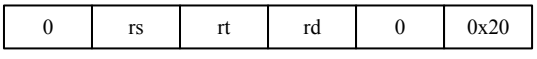
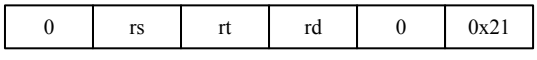
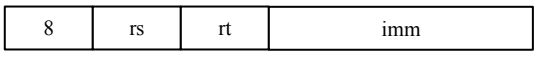
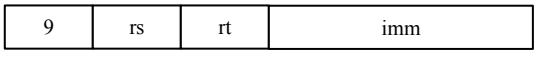
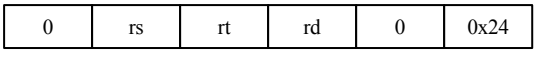
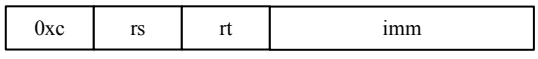
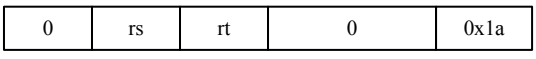
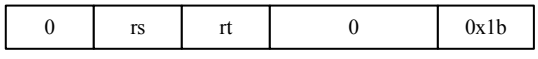
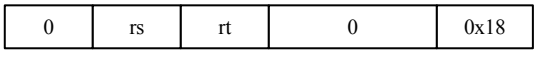
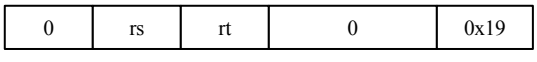
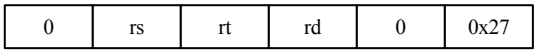
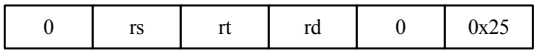
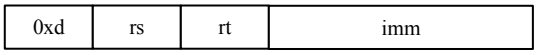
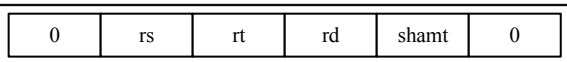
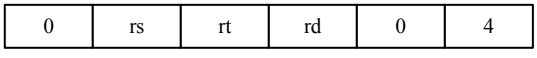
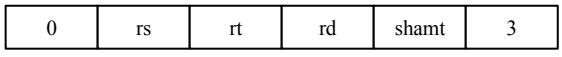
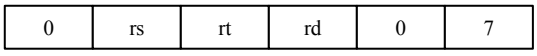
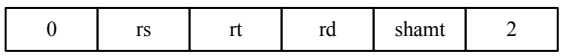
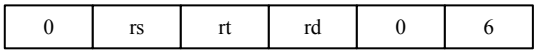
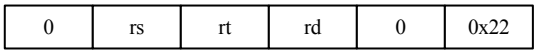
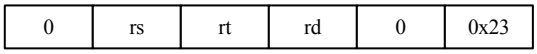
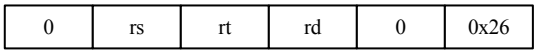
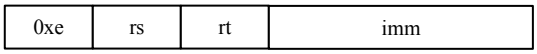
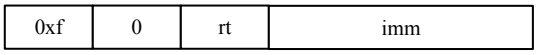
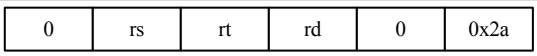
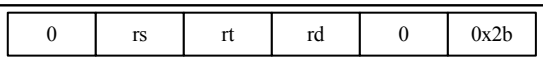
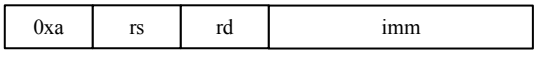
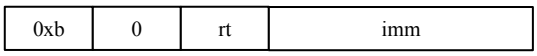
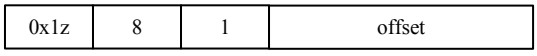
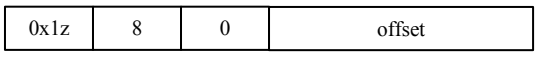
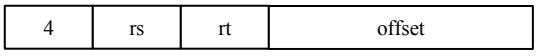
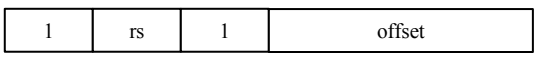
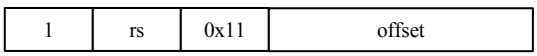
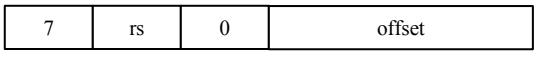
MIPS R2000 Assembly Language

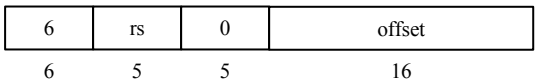
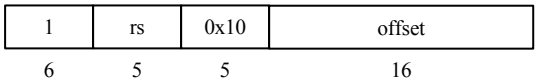
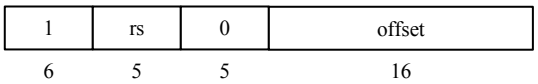
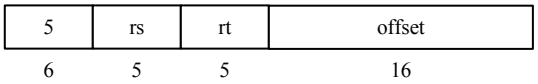
Arithmetic and Logical Instructions

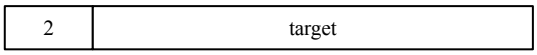
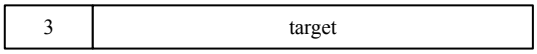
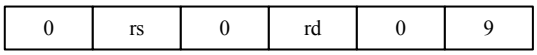
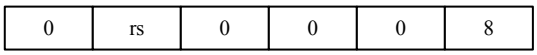
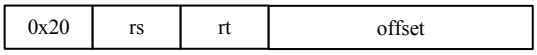
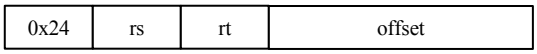
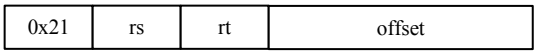
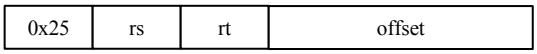
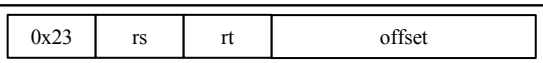
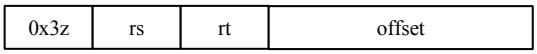
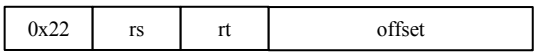
Instruction	Format	Comment
Absolute value abs rdest, rsrc	pseudoinstruction	Put the absolute value of register rsrc in register rdest
Addition (with overflow) add rd, rs, rt		
Addition (without overflow) addu rd, rs, rt		Put the sum of the register rs and rt into register rd
Addition immediate (with overflow) addi rt, rs, imm		
Addition immediate (without overflow) addiu rt, rs, imm		Put the sum of register rs and the sign-extended immediate into register rt
AND and rd, rs, rt		Put the logical AND of register rs and rt into register rd
AND immediate andi rt, rs, imm		Put the logical AND of register rs and the zero-extended immediate into register rt
Divide (with overflow) div rs, rt		
Divide (without overflow) divu rs, rt		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.
Divide (with overflow) div rdest, rsrc1, src2	pseudoinstruction	
Divide (without overflow) div rdest, rsrc1, src2	pseudoinstruction	Put the quotient of register rsrc1 and src2 into register rdest.
Multiply mult rs, rt		
Unsigned multiply multu rs, rt		Multiply registers rs and rt. Leave the low-order word of the product in register lo and the high-order word in register hi

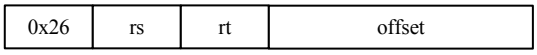
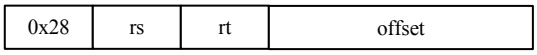
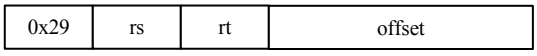
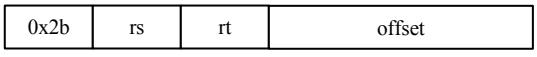
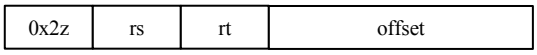
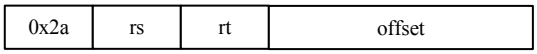
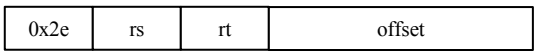
Multiply (without overflow) mul rdest, rsrc1, src2	pseudoinstruction	
Multiply (with overflow) mulo rdest, rsrc1, src2	pseudoinstruction	
Unsigned multiply (with overflow) mulou rdest, rsrc1, src2	pseudoinstruction	Put the product of register rsrc1 and src2 into register rdest.
Negate value (with overflow) neg rdest, rsrc	pseudoinstruction	
Negate value (without overflow) negu rdest, rsrc	pseudoinstruction	Put the negative of register rsrc into register rd.
NOR nor rd, rs, rt		Put the logical NOR of registers rs and rt into register rd
NOT not rdest, rsrc	pseudoinstruction	Put the bitwise logical negation of register rsrc into register rdest.
OR or rd, rs, rt		Put the logical OR of registers rs and rt into register rd.
OR immediate ori rt, rs, imm		Put the logical OR of register rs and the zero-extended immediate into register rt.
Remainder rem rdest, rsrc1, rsrc2	pseudoinstruction	
Unsigned remainder 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.
Shift left logical sll rd, rt, shamt		
Shift left logical variable sllv rd, rt, rs		
Shift right arithmetic sra rd, rt, shamt		
Shift right arithmetic variable srav rd, rt, rs		
Shift right logical srl rd, rt, shamt		

Shift right logical variable srlv rd, rt, rs		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.
Rotate left rol rdest, rsrc1, rsrc2	pseudo-instruction	
Rotate right ror rdest, rsrc1, rsrc2	pseudo-instruction	Rotate register rsrc1 left (right) by the distance indicated by rsrc2 and put the result into register rdest.
Subtract (with overflow) sub rd, rs, rt		
Subtract (without overflow) subu rd, rs, rt		Put the difference of registers rs and rt into register rd.
Exclusive OR xor rd, rs, rt		Put the logical XOR of registers rs and rt into register rd.
XOR immediate xori rt, rs, imm		Put the logical XOR of register rs and the zero-extended immediate into register rt.
Constant-Manipulating Instructions		
Load upper immediate lui rt, imm		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.
Load immediate li rdest, imm	pseudoinstruction	Move the immediate imm into register rdest.
Comparison instructions		
Set less than slt rd, rs, rt		
Set less than unsigned sltu rd, rs, rt		Set register rd to 1 if register rs is less than rt, and to 0 otherwise.
Set less than immediate slti rd, rs, imm		
Set less than unsigned immediate sltiu rd, rs, imm		Set register rd to 1 if register rs is less than the sign-extended immediate, and to o otherwise.
Set equal seq rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 equals rsrc2, and to 0 otherwise.

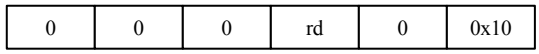
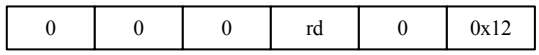
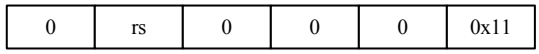
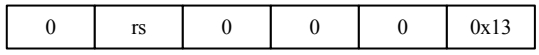
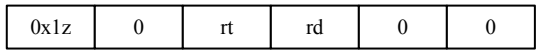
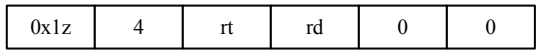
Set greater than equal sge rdest, rsrc1, rsrc2	pseudoinstruction	
Set greater than equal unsigned sgeu rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 is greater than or equal to register rsrc2, and to 0 otherwise.
Set greater than sgt rdest, rsrc1, rsrc2	pseudoinstruction	
Set greater than unsigned sgtu rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 is greater than register rsrc2, and to 0 otherwise.
Set less than equal sle rdest, rsrc1, rsrc2	pseudoinstruction	
Set less than equal unsigned sleu rdest, rsrc1, rsrc2	pseudoinstruction	Set register rdest to 1 if register rsrc1 is less than or equal to rsrc2, and to 0 otherwise.
Branch instructions		
Branch instruction b label	pseudoinstruction	Unconditionally branch to the instruction at the label.
Branch coprocessor z true bczt label		
Branch coprocessor z false bczf label		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.
Branch on equal beq rs, rt, label		Conditionally branch the number of instructions specified by the offset if register rs equals rt.
Branch on greater than equal zero bgez rs, label		Conditionally branch the number of instructions specified by the offset if register rs is greater than or equal to 0.
Branch on greater than equal zero and link bgezal rs, label		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.
Branch on greater than zero bgtz rs, label		Conditionally branch the instructions specified by the offset if register rs is greater than 0.

Branch on less than equal zero blez rs, label		Conditionally branch the instructions specified by the offset if register rs is less than or equal to 0.
Branch on less than zero and link bltzal rs, label		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.
Branch on less than zero bltz rs, label		Conditionally branch the instructions specified by the offset if register rs is less than 0.
Branch on not equal bne rs, rt, label		Conditionally branch the instructions specified by the offset if register rs is not equal to rt.
Branch on equal zero beqz rsrc, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc equals 0.
Branch on greater than equal bge rsrc1, rsrc2, label	pseudoinstruction	
Branch on greater than equal unsigned bgeu rsrc1, rsrc2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is greater than or equal to rsrc2.
Branch on greater than bgt rsrc1, src2, label	pseudoinstruction	
Branch on greater than unsigned bgtu rsrc1, src2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is greater than src2.
Branch on less than equal ble rsrc1, src2, label	pseudoinstruction	
Branch on less than equal unsigned bleu rsrc1, src2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is less than or equal to src2.
Branch on less than blt rsrc1, src2, label	pseudoinstruction	
Branch on less than unsigned bltu rsrc1, src2, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc1 is less than src2.

Branch on not equal zero bnez rsrc, label	pseudoinstruction	Conditionally branch to the instruction at the label if register rsrc is not equal to zero
Jump instructions		
Jump j target		Unconditionally jump to the instruction at target.
Jump and link jal target		Unconditionally jump to the instruction at target. Save the address of the next instruction in register $ra.
Jump and link register jalr rs, rd		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).
Jump register jr rs		Unconditionally jump to the instruction whose address is in register rs.
Load instructions		
Load rdest, address la rdest, address	pseudoinstruction	Load computed address – not the contents of the location – into register rd.
Load byte lb rt, address		
Load unsigned byte lbu rt, address		Load the byte at address into register rt. The byt is sign-extended by lb, but not by lbu.
Load halfword lh rt address		
Load unsigned halfword lhu rt, address		Load the byte at address into register rt. The byt is sign-extended by lh, but not by lhu.
Load word lw rt, address		Load 32-bit word at address into register rt.
Load word coprocessor lwcz rt, address		Load the word at address into register rt of coprocessor z (0-3). The FP unit is z = 1.
Load word left lwl rt, address		

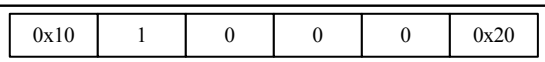
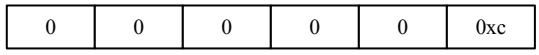
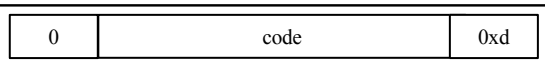
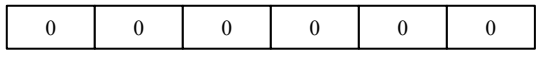
Load word right lwr rt, address		Load the left (right) bytes from the word at the possibly unaligned address into register rt.
Load doubleword ld rdest, address	pseudoinstruction	Load the 64-bit double word at address into registers rdest and rest + 1.
Unaligned load halfword ulh rdest, address	pseudoinstruction	
Unaligned load halfword unsigned 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.
Unaligned load word ulw rdest, address	pseudoinstruction	Load the 32-bit word at the possibly unaligned address into register rdest.
Store instructions		
Store byte sb rt, address		Store the low byte from register rt at address.
Store halfword sh rt, address		Store the low halfword from register rt at address.
Store word sw rt, address		Store the word from register rt at address.
Store word coprocessor swcz rt, address		Store the word from register rt of coprocessor z at address. The FP unit is z=1.
Store word left swl rt, address		
Store word right swr rt, address		Store the left (right) bytes from register rt at the possibly unaligned address.
Store doubleword sd rsrc, address	pseudoinstruction	Store the 64-bit double word in registers rsrc and rsrc+1 at address
Unaligned store halfword ush rsrc, address	pseudoinstruction	Store the low halfword from register rsrc at the possibly unaligned address.
Unaligned store word usw rsrc, address	pseudoinstruction	Store the word from register rsrc at the possibly unaligned address.

Data movement instructions

Move from hi mfhi rd		
Move from lo mflo rd		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.
Move to hi mthi rs		
Move to lo mtlo rs		Move register rs to the high (lo) register.
Move from coprocessor z mfcz rt, rd		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.
Move double from coprocessor 1 mfc1.d rdest, frsrc1	pseudoinstruction	Move FP registers frsrc1 and frsrc1+1 to CPU registers rdest and rdest+1.
Move to coprocessor z mtcz rd, rt		Move CPU register rt to coprocessor z's register rd.

FP instructions (vergl. Patterson/Hennessy: Computer Organization & Design)

Exception and interrupt instructions

Return from exception rfe		Restore the status register.
System call syscall		Register $v0 contains the number of the systems call provided by SPIM
Break break code		Cause exception code. Exception 1 is reserved for the debugger.
No operation nop		Do nothing

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