Registers

Name	Number	Use
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Evaluation of expressions and function results
v1	3	Evaluation of expressions and function results
a0	4	Argument 1
a1	5	Argument 2
a2	6	Argument 3
a3	7	Argument 4
t0..t7	8..15	Temporal (no value is saved between calls)
s0..s7	16..23	Temporal (value is saved between calls)
t8, t9	24, 25	Temporal (no value is saved between calls)
k0, k1	26, 27	Reserved for the operating system kernel
gp	28	Pointer to the global area
sp	29	Stack pointer
fp	30	Stack frame pointer
ra	31	Return address, used by function calls

System calls

Service	Call code	Arguments	Results
print_integer	$v0 = 1	$a0 = integer	
print_float	$v0 = 2	$f12 = real (32 bits)	
print_double	$v0 = 3	$f12 = real (64 bits)	
print_string	$v0 = 4	$a0 = string	
read_integer	$v0 = 5		Integer ($v0)
read_float	$v0 = 6		Real 32-bits ($f0)
read_double	$v0 = 7		Real 64-bits ($f0)
read_string	$v0 = 8	$a0 = buffer, $a1 = length	
sbrk	$v0 = 9	$a0 = quantity	Address ($v0)
exit	$v0 = 10		
print_char	$v0 = 11	$a0 = byte	
read_char	$v0 = 12		$v0 (ASCII code)

Assembly directives

.data	The following data definitions that appear are stored in the data segment. It can include an argument that indicates the address from where the data will begin to be stored.
.text	The instructions that follow this directive are placed in the code segment. It can include a parameter that indicates where the code zone begins.
.globl symbol	Declares a global symbol that can be referenced from other programs.
.extern label n	Declares that data stored from label occupies N bytes and that label is a global symbol. This directive allows the assembler to store data in an area of the data segment that can be accessed through the $gp register.
.ascii string	Store the string in memory, but it does not end with NULL ('\ 0')
.asciiz string	Store the string in memory and place a NULL ('\ 0') at the end.
.byte b1, ..., bn	Stores N values in successive bytes of memory.
.half h1, ..., hn	Stores N 16-bit numbers in consecutive half-words.
.word w1, ..., wn	Stores N 32-bit elements (words) in consecutive memory locations.
.float f1, ..., fn	Stores N real values of simple precision in consecutive memory positions.
.double d1, ..., dn	Stores N real double precision values in consecutive memory addresses.

Constant Handling Instructions

li Rdest, inmediate	Load immediate value
lui Rdest, inmediate	Load the 16 bits of the lower part of the immediate value in the upper part of the register. The bits of the lower part are set to 0

Data transfer instructions

move Rdest, Rsrc	Move the contents of the Rsrc register to the Rdest register.
mfhi Rdest	Move the contents of the HI register to the Rdest register.
mflo Rdest	Move the contents of the LO register to the Rdest register.
mthi Rsrc	Move the contents of the Rsrc register to the HI register.
mtlo Rsrc	Move the contents of the Rsrc register to the LO register.

Load instructions

la Rdest, address	Load address in Rdest (the address value, not the content)
lb Rdest, address	Load the byte of the specified address and extend the sign
lbu Rdest, address	Load the byte of the specified address, do not extend the sign
lh Rdest, address	Load 16 bits of the specified address, sign is extended
lhu Rdest, address	Load 16 bits of the specified address, no sign is extended
lw Rdest, address	Load a word from the specified address.

Storing instructions

sb Rsrc, address	Stores the lowest Rsrc byte in the indicated address.
sh Rsrc, address	Stores the low half word (16 bits) of a register in the indicated memory address.
sw Rsrc, address	Store the Rsrc in the indicated address.

CC BY NC SA

Arithmetic and logical instructions

In all the following instructions, *Src2* can be both a register and an immediate value (a 16-bit integer) and in those where it puts *inm* it only accepts an immediate value

add Rdest, Rsrc1, Src2	Sum with overflow
addi Rdest, Rsrc1, inm	Add an immediate number with overflow
addu Rdest, Rsrc1, Src2	Sum without overflow
addiu Rdest, Rsrc1, inm	Add an immediate number without overflow
and Rdest, Rsrc1, Src2	AND logical operation
andi Rdest, Rsrc1, inm	AND logical operation with an immediate number
div Rsrc1, Rsrc2	Divide with overflow. Leave the quotient in the register lo and the rest in the register hi
divu Rsrc1, Rsrc2	Divide without overflow. Leave quotient in the register lo and the rest in the register hi
div Rdest, Rsrc1, Rrc2	Divide with overflow
divu Rdest, Rsrc1, Rrc2	Divide without overflow
mul Rdest, Rsrc1, Src2	Multiply without overflow
mult Rsrc1, Rsrc2	Multiply, the low part of the result is left in the lo register and the high part in the hi register
multu Rsrc1, Rsrc2	Multiply without overflow, the low part of the result goes to LO and the high part to HI
mod Rdest, Rsrc1, Rsrc2	Division module with overflow
modu Rdest, Rsrc1, Rsrc2	Division module without overflow
nop	It does not perform any operation
nor Rdest, Rsrc1, Src2	NOR Logic Operation
or Rdest, Rsrc1, Src2	OR Logic Operation
ori Rdest, Rsrc1, inm	OR Logic Operation with immediate
rem Rdest, Rsrc1, Rsrc2	Division module with overflow
rotr rdest, rsrc1, inm	Right rotation of Src2 number of bits
sll Rdest, Rsrc1, inm	Logical bit shift to the left
srl Rdest, Rsrc1, inm	Logical bit shift to the right
sra Rdest, Rsrc1, inm	Arithmetic bit shift to the right
sub Rdest, Rsrc1, Src2	Subtraction (with overflow)
subu Rdest, Rsrc1, Src2	Subtraction (without overflow)
xor Rdest, Rsrc1, Src2	XOR Logic Operation

Branch and jump instructions

In all the following instructions, Src2 can be a record or an immediate value. Branch instructions use a signed 16-bit offset; So you can skip 215-1 instructions forward or 215 instructions backward. The jump instructions contain a 26-bit address field.

b label	Unconditional branch to the instruction that is on label.
beq Rsrc1, Src2, label	Conditional branch if Rsrc1 is equal to Src2.
beqz Rsrc, label	Conditional branch if the Rsrc register is equal to 0.
bge Rsrc1, Src2, label	Conditional branch if the Rsrc1 register is greater than or equal to Src2 (signed).
bgeu Rsrc1, Src2, label	Conditional branch if the Rsrc1 register is greater than or equal to Src2 (unsigned).
bgez Rsrc, label	Conditional branch if the Rsrc register is greater than or equal to 0.
bgezal Rsrc, label	Conditional branch if the Rsrc register is greater than or equal to 0. Save the current address in the $ra register ($31)
bgt Rsrc1, Src2, label	Conditional branch if the Rsrc1 register is greater than Src2 (signed).
bgtu Rsrc1, Src2, label	Conditional branch if the Rsrc1 register is greater than Src2 (unsigned).
bgtz Rsrc, label	Conditional branch if Rsrc is greater than 0.
ble Rsrc1, Src2, label	Conditional branch if Rsrc1 is less than or equal to Src2 (signed).
bleu Rsrc1, Src2, label	Conditional branch if Rsrc1 is less than or equal to Src2 (unsigned).
blez Rsrc, label	Conditional branch if Rsrc is less than or equal to 0.
blt Rsrc1, Src2, label	Conditional branch if Rsrc1 is less than Src2 (signed).
bltu Rsrc1, Src2, label	Conditional branch if Rsrc1 is less than Src2 (unsigned).
bltz Rsrc, label	Conditional branch if Rsrc is less than 0.
bne Rsrc1, Src2, label	Conditional branch if Rsrc1 is not equal to Src2.
bnez Rsrc, label	Conditional branch if Rsrc is not equal to 0.
j label	Unconditional jump.
jal label	Unconditional jump, stores the current address at $ ra ($31).
jalr Rsrc	Unconditional jump, stores the current address at $ ra ($31).
jalr Rsrc1, Rsrc2	Unconditional jump, stores the current address in Rsrc1.
jr Rsrc	Unconditional jump.

Comparison instructions

In all the following instructions, Src2 can be both a register and an immediate value (a 16-bit integer).

seq Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is equal to Src2, 0 in other case.
sge Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is greater or equal to Src2, and 0 in other case (with sign).
sgeu Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is greater or equal to Src2, and 0 in other case (without sign).
sgt Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is greater than Src2, and 0 in other case (with sign).
sgtu Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is greater than Src2, y 0 in other case (without sign).
sle Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is little or equal to Src2, 0 in other case (with sign).
sleu Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is little or equal to Src2, 0 in other case (without sign).
slt Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is little than Src2, 0 in other case (with sign).
sltu Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 is little than Src2, 0 in other case (without sign).
sne Rdest, Rsrc1, Src2	Set Rdest to 1 if Rsrc1 not equal to Src2, and 0 in other case.

(Floating point) Constant Handling Instructions

li.s fs, value	Load the value (float) into the fs register of the mathematical coprocessor
li.d fd, value	Load the value (double) into the fd register of the mathematical coprocessor

(Floating point) Data transfer instructions

mfc1 Rdest, CPsrc	Copy contents of the CPsrc register of the floating-point coprocessor to the Rdest CPU register.
mtc1 Rsrc, CPdest	Copy contents of the Rsrc register of the CPU to the CPdest register of the floating-point coprocessor.
mov.s fd, fs	Move the contents of the fs record to the fd record. (float)
mov.d fd, fs	Move the contents of the fs record to the fd record. (double)

(Floating point) Arithmetic and logical instructions

In all the following instructions, Src2 can be both a register and an immediate value (a 16-bit integer) and
in those where it puts inm it only accepts an immediate value

abs.s fd, fs	Absolute value of a real 32-bit number.
abs.d fd, fs	Absolute value of a real 64-bit number.
add.s fd, fs, ft	Add the register fs and ft and store the result in fd (float)
add.d fd, fs, ft	Add the register fs and ft and store the result in fd (double)
div.s fd, fs, ft	Divide fs by ft and leave the result in fd (float)
div.d fd, fs, ft	Divide fs by ft and leave the result in fd (double)
mul.s fd, fs, ft	Multiply the register fs and ft and leave your result in fd. (float)
mul.d fd, fs, ft	Multiply the register fs and ft and leave your result in fd. (double)
rsqrt.s fd, fs	Reciprocal Square Root fd = 1.0/sqrt.s(fs) (float)
rsqrt.d fd, fs	Reciprocal Square Root fd = 1.0/sqrt.s(fs) (double)
sqrt.s fd, fs	Square root of fs (float): fd=sqrt(fs)
sqrt.d fd, fs	Square root of fs (double)
sub.s fd, fs, ft	Subtraction (float): fd = fs-ft
sub.d fd, fs, ft	Subtraction (double)

(Floating point) Load instructions

l.s fs, address	Load in fs the value of the float (32 bits) found in the specified address.
l.d fd, address	Load fd with the value of double (64 bits) found in the specified address.

(Floating point) Store instructions

s.s fs, address	Stores the fs register in the indicated address. (float)
s.d fd, address	Store a double (64 bits) in the indicated address, the value of 64 bits comes from fd.

Conversion instructions

cvt.d.s fd, fs	Turn a float into a double, the result is saved in fd
cvt.d.w fd, Rsrc	Convert an integer to a double, the result is saved in fd
cvt.s.d fd, fs	Turn a double into a float, the result is saved in fd
cvt.s.w fd, Rsrc	Convert an integer into a float, the result is saved in fd
cvt.w.s Rdest, fs	Convert a float into an integer, the result is saved in Rdest
cvt.w.d Rdest, fs	Convert a double into an integer, the result is saved in Rdest