Common MIPS instructions.

Notes: *op*, *funct*, *rd*, *rs*, *rt*, *imm*, *address*, *shamt* refer to fields in the instruction format. The program counter PC is assumed to point to the next instruction (usually 4 + the address of the current instruction). M is the byte-addressed main memory.

Assembly instruction	Instr. format	*op* *op/funct*	Meaning	Comments
add $rd, $rs, $rt	R	0/32	$rd = $rs + $rt	Add contents of two registers
sub $rd, $rs, $rt	R	0/34	$rd = $rs - $rt	Subtract contents of two registers
addi $rt, $rs, imm	I	8	$rt = $rs + imm	Add signed constant
addu $rd, $rs, $rt	R	0/33	$rd = $rs + $rt	Unsigned, no overflow
subu $rd, $rs, $rt	R	0/35	$rd = $rs - $rt	Unsigned, no overflow
addiu $rt, $rs, imm	I	9	$rt = $rs + imm	Unsigned, no overflow
mfc0 $rt, $rd	R	16	$rt = $rd	*rd* = coprocessor register (e.g. epc, cause, status)
mult $rs, $rt	R	0/24	Hi, Lo = $rs * $rt	64 bit signed product in Hi and Lo
multu $rs, $rt	R	0/25	Hi, Lo = $rs * $rt	64 bit unsigned product in Hi and Lo
div $rs, $rt	R	0/26	Lo = $rs / $rt, Hi = $rs mod $rt	
divu $rs, $rt	R	0/27	Lo = $rs / $rt, Hi = $rs mod $rt (unsigned)	
mfhi $rd	R	0/16	$rd = Hi	Get value of Hi
mflo $rd	R	0/18	$rd = Lo	Get value of Lo
and $rd, $rs, $rt	R	0/36	$rd = $rs & $rt	Logical AND
or $rd, $rs, $rt	R	0/37	$rd = $rs | $rt	Logical OR
andi $rt, $rs, imm	I	12	$rt = $rs & imm	Logical AND, unsigned constant
ori $rt, $rs, imm	I	13	$rt = $rs | imm	Logical OR, unsigned constant
sll $rd, $rs, shamt	R	0/0	$rd = $rs << shamt	Shift left logical (shift in zeros)
srl $rd, $rs, shamt	R	0/2	$rd = $rs >> shamt	Shift right logical (shift in zeros)
lw $rt, imm($rs)	I	35	$rt = M[$rs + imm]	Load word from memory
sw $rt, imm($rs)	I	43	M[$rs + imm] = $rt	Store word in memory
lbu $rt, imm($rs)	I	37	$rt = M[$rs + imm]	Load a single byte, set bits 8-31 of $rt to zero
sb $rt, imm($rs)	I	41	M[$rs + imm] = $rt	Store byte (bits 0-7 of $rt) in memory
lui $rt, imm	I	15	$rt = imm * 2^{16}	Load constant in bits 16-31 of register $rt
beq $rs, $rt, imm	I	4	if($rs==$rt) PC = PC + imm (PC always points to next instruction)	
bne $rs, $rt, imm	I	5	if($rs!=$rt) PC = PC + imm (PC always points to next instruction)	
slt $rd, $rs, $rt	R	0/42	if($rs<$rt) $rd = 1; else $rd = 0	
slti $rt, $rs, imm	I	10	if($rs<imm) $rt = 1; else $rt = 0	
sltu $rd, $rs, $rt	R	0/43	if($rs<$rt) $rd = 1; else $rd = 0 (unsigned numbers)	
sltiu $rt, $rs, imm	I	11	if($rs<imm) $rt = 1; else $rt = 0 (unsigned numbers)	
j destination	J	2	PC = address*4	Jump to *destination, address = destination*/4
jal destination	J	3	$ra = PC; PC = address*4 (Jump and link, *address = destination*/4)	
jr $rs	R	0/8	PC = $rs	Jump to address stored in register $rs

MIPS registers

Name	Number	Usage
`$zero`	0	constant 0
`$at`	1	reserved for assembler
`$v0 - $v1`	2-3	expression evaluation and function results
`$a0 - $a3`	4-7	arguments
`$t0 - $t7`	8-15	temporary, saved by caller
`$s0 - $s7`	16-23	temporary, saved by called function
`$t8 - $t9`	24-25	temporary, saved by caller
`$k0 - $k1`	26-27	reserved for kernel (OS)
`$gp`	28	points to middle of a 64K block in the data segment
`$sp`	29	stack pointer (top of stack)
`$fp`	30	frame pointer (beginning of current frame)
`$ra`	31	return address
`Hi, Lo`	-	store partial result of mult and div operations
`PC`	-	contains the address of the next instruction to be fetched (this is not a real MIPS register, and is only used to define instructions)
`status`	-	register 12 in coprocessor 0, stores interrupt mask and enable bits
`cause`	-	register 13 in coprocessor 0, stores exception type and pending interrupt bits
`epc`	-	register 14 in coprocessor 0, stores address of instruction causing exception

MIPS Instruction formats

Format	Bits 31-26	Bits 25-21	Bits 20-16	Bits 15-11	Bits 10-6	Bits 5-0
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	imm		
J	op	address				