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  
Put the absolute value of the integer from register Rsrc in register Rdest.

add Rdest, Rsrd1, Src2  
Addition (with overflow)  
Put the sum of the integers from register Rsrd1 and Src2 (or Imm) into register Rdest.

and Rdest, Rsrd1, Src2  
AND  
Put the logical AND of the integers from register Rsrd1 and Src2 into register Rdest.

div Rdest, Rsrd1, Src2  
Divide (with overflow)  
Put the quotient of the integers from register Rsrd1 and Src2 into register Rdest.

mul Rdest, Rsrd1, Src2  
Multiply (without overflow)  
Put the product of the integers from register Rsrd1 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, Rsrd1, Src2  
NOR  
Put the logical NOR of the integers from register Rsrd1 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, Rsrd1, Src2  
OR  
Put the logical OR of the integers from register Rsrd1 and Src2 (or Imm) into register Rdest.

rem Rdest, Rsrd1, Src2  
Remainder  
Put the remainder from dividing the integer in register Rsrd1 by the integer in Src2 into register Rdest.

rol Rdest, Rsrd1, Src2  
Rotate Left  
Rotate the contents of register Rsrd1 left (right) by the distance indicated by Src2 and put the result in register Rdest.

sll Rdest, Rsrd1, Src2  
Shift Left Logical  
Shift the contents of register Rsrd1 left (right) by the distance indicated by Src2 (Rsrd2) and put the result in register Rdest.

sra Rdest, Rsrd1, Src2  
Shift Right Arithmetic  
Shift the contents of register Rsrd1 right (left) by the distance indicated by Src2 and put the result in register Rdest.

srl Rdest, Rsrd1, Src2  
Shift Right Logical  
Shift the contents of register Rsrd1 left (right) by the distance indicated by Src2 and put the result in register Rdest.

sub Rdest, Rsrd1, Src2  
Subtract (with overflow)  
Put the difference of the integers from register Rsrd1 and Src2 into register Rdest.

xor Rdest, Rsrd1, Src2  
XOR  
Put the logical XOR of the integers from register Rsrd1 and Src2 (or Imm) into register Rdest.

Comparison Instructions

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

seq Rdest, Rsrd1, Src2  
Set Equal  
Set register Rdest to 1 if register Rsrd1 equals Src2 and 0 otherwise.

sge Rdest, Rsrd1, Src2  
Set Greater Than Equal  
Set register Rdest to 1 if register Rsrd1 is greater than or equal to Src2 and to 0 otherwise.

sgt Rdest, Rsrd1, Src2  
Set Greater Than  
Set register Rdest to 1 if register Rsrd1 is greater than Src2 and to 0 otherwise.

sle Rdest, Rsrd1, Src2  
Set Less Than Equal  
Set register Rdest to 1 if register Rsrd1 is less than or equal to Src2 and to 0 otherwise.

slt Rdest, Rsrd1, Src2  
Set Less Than  
Set register Rdest to 1 if register Rsrd1 is less than Src2 and to 0 otherwise.

sne Rdest, Rsrd1, Src2  
Set Not Equal  
Set register Rdest to 1 if register Rsrd1 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 Rsrd1, Src2, label  
Branch on Equal  
Conditionally branch to the instruction at the label if the contents of register Rsrd1 equals Src2.

bge Rsrd1, Src2, label  
Branch on Greater Than Equal  
Conditionally branch to the instruction at the label if the contents of register Rsrd1 are greater than or equal to Src2.

bgt Rsrd1, Src2, label  
Branch on Greater Than  
Conditionally branch to the instruction at the label if the contents of register Rsrd1 are greater than Src2.

ble Rsrd1, Src2, label  
Branch on Less Than Equal  
Conditionally branch to the instruction at the label if the
the contents of register $rsrc1$ are less than or equal to $src2$.

```mips
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 $src2$.

```mips
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 $src2$.

```mips
jal label
```

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

```mips
jr $rsrc$
```

Jump Register
Unconditionally jump to the instruction whose address is in register $rsrc$.

### Load/Store/Move Instructions

```mips
move $rdest$, $rsrc$
```

Move
Move the contents of $rsrc$ to $rdest$.

```mips
li $rdest$, imm
```

Load Immediate
Move the immediate value $imm$ into register $rdest$.

```mips
la $rdest$, address
```

Load Address
Load computed $address$, not the contents of the location, into register $rdest$.

```mips
lb $rdest$, address
```

Load Byte
Load the byte at $address$ into register $rdest$.

```mips
lh $rdest$, address
```

Load Halfword
Load the 16-bit quantity (halfword) at $address$ into register $rdest$.

```mips
lw $rdest$, address
```

Load Word
Load the 32-bit quantity (word) at $address$ into register $rdest$.

```mips
sb $rsrc$, address
```

Store Byte
Store the low byte from register $rsrc$ at $address$.

```mips
sh $rsrc$, address
```

Store Halfword
Store the low halfword from register $rsrc$ at $address$.

```mips
sw $rsrc$, address
```

Store Word
Store the word from register $rsrc$ at $address$.

### MIPS Assembler Directives

```mips
.align n
```

Align data on a n-byte boundary.

```mips
.asciiz str
```

Store string in memory and null-terminate it.

```mips
.data
```

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

```mips
.space n
```

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

```mips
.text
```

The next items are put in the user text segment.

```mips
.word w1, ..., wn
```

Store the $n$ 32-bit quantities in successive memory words.

### SPIM System Calls

<table>
<thead>
<tr>
<th>Service</th>
<th>$2$</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_int</td>
<td>1</td>
<td>$4 = integer</td>
<td>integer (in $2$)</td>
</tr>
<tr>
<td>print_string</td>
<td>4</td>
<td>$4 = string</td>
<td></td>
</tr>
<tr>
<td>read_int</td>
<td>5</td>
<td>$4 = buffer, $5 = length</td>
<td>address (in $2$)</td>
</tr>
<tr>
<td>read_string</td>
<td>8</td>
<td>$4 = buffer, $5 = length</td>
<td></td>
</tr>
<tr>
<td>sbrk</td>
<td>9</td>
<td>$4 = amount</td>
<td></td>
</tr>
<tr>
<td>exit</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>