# MIPS Instruction Set

## Arithmetic Instructions

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

## Logical

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

## Data Transfer

| Instruction | Example | Meaning | Comments |
|---|---|---|---|
| **load word** | `lw $1,100($2)` | $1=Memory[$2+100] | Copy from memory to register |
| **store word** | `sw $1,100($2)` | Memory[$2+100]=$1 | Copy from register to memory |
| **load upper immediate** | `lui $1,100` | $1=100x2^16 | Load constant into upper 16 bits.<br>Lower 16 bits are set to zero. |
| **load address** | `la $1,label` | $1=Address of label | *Pseudo-instruction* (provided by assembler, not processor!)<br>Loads computed address of label (not its contents) into register |
| **load immediate** | `li $1,100` | $1=100 | *Pseudo-instruction* (provided by assembler, not processor!)<br>Loads immediate value into register |

| | | | |
|---|---|---|---|
| **move from hi** | `mfhi $2` | $2=hi | Copy from special register `hi` to general register |
| **move from lo** | `mflo $2` | $2=lo | Copy from special register `lo` to general register |
| **move** | `move $1,$2` | $1=$2 | *Pseudo-instruction* (provided by assembler, not processor!)<br>Copy from register to register. |

## Conditional Branch

| Instruction | Example | Meaning | Comments |
|---|---|---|---|
| **branch on equal** | `beq $1,$2,100` | if($1==$2) go to PC+4+100 | Test if registers are equal |
| **branch on not equal** | `bne $1,$2,100` | if($1!=$2) go to PC+4+100 | Test if registers are not equal |
| **branch on greater than** | `bgt $1,$2,100` | if($1>$2) go to PC+4+100 | *Pseduo-instruction* |
| **branch on greater than or equal** | `bge $1,$2,100` | if($1>=$2) go to PC+4+100 | *Pseduo-instruction* |
| **branch on less than** | `blt $1,$2,100` | if($1<$2) go to PC+4+100 | *Pseduo-instruction* |
| **branch on less than or equal** | `ble $1,$2,100` | if($1<=$2) go to PC+4+100 | *Pseduo-instruction* |

## Comparison

| Instruction | Example | Meaning | Comments |
|---|---|---|---|
| **set on less than** | `slt $1,$2,$3` | if($2<$3)$1=1; else $1=0 | Test if less than. If true, set $1 to 1. Otherwise, set $1 to 0. |
| **set on less than immediate** | `slti $1,$2,100` | 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 |
|---|---|---|---|
| **jump** | `j 1000` | go to address 1000 | Jump to target address |
| **jump register** | `jr $1` | go to address stored in $1 | For switch, procedure return |
| **jump and link** | `jal 1000` | $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 |
|---|---|---|---|---|
| **print_int** | Print integer number (32 bit) | 1 | $a0 = integer to be printed | None |
| **print_float** | Print floating-point number (32 bit) | 2 | $f12 = float to be printed | None |
| **print_double** | Print floating-point number (64 bit) | 3 | $f12 = double to be printed | None |

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

## Assembler Directives

| Directive | Result |
|---|---|
| `.word w1, ..., wn` | Store *n* 32-bit values in successive memory words |
| `.half h1, ..., hn` | Store *n* 16-bit values in successive memory words |
| `.byte b1, ..., bn` | Store *n* 8-bit values in successive memory words |

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

## Registers

| Register Number | Register Name | Description |
|---|---|---|
| 0 | **$zero** | The value 0 |
| 2-3 | **$v0 - $v1** | (**v**alues) from expression evaluation and function results |
| 4-7 | **$a0 - $a3** | (**a**rguments) First four parameters for subroutine |
| 8-15, 24-25 | **$t0 - $t9** | **T**emporary variables |
| 16-23 | **$s0 - $s7** | **S**aved values representing final computed results |
| 31 | **$ra** | **R**eturn **a**ddress |