

# MIPS Instruction Set

## Arithmetic Instructions

Instruction	Example	Meaning	Comments
<b>add</b>	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	
<b>subtract</b>	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	
<b>add immediate</b>	addi \$1,\$2,100	$\$1 = \$2 + 100$	"Immediate" means a constant number
<b>add unsigned</b>	addu \$1,\$2,\$3	$\$1 = \$2 + \$3$	Values are treated as unsigned integers, not two's complement integers
<b>subtract unsigned</b>	subu \$1,\$2,\$3	$\$1 = \$2 - \$3$	Values are treated as unsigned integers, not two's complement integers
<b>add immediate unsigned</b>	addiu \$1,\$2,100	$\$1 = \$2 + 100$	Values are treated as unsigned integers, not two's complement integers
<b>Multiply (without overflow)</b>	mul \$1,\$2,\$3	$\$1 = \$2 * \$3$	Result is only 32 bits!
<b>Multiply</b>	mult \$2,\$3	$\$hi, \$low = \$2 * \$3$	Upper 32 bits stored in special register <code>hi</code> Lower 32 bits stored in special register <code>lo</code>
<b>Divide</b>	div \$2,\$3	$\$hi, \$low = \$2 / \$3$	Remainder stored in special register <code>hi</code> Quotient stored in special register <code>lo</code>

## Logical

Instruction	Example	Meaning	Comments
<b>and</b>	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	Bitwise AND
<b>or</b>	or \$1,\$2,\$3	$\$1 = \$2   \$3$	Bitwise OR
<b>and immediate</b>	andi \$1,\$2,100	$\$1 = \$2 \& 100$	Bitwise AND with immediate value
<b>or immediate</b>	ori \$1,\$2,100	$\$1 = \$2   100$	Bitwise OR with immediate value
<b>shift left logical</b>	sll \$1,\$2,10	$\$1 = \$2 \ll 10$	Shift left by constant number of bits
<b>shift right logical</b>	srl \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right by constant number of bits

## Data Transfer

Instruction	Example	Meaning	Comments
<b>load word</b>	lw \$1,100(\$2)	$\$1 = \text{Memory}[\$2 + 100]$	Copy from memory to register
<b>store word</b>	sw \$1,100(\$2)	$\text{Memory}[\$2 + 100] = \$1$	Copy from register to memory
<b>load upper immediate</b>	lui \$1,100	$\$1 = 100 \times 2^{16}$	Load constant into upper 16 bits. Lower 16 bits are set to zero.
<b>load address</b>	la \$1,label	$\$1 = \text{Address of label}$	<i>Pseudo-instruction</i> (provided by assembler, not processor!) Loads computed address of label (not its contents) into register
<b>load immediate</b>	li \$1,100	$\$1 = 100$	<i>Pseudo-instruction</i> (provided by assembler, not processor!) Loads immediate value into register

<b>move from hi</b>	<code>mfhi \$2</code>	$\$2 = hi$	Copy from special register <code>hi</code> to general register
<b>move from lo</b>	<code>mflo \$2</code>	$\$2 = lo$	Copy from special register <code>lo</code> to general register
<b>move</b>	<code>move \$1, \$2</code>	$\$1 = \$2$	<i>Pseudo-instruction</i> (provided by assembler, not processor!) Copy from register to register.

## Conditional Branch

Instruction	Example	Meaning	Comments
<b>branch on equal</b>	<code>beq \$1, \$2, 100</code>	if( $\$1 == \$2$ ) go to PC+4+100	Test if registers are equal
<b>branch on not equal</b>	<code>bne \$1, \$2, 100</code>	if( $\$1 \neq \$2$ ) go to PC+4+100	Test if registers are not equal
<b>branch on greater than</b>	<code>bgt \$1, \$2, 100</code>	if( $\$1 > \$2$ ) go to PC+4+100	<i>Pseudo-instruction</i>
<b>branch on greater than or equal</b>	<code>bge \$1, \$2, 100</code>	if( $\$1 \geq \$2$ ) go to PC+4+100	<i>Pseudo-instruction</i>
<b>branch on less than</b>	<code>blt \$1, \$2, 100</code>	if( $\$1 < \$2$ ) go to PC+4+100	<i>Pseudo-instruction</i>
<b>branch on less than or equal</b>	<code>ble \$1, \$2, 100</code>	if( $\$1 \leq \$2$ ) go to PC+4+100	<i>Pseudo-instruction</i>

## Comparison

Instruction	Example	Meaning	Comments
<b>set on less than</b>	<code>slt \$1,\$2,\$3</code>	if( $\$2 < \$3$ ) $\$1=1$ ; else $\$1=0$	Test if less than. If true, set $\$1$ to 1. Otherwise, set $\$1$ to 0.
<b>set on less than immediate</b>	<code>slti \$1,\$2,100</code>	if( $\$2 < 100$ ) $\$1=1$ ; else $\$1=0$	Test if less than. If true, set $\$1$ to 1. Otherwise, set $\$1$ to 0.

## Unconditional Jump

Instruction	Example	Meaning	Comments
<b>jump</b>	<code>j 1000</code>	go to address 1000	Jump to target address
<b>jump register</b>	<code>jr \$1</code>	go to address stored in $\$1$	For switch, procedure return
<b>jump and link</b>	<code>jal 1000</code>	$\$ra=PC+4$ ; go to address 1000	Use when making procedure call. This saves the return address in $\$ra$

## System Calls

Service	Operation	Code (in $\$v0$ )	Arguments	Results
<b>print_int</b>	Print integer number (32 bit)	1	$\$a0$ = integer to be printed	None
<b>print_float</b>	Print floating-point number (32 bit)	2	$\$f12$ = float to be printed	None
<b>print_double</b>	Print floating-point number (64 bit)	3	$\$f12$ = double to be printed	None

<b>print_string</b>	Print null-terminated character string	4	\$a0 = address of string in memory	None
<b>read_int</b>	Read integer number from user	5	None	Integer returned in \$v0
<b>read_float</b>	Read floating-point number from user	6	None	Float returned in \$f0
<b>read_double</b>	Read double floating-point number from user	7	None	Double returned in \$f0
<b>read_string</b>	Works the same as Standard C Library <code>fgets()</code> function.	8	\$a0 = memory address of string input buffer \$a1 = length of string buffer (n)	None
<b>sbrk</b>	Returns the address to a block of memory containing n additional bytes. (Useful for dynamic memory allocation)	9	\$a0 = amount	address in \$v0
<b>exit</b>	Stop program from running	10	None	None
<b>print_char</b>	Print character	11	\$a0 = character to be printed	None
<b>read_char</b>	Read character from user	12	None	Char returned in \$v0
<b>exit2</b>	Stops program from running and returns an integer	17	\$a0 = result (integer number)	None

## Assembler Directives

Directive	Result
<code>.word w1, ..., wn</code>	Store <i>n</i> 32-bit values in successive memory words
<code>.half h1, ..., hn</code>	Store <i>n</i> 16-bit values in successive memory words
<code>.byte b1, ..., bn</code>	Store <i>n</i> 8-bit values in successive memory words

<code>.ascii str</code>	Store the ASCII string <code>str</code> in memory. Strings are in double-quotes, i.e. "Computer Science"
<code>.asciiz str</code>	Store the ASCII string <code>str</code> in memory and null-terminate it Strings are in double-quotes, i.e. "Computer Science"
<code>.space n</code>	Leave an empty $n$ -byte region of memory for later use
<code>.align n</code>	Align the next datum on a $2^n$ byte boundary. For example, <code>.align 2</code> aligns the next value on a word boundary

## Registers

Register Number	Register Name	Description
0	<b>\$zero</b>	The value 0
2-3	<b>\$v0 - \$v1</b>	(values) from expression evaluation and function results
4-7	<b>\$a0 - \$a3</b>	(arguments) First four parameters for subroutine
8-15, 24-25	<b>\$t0 - \$t9</b>	Temporary variables
16-23	<b>\$s0 - \$s7</b>	Saved values representing final computed results
31	<b>\$ra</b>	Return address