											
<b>Load word left</b> lwl rt, address	<table><tr><td>0x22</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x22	rs	rt	offset	6	5	5	16					
0x22	rs	rt	offset											
6	5	5	16											

<b>Load word right</b> lwr rt, address	<table><tr><td>0x26</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x26	rs	rt	offset	6	5	5	16	Load the left (right) bytes from the word at the possibly unaligned address into register rt.
0x26	rs	rt	offset							
6	5	5	16							
<b>Load doubleword</b> ld rdest, address	pseudoinstruction	Load the 64-bit double word at address into registers rdest and rest + 1.								
<b>Unaligned load halfword</b> ulh rdest, address	pseudoinstruction									
<b>Unaligned load halfword unsigned</b> ulhu rdest, address	pseudoinstruction	Load the 16-bit halfword at the possibly unaligned address into register rdest. The halfword is sign-extended by ulh, but not ulhu.								
<b>Unaligned load word</b> ulw rdest, address	pseudoinstruction	Load the 32-bit word at the possibly unaligned address into register rdest.								
<b>Store instructions</b>										
<b>Store byte</b> sb rt, address	<table><tr><td>0x28</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x28	rs	rt	offset	6	5	5	16	Store the low byte from register rt at address.
0x28	rs	rt	offset							
6	5	5	16							
<b>Store halfword</b> sh rt, address	<table><tr><td>0x29</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x29	rs	rt	offset	6	5	5	16	Store the low halfword from register rt at address.
0x29	rs	rt	offset							
6	5	5	16							
<b>Store word</b> sw rt, address	<table><tr><td>0x2b</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x2b	rs	rt	offset	6	5	5	16	Store the word from register rt at address.
0x2b	rs	rt	offset							
6	5	5	16							
<b>Store word coprocessor</b> swcz rt, address	<table><tr><td>0x2z</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x2z	rs	rt	offset	6	5	5	16	Store the word from register rt of coprocessor z at address. The FP unit is z=1.
0x2z	rs	rt	offset							
6	5	5	16							
<b>Store word left</b> swl rt, address	<table><tr><td>0x2a</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x2a	rs	rt	offset	6	5	5	16	
0x2a	rs	rt	offset							
6	5	5	16							
<b>Store word right</b> swr rt, address	<table><tr><td>0x2e</td><td>rs</td><td>rt</td><td>offset</td></tr><tr><td>6</td><td>5</td><td>5</td><td>16</td></tr></table>	0x2e	rs	rt	offset	6	5	5	16	Store the left (right) bytes from register rt at the possibly unaligned address.
0x2e	rs	rt	offset							
6	5	5	16							
<b>Store doubleword</b> sd rsrc, address	pseudoinstruction	Store the 64-bit double word in registers rsrc and rsrc+1 at address								
<b>Unaligned store halfword</b> ush rsrc, address	pseudoinstruction	Store the low halfword from register rsrc at the possibly unaligned address.								
<b>Unaligned store word</b> usw rsrc, address	pseudoinstruction	Store the word from register rsrc at the possibly unaligned address.								

Data movement instructions														
<b>Move from hi</b> mfhi rd	<table><tr><td>0</td><td>0</td><td>0</td><td>rd</td><td>0</td><td>0x10</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	0	0	rd	0	0x10	6	5	5	5	5	6	
0	0	0	rd	0	0x10									
6	5	5	5	5	6									
<b>Move from lo</b> mflo rd	<table><tr><td>0</td><td>0</td><td>0</td><td>rd</td><td>0</td><td>0x12</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	0	0	rd	0	0x12	6	5	5	5	5	6	The multiply and divide unit produces its results in two additional registers, hi and lo. These instructions move values to and from these registers.  Move the hi (lo) register to register rd.
0	0	0	rd	0	0x12									
6	5	5	5	5	6									
<b>Move to hi</b> mthi rs	<table><tr><td>0</td><td>rs</td><td>0</td><td>0</td><td>0</td><td>0x11</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	0	0	0	0x11	6	5	5	5	5	6	
0	rs	0	0	0	0x11									
6	5	5	5	5	6									
<b>Move to lo</b> mtlo rs	<table><tr><td>0</td><td>rs</td><td>0</td><td>0</td><td>0</td><td>0x13</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	rs	0	0	0	0x13	6	5	5	5	5	6	Move register rs to the high (lo) register.
0	rs	0	0	0	0x13									
6	5	5	5	5	6									
<b>Move from coprocessor z</b> mfcz rt, rd	<table><tr><td>0x1z</td><td>0</td><td>rt</td><td>rd</td><td>0</td><td>0</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0x1z	0	rt	rd	0	0	6	5	5	5	5	6	Coprocessors have their own register sets. These instructions move values between these registers and the CPU’s registers.  Move coprocessor z’s register rd to CPU register rt. The FP unit is z=1.
0x1z	0	rt	rd	0	0									
6	5	5	5	5	6									
<b>Move double from coprocessor 1</b> mfc1.d rdest, frsrc1	pseudoinstruction	Move FP registers frsrc1 and frsrc1+1 to CPU registers rdest and rdest+1.												
<b>Move to coprocessor z</b> mtcz rd, rt	<table><tr><td>0x1z</td><td>4</td><td>rt</td><td>rd</td><td>0</td><td>0</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0x1z	4	rt	rd	0	0	6	5	5	5	5	6	Move CPU register rt to coprocessor z’s register rd.
0x1z	4	rt	rd	0	0									
6	5	5	5	5	6									
<b>FP instructions (vergl. Patterson/Hennessy: Computer Organization &amp; Design)</b>														
<b>Exception and interrupt instructions</b>														
<b>Return from exception</b> rfe	<table><tr><td>0x10</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0x20</td></tr><tr><td>6</td><td>1</td><td>9</td><td>5</td><td>5</td><td>6</td></tr></table>	0x10	1	0	0	0	0x20	6	1	9	5	5	6	Restore the status register.
0x10	1	0	0	0	0x20									
6	1	9	5	5	6									
<b>System call</b> syscall	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0xc</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	0	0	0	0	0xc	6	5	5	5	5	6	Register \$v0 contains the number of the systems call provided by SPIM
0	0	0	0	0	0xc									
6	5	5	5	5	6									
<b>Break</b> break code	<table><tr><td>0</td><td colspan="4">code</td><td>0xd</td></tr><tr><td>6</td><td colspan="4">20</td><td>6</td></tr></table>	0	code				0xd	6	20				6	Cause exception code. Exception 1 is reserved for the debugger.
0	code				0xd									
6	20				6									
<b>No operation</b> nop	<table><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>6</td><td>5</td><td>5</td><td>5</td><td>5</td><td>6</td></tr></table>	0	0	0	0	0	0	6	5	5	5	5	6	Do nothing
0	0	0	0	0	0									
6	5	5	5	5	6									



## MIPS Register und Konventionen für die Verwendung der Register

Register name	Number	Usage
\$zero	0	constant 0
\$at	1	reserved for assembler
\$v0	2	expression evaluation and results of a function
\$v1	3	expression evaluation and results of a function
\$a0	4	argument 1
\$a1	5	argument 2
\$a2	6	argument 3
\$a3	7	argument 4
\$t0	8	temporary (not preserved across call)
\$t1	9	temporary (not preserved across call)
\$t2	10	temporary (not preserved across call)
\$t3	11	temporary (not preserved across call)
\$t4	12	temporary (not preserved across call)
\$t5	13	temporary (not preserved across call)
\$t6	14	temporary (not preserved across call)
\$t7	15	temporary (not preserved across call)
\$s0	16	saved temporary (preserved across call)
\$s1	17	saved temporary (preserved across call)
\$s2	18	saved temporary (preserved across call)
\$s3	19	saved temporary (preserved across call)
\$s4	20	saved temporary (preserved across call)
\$s5	21	saved temporary (preserved across call)
\$s6	22	saved temporary (preserved across call)
\$s7	23	saved temporary (preserved across call)
\$t8	24	temporary (not preserved across call)
\$t9	25	temporary (not preserved across call)
\$k0	26	reserved for OS kernel
\$k1	27	reserved for OS kernel
\$gp	28	pointer to global area
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address (used by function call)

Reference:

<http://www.cs.wisc.edu/~larus/SPIM/cod-appa.pdf>