

## Instruction Types

R-Type (Register Type) Instruction					
Bit <sub>31-26</sub>	Bit <sub>25-21</sub>	Bit <sub>20-16</sub>	Bit <sub>15-11</sub>	Bit <sub>10-6</sub>	Bit <sub>5-0</sub>
opcode	register s	register t	register d	shift amount	function

I-Type (Immediate Type) Instruction				J-Type (Jump Type) Instruction	
Bit <sub>63-44</sub>	Bit <sub>43-37</sub>	Bit <sub>36-30</sub>	Bit <sub>29-0</sub>	Bit <sub>31-26</sub>	Bit <sub>25-0</sub>
opcode	register s	register t	immediate	opcode	target

## Registers

Register Number	Register Alias	Register Information
0	\$0	Constant. Always holds the value 0.
1	\$at	( <b>assembler temporary</b> ) Reserved by the assembler.
2 - 3	\$v0 - \$v1	( <b>values</b> ) Results from expression evaluation and function returns.
4 - 7	\$a0 - \$a3	( <b>arguments</b> ) First four parameters for subroutine. Not preserved across procedure calls.
8 - 15	\$t0 - \$t7	( <b>temporaries</b> ) Temporary registers for computation.
16 - 23	\$s0 - \$s7	( <b>saved values</b> ) Registers preserved across procedure calls. Must save and restore in subroutine.
24 - 25	\$t8 - \$t9	( <b>temporaries</b> ) Temporary registers for computation.
26 - 27	\$k0 - \$k1	Reserved for use by interrupt / trap handler.
28	\$gp	( <b>global pointer</b> ) Points to the middle of the 64K block of memory in the static data segment.
29	\$sp	( <b>stack pointer</b> ) Points to last location on the stack.
30	\$fp	( <b>frame pointer</b> ) Must save and restore in subroutine.
31	\$ra	( <b>return address</b> ). Address to return to after subroutine.

## Instructions

### Load Instructions (I-Type)

lw	\$t0 label	# \$t0 = *label	Loads word at label into a register
lw	\$t0 off(\$t1)	# \$t0 = MEM[\$t1 + off]	Loads word from offset memory address in register to a register
lb	\$t0 label	# \$t0 = *label	Loads byte at label into a register
lb	\$t0 off(\$t1)	# \$t0 = MEM[\$t1 + off]	Loads byte from offset memory address in register (sign padded)
lbu	\$t0 off(\$t1)	# \$t0 = MEM[\$t1 + off]	Loads byte from offset memory address in register (0 padded)
li	\$t0 imm	# \$t0 = imm	Loads constant value into a register
la	\$t0 label	# \$t0 = &label	Loads the pointer to a label into a register

## Store Instructions (I-Type)

sw	\$t0 label	# MEM[&label] = \$t0	Stores a word into a labelled memory location
sw	\$t0 off(\$t1)	# MEM[\$t1 + off] = \$t0	Stores a word into offset memory address from register
sb	\$t0 label	# MEM[&label] = \$t0	Stores a byte into a labelled memory location
sb	\$t0 off(\$t1)	# MEM[\$t1 + off] = \$t0	Stores a byte into offset memory address from register

## Basic Arithmetic (R-Type)

add	\$t0 \$t1 \$t2	# \$t0 = \$t1 + \$t2	Adds two signed (2's compliment) registers
addu	\$t0 \$t1 \$t2	# \$t0 = \$t1 + \$t2	Adds two unsigned registers
addi	\$t0 \$t1 imm	# \$t0 = \$t1 + imm	Adds a register and a constant (I-Type Instruction)
sub	\$t0 \$t1 \$t2	# \$t0 = \$t1 - \$t2	Subtracts two signed (2's compliment) registers
subu	\$t0 \$t1 \$t2	# \$t0 = \$t1 - \$t2	Subtracts two unsigned registers
mult	\$t0 \$t1	# (Hi, Lo) = \$t0 * \$t1	Multiplies two registers. Stores first 32 bits in Hi and last in Lo
div	\$t0 \$t1	# Hi =  _ \$t0 / \$t1_  # Lo = \$t0 % \$t2	Divides two registers. Stores the integer result in Hi Stores the remainder in Lo
mfhi	\$t0	# \$t0 = Hi	Stores the Hi value in a register
mflo	\$t0	# \$t0 = Lo	Stores the Lo value in a register

## Bitwise Operations (R-Type)

and	\$t0 \$t1 \$t2	# \$t0 = \$t1 & \$t2	Performs bitwise and on two registers
or	\$t0 \$t1 \$t2	# \$t0 = \$t1   \$t2	Performs bitwise or on two registers
nor	\$t0 \$t1 \$t2	# \$t0 = \$t1 ~ \$t2	Performs bitwise nor on two registers
xor	\$t0 \$t1 \$t2	# \$t0 = \$t1 ^ \$t2	Performs bitwise xor on two registers

## Bitwise Operations (I-Type)

ori	\$t0 \$t2 imm	# \$t0 = \$t1   imm	Performs bitwise and on a register and a constant
andi	\$t0 \$t1 imm	# \$t0 = \$t1 & imm	Performs bitwise and on a register and a constant
xori	\$t0 \$t1 imm	# \$t0 = \$t1 ^ imm	Performs bitwise xor on a register and a constant

## BitShift Operations (R-Type)

sll	\$t0 \$t1 imm	# \$t0 = \$t1 << imm	Shifts a register left by a constant amount (0 padded)
sllv	\$t0 \$t1 \$t2	# \$t0 = \$t1 << \$t2	Shifts a register left by the amount in a register (0 padded)
srl	\$t0 \$t1 imm	# \$t0 = \$t1 >> imm	Shifts a register right by a constant amount (0 padded)
srlv	\$t0 \$t1 \$t2	# \$t0 = \$t1 >> \$t2	Shifts a register right by the amount in a register (0 padded)
sra	\$t0 \$t1 imm	# \$t0 = \$t1 / (2^imm)	Shifts a register right by an immediate value (sign padded)
srav	\$t0 \$t1 \$t2	# \$t0 = \$t1 / (2^ \$t2)	Shifts a register right bt the amount in a register (sign padded)

## Branch Operations (I-Type)

beq	\$t0 \$t1 label	# \$t0 == \$t1 ? goto label	Branch to label if two registers are equal
bne	\$t0 \$t1 label	# \$t0 != \$t1 ? goto label	Branch to label if two registers are not equal
bgt	\$t0 \$t1 label	# \$t0 > \$t1 ? goto label	Branch to label if one register is greater than another
bgtz	\$t0 label	# \$t0 > 0 ? goto label	Branch to label if register is greater than zero
bge	\$t0 \$t1 label	# \$t0 >= \$t1 ? goto label	Branch to label if one register is greater than or equal to another
bgez	\$t0 label	# \$t0 >= 0 ? goto label	Branch to label if register is greater than or equal to zero
blt	\$t0 \$t1 label	# \$t0 < \$t1 ? goto label	Branch to label if one register is less than another
bltz	\$t0 label	# \$t0 < 0 ? goto label	Branch to label if register is less than zero
ble	\$t0 \$t1 label	# \$t0 <= \$t1 ? goto label	Branch to label if one register is less than or equal to another
blez	\$t0 label	# \$t0 <= 0 ? goto label	Branch to label if register is less than or equal to zero

## Jump Operations (J-Type)

j	label	# goto label	Jumps to a label
jal	label	# \$ra = PC; goto label	Jumps to a label, storing location in return register

## Jump Operations (I-Type)

jr	\$t0	# goto \$t0	Jumps to location stored in register
jalr	\$t0	# \$ra = PC; goto \$t0	Jumps to location stored in register, storing location in return register