The MIPS Info Sheet

MIPS Instructions

Arithmetic/Logic

In the instructions below, `Src2` can either be a register or an immediate value (integer). Many of these instructions have an unsigned version, obtained by appending `u` to the opcode (e.g. `addu`).

`abs Rdest, Rsrc` *Absolute Value*
Put the absolute value of the integer from register `Rsrc` in register `Rdest`.

`add Rdest, Rsrc1, Src2` *Addition (with overflow)*
Put the sum of the integers from register `Rsrc1` and `Src2` (or `Imm`) into register `Rdest`.

`and Rdest, Rsrc1, Src2` *AND*
Put the logical AND of the integers from register `Rsrc1` and `Src2` (or `Imm`) into register `Rdest`.

`div Rdest, Rsrc1, Src2` *Divide (with overflow)*
Put the quotient of the integers from register `Rsrc1` and `Src2` into register `Rdest`.

`mul Rdest, Rsrc1, Src2` *Multiply (without overflow)*
Put the product of the integers from register `Rsrc1` and `Src2` into register `Rdest`.

`neg Rdest, Rsrc` *Negate Value (with overflow)*
Put the negative of the integer from register `Rsrc` into register `Rdest`.

`nor Rdest, Rsrc1, Src2` *NOR*
Put the logical NOR of the integers from register `Rsrc1` and `Src2` into register `Rdest`.

`not Rdest, Rsrc` *NOT*
Put the bitwise logical negation of the integer from register `Rsrc` into register `Rdest`.

`or Rdest, Rsrc1, Src2` *OR*
Put the logical OR of the integers from register `Rsrc1` and `Src2` (or `Imm`) into register `Rdest`.

`rem Rdest, Rsrc1, Src2` *Remainder*
Put the remainder from dividing the integer in register `Rsrc1` by the integer in `Src2` into register `Rdest`.

`rol Rdest, Rsrc1, Src2` *Rotate Left*
Rotate the contents of register `Rsrc1` left (right) by the distance indicated by `Src2` and put the result in register `Rdest`.

`sll Rdest, Rsrc1, Src2` *Shift Left Logical*
`sra Rdest, Rsrc1, Src2` *Shift Right Arithmetic*
`srl Rdest, Rsrc1, Src2` *Shift Right Logical*
Shift the contents of register `Rsrc1` left (right) by the distance indicated by `Src2` (`Rsrc2`) and put the result in register `Rdest`.

`sub Rdest, Rsrc1, Src2` *Subtract (with overflow)*
Put the difference of the integers from register `Rsrc1` and `Src2` into register `Rdest`.

`xor Rdest, Rsrc1, Src2` *XOR*
Put the logical XOR of the integers from register `Rsrc1` and `Src2` (or `Imm`) into register `Rdest`.

Comparison Instructions

In all instructions below, `Src2` can either be a register or an immediate value (a 16 bit integer). Most instructions also have an unsigned version (append `u`).

`seq Rdest, Rsrc1, Src2` *Set Equal*
Set register `Rdest` to 1 if register `Rsrc1` equals `Src2` and to be 0 otherwise.

`sge Rdest, Rsrc1, Src2` *Set Greater Than Equal*
Set register `Rdest` to 1 if register `Rsrc1` is greater than or equal to `Src2` and to 0 otherwise.

`sgt Rdest, Rsrc1, Src2` *Set Greater Than*
Set register `Rdest` to 1 if register `Rsrc1` is greater than `Src2` and to 0 otherwise.

`sle Rdest, Rsrc1, Src2` *Set Less Than Equal*
Set register `Rdest` to 1 if register `Rsrc1` is less than or equal to `Src2` and to 0 otherwise.

`slt Rdest, Rsrc1, Src2` *Set Less Than*
Set register `Rdest` to 1 if register `Rsrc1` is less than `Src2` (or `Imm`) and to 0 otherwise.

`sne Rdest, Rsrc1, Src2` *Set Not Equal*
Set register `Rdest` to 1 if register `Rsrc1` is not equal to `Src2` and to 0 otherwise.

Branch and Jump Instructions

In all instructions below, `Src2` can either be a register or an immediate value (integer).

`b label` *Branch instruction*
Unconditionally branch to the instruction at the label.

`beq Rsrc1, Src2, label` *Branch on Equal*
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` equals `Src2`.

`bge Rsrc1, Src2, label` *Branch on Greater Than Equal*
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are greater than or equal to `Src2`.

`bgt Rsrc1, Src2, label` *Branch on Greater Than*
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are greater than `Src2`.

`ble Rsrc1, Src2, label` *Branch on Less Than Equal*
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are less than or equal to `Src2`.

`blt Rsrc1, Src2, label` *Branch on Less Than*
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are less than `Src2`.

`bne Rsrc1, Src2, label` *Branch on Not Equal*
Conditionally branch to the instruction at the label if the contents of register `Rsrc1` are not equal to `Src2`.

`jal label` *Jump and Link*
Unconditionally jump to the instruction at the label Save the address of the next instruction in register 31.

`jr Rsrc` *Jump Register*
Unconditionally jump to the instruction whose address is in register `Rsrc`.

Load/Store/Move Instructions

`move Rdest, Rsrc` *Move*
Move the contents of `Rsrc` to `Rdest`.

`li Rdest, imm` *Load Immediate*
Move the immediate value `imm` into register `Rdest`.

`la Rdest, address` *Load Address*
Load computed *address*, not the contents of the location, into register `Rdest`.

`lb Rdest, address` *Load Byte*
Load the byte at *address* into register `Rdest`.

`lh Rdest, address` *Load Halfword*
Load the 16-bit quantity (halfword) at *address* into register `Rdest`.

`lw Rdest, address` *Load Word*
Load the 32-bit quantity (word) at *address* into register `Rdest`.

`sb Rsrc, address` *Store Byte*
Store the low byte from register `Rsrc` at *address*.

`sh Rsrc, address` *Store Halfword*
Store the low halfword from register `Rsrc` at *address*.

`sw Rsrc, address` *Store Word*
Store the word from register `Rsrc` at *address*.

SPIM System Calls

Note that in MIPS assembler register `$v0` is synonymous with `$2`, register `$a0` is synonymous with `$4` and register `$a1` is synonymous with `$5`.

Service	`$v0`	Arguments	Result
print_int	1	`$4` = integer	
print_string	4	`$4` = string	
read_int	5		integer (in `$2`)
read_string	8	`$4` = buffer, `$5` = length	
sbrk	9	`$4` = amount	address (in `$2`)
exit	10		

MIPS Assembler Directives

`.align n`
Align data on a n-byte boundary.

`.asciiz str`
Store string in memory and null-terminate it.

`.data`
The following data items should be stored in the data segment.

`.space n`
Allocate *n* bytes of space in the current segment (which must be the data segment in SPIM).

`.text`
The next items are put in the user text segment.

`.word w1, ..., wn`
Store the *n* 32-bit quantities in successive memory words.