**MIPS Assembly Instructions**

**Arithmetic & Logical Instructions**
- `abs` Rdest, Rsr Absolute Value y
- `add` Rdest, Rsr1, Src2 Addition (with overflow)
- `addi` Rdest, Rsr1, Imm Addition Immediate (with overflow)
- `addu` Rdest, Rsr1, Src2 Addition (without overflow)
- `addiu` Rdest, Rsr1, Imm Addition Immediate (without overflow)
- `and` Rdest, Rsr1, Src2 AND
- `andi` Rdest, Rsr1, Imm AND Immediate
- `div` Rsr1, Rsr2 Divide (signed)
- `divu` Rsr1, Rsr2 Divide (unsigned)
- `divu` Rdest, Rsr1, Src2 Divide (unsigned, without overflow)
- `mul` Rdest, Rsr1, Src2 Multiply (without overflow)
- `mulu` Rdest, Rsr1, Src2 Multiply (with overflow)
- `mulu` Rsr1, Rsr2 Unsigned Multiply
- Multiply the contents of the two registers. Leave the low-order word of the product in register Rdest.
- `neg` Rdest, Rsr Negate Value (with overflow)
- `negu` Rdest, Rsr Negate Value (without overflow)
- `nor` Rdest, Rsr1, Src2 NOR
- `not` Rdest, Rsr NOT y
- `or` Rdest, Rsr1, Src2 OR
- `ori` Rdest, Rsr1, Imm OR Immediate
- `rem` Rdest, Rsr1, Src2 Remainder y
- `remu` Rdest, Rsr1, Src2 Unsigned Remainder
- Put the remainder from dividing the integer in register Rsr1 by the integer in Src2 into register Rdest.
- `rol` Rdest, Rsr1, Src2 Rotate Left
- `ror` Rdest, Rsr1, Src2 Rotate Right
- `sll` Rdest, Rsr1, Src2 Shift Left Logical
- `sllv` Rdest, Rsr1, Src2 Shift Left Logical Variable
- `sra` Rdest, Rsr1, Src2 Shift Right Arithmetic
- `sra` Rdest, Rsr1, Src2 Shift Right Arithmetic Variable
- `srav` Rdest, Rsr1, Src2 Shift Right Logical
- `srl` Rdest, Rsr1, Src2 Shift Right Logical Variable
- `srlv` Rdest, Rsr1, Src2 Shift Right Logical Variable
- `sub` Rdest, Rsr1, Src2 Subtract (with overflow)
- `subu` Rdest, Rsr1, Src2 Subtract (without overflow)
- `xor` Rdest, Rsr1, Src2 XOR
- `xori` Rdest, Rsr1, Imm XOR Immediate
- Set register Rdest to 1 if register Rsr1 is greater than Src2 and to 0 otherwise.
- `slt` Rdest, Rsr1, Src2 Set Less Than
- `slt` Rdest, Rsr1, Src2 Set Less Than Immediate
- `sltu` Rdest, Rsr1, Src2 Set Less Than Unsigned
- `sltu` Rdest, Rsr1, Imm Set Less Than Unsigned Immediate
- Set register Rdest to 1 if register Rsr1 is less than Src2 (or Imm) and to 0 otherwise.
- `sne` Rdest, Rsr1, Src2 Set Not Equal
- Set register Rdest to 1 if register Rsr1 is not equal to Src2 and to 0 otherwise.

**Branch and Jump Instructions**
- `b` label Branch instruction y
- Unconditionally branch to the instruction at the label.
- `bcz` label Branch Coprocessor z True
- `bczf` label Branch Coprocessor z False
- Conditionally branch to the instruction at the label if coprocessor z's condition flag is true (false).
- `beq` Rsr1, Src2, label Branch on Equal
- `beqz` Rsr, label Branch on Equal Zero
- Conditionally branch to the instruction at the label if the contents of register Rsr1 equals Src2.
- `beqz` Rsr, label Branch on Equal Zero y
- Conditionally branch to the instruction at the label if the contents of register Rsr equals 0.
- `bge` Rsr1, Src2, label Branch on Greater Than Equal
- `bgez` Rsr, label Branch on Greater Than Equal Zero
- Conditionally branch to the instruction at the label if the contents of register Rsr1 are greater than or equal to Src2.
- `bgez` Rsr, label Branch on Greater Than Equal Zero
- Conditionally branch to the instruction at the label if the contents of register Rsr1 are greater than or equal to 0.
- `bgezal` Rsr, label Branch on Greater Than Equal Zero And Link
- Conditionally branch to the instruction at the label if the contents of register Rsr1 are greater than or equal to 0. Save the address of the next instruction in register 31.
- `bgt` Rsr1, Src2, label Branch on Greater Than
- `bgt` Rsr1, Src2, label Branch on Greater Than Zero
- Conditionally branch to the instruction at the label if the contents of register Rsr1 are greater than Src2.
- `bgtz` Rsr, label Branch on Greater Than Zero
- Conditionally branch to the instruction at the label if the contents of register Rsr are greater than 0.
- `ble` Rsr1, Src2, label Branch on Less Than Equal
- `bleu` Rsr1, Src2, label Branch on LTE Unsigned
- Conditionally branch to the instruction at the label if the contents of register Rsr1 are less than or equal to Src2.
- `blez` Rsr, label Branch on Less Than Equal Zero
- Conditionally branch to the instruction at the label if the contents of register Rsr are less than or equal to 0.
- `bgezal` Rsr, label Branch on Greater Than Equal Zero And Link
- `bgtzal` Rsr, label Branch on Greater Than Equal Zero And Link
- Conditionally branch to the instruction at the label if the contents of register Rsr are greater or equal to 0 or less than 0, respectively. Save the address of the next instruction in register 31.
- `bl` Rsr1, Src2, label Branch on Less Than
- `blt` Rsr1, Src2, label Branch on Less Than Unsigned

**Constant-Manipulating Instructions**
- `li` Rdest, imm Load Immediate y
- `lui` Rdest, imm Load Upper Immediate

**Comparison Instructions**
- `seq` Rdest, Rsr1, Src2 Set Equal
- Set register Rdest to 1 if register Rsr1 equals Src2 and to 0 otherwise.
- `sge` Rdest, Rsr1, Src2 Set Greater Than Equal
- `sgeu` Rdest, Rsr1, Src2 Set Greater Than Equal Unsigned y
- Set register Rdest to 1 if register Rsr1 is greater than or equal to Src2 and to 0 otherwise.
- `sgt` Rdest, Rsr1, Src2 Set Greater Than
- `sgtu` Rdest, Rsr1, Src2 Set Greater Than Unsigned
- Set register Rdest to 1 if register Rsr1 is greater than Src2 and to 0 otherwise.
- `sle` Rdest, Rsr1, Src2 Set Less Than Equal y
- `sleu` Rdest, Rsr1, Src2 Set Less Than Equal Unsigned y
- `slt` Rdest, Rsr1, Src2 Set Less Than
- `sltu` Rdest, Rsr1, Src2 Set Less Than Unsigned
- `sltu` Rdest, Rsr1, Imm Set Less Than Immediate
- `sltu` Rdest, Rsr1, Imm Set Less Than Unsigned Immediate
- `sltu` Rdest, Rsr1, Imm Set Less Than Unsigned Immediate
- Set register Rdest to 1 if register Rsr1 is less than Src2 (or Imm) and to 0 otherwise.
- `sne` Rdest, Rsr1, Src2 Set Not Equal
- Set register Rdest to 1 if register Rsr1 is not equal to Src2 and to 0 otherwise.
MIPS Assembly Instructions

Conditional branch to the instruction at the label if the contents of register Rsrc1 are less than Src2.

bltz Rsrc, label Branch on Less Than Zero
Conditionally branch to the instruction at the label if the contents of Rsrc are less than 0.

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.

bnez Rsrc, label Branch on Not Equal Zero
Conditionally branch to the instruction at the label if the contents of Rsrc are not equal to 0.

j label Jump
Unconditionally jump to the instruction at the label.

jal label Jump and Link
Jalr Rsrc Jump and Link Register
Unconditionally jump to the instruction at the label or whose address is in register Rsrc. 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 Instructions

la Rdest, address Load Address y
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. The byte is sign-extended by the lb, but not the lbu, instruction.

lbu Rdest, address Load Unsigned Byte
Load the byte at address into register Rdest. The byte is sign-extended by the lbu, but not the lb, instruction.

ld Rdest, address Load Double-Word
Load the 64-bit quantity at address into registers Rdest and Rdest + 1.

lh Rdest, address Load Halfword
Load the 16-bit quantity (halfword) at address into register Rdest. The halfword is sign-extended by the lh, but not the lhu, instruction.

lhu Rdest, address Load Unsigned Halfword
Load the 16-bit quantity (halfword) at address into register Rdest. The halfword is sign-extended by the lhu, instruction.

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

lwz Rdest, address Load Word Zero
Load the 32-bit quantity (word) at address into register Rdest. The word is sign-extended by the lw, but not the lwl, instruction.

lwcz Rdest, address Load Word Coprocessor
Load the word at address into register Rdest of coprocessor z (0–3).

lw1 Rdest, address Load Word Left
Load the left (right) bytes from the word at the possibly-unaligned address into register Rdest.

lw2 Rdest, address Load Word Right
Load the left (right) bytes from the word at the possibly-unaligned address into register Rdest.

ulh Rdest, address Unaligned Load Halfword
Load the 16-bit quantity (halfword) at the possibly-unaligned address into register Rdest. The halfword is sign-extended by the ulh, but not the ulhu, instruction.

ulhu Rdest, address Unaligned Load Halfword Unsigned
Load the 16-bit quantity (halfword) at the possibly-unaligned address into register Rdest. The halfword is sign-extended by the ulh, but not the ulhu, instruction.

ulw Rdest, address Unaligned Load Word
Load the 32-bit quantity (word) at the possibly-unaligned address into register Rdest.

Store Instructions

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

sd Rsrc, address Store Double-Word y
Store the 64-bit quantity in registers Rsrc and Rsrc + 1 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.

swcz Rsrc, address Store Word Coprocessor
Store the word from register Rsrc of coprocessor z at address.

swl Rsrc, address Store Word Left
Store the left (right) bytes from register Rsrc at the possibly-unaligned address.

ush Rsrc, address Unaligned Store Halfword
Store the low halfword from register Rsrc at the possibly-unaligned address.

usw Rsrc, address Unaligned Store Word
Store the word from register Rsrc at the possibly-unaligned address.

Data Movement Instructions

move Rdest, Rsrc Move y
Move the contents of Rsrc to Rdest.

The multiply and divide unit produces its result in two additional registers, hi and lo. These instructions move values to and from these registers. The multiply, divide, and remainder instructions described above are pseudoinstructions that make it appear as if this unit operates on the general registers and detect error conditions such as divide by zero or overflow.

mthi Rdest Move From hi
Move the contents of the hi (lo) register to register Rdest.

mflo Rdest Move From lo
Move the contents register Rdest to the hi (lo) register.

Coprocessors have their own register sets. These instructions move values between these registers and the CPU’s registers.

mfcz Rdest, CPsrc Move From Coprocessor z
Move the contents of coprocessor z’s register CPsrc to CPU register Rdest.

mfc1.d Rdest, FRsrc1 Move Double From Coprocessor 1
Move the contents of floating point registers FRsrc1 and FRsrc1 + 1 to CPU registers Rdest and Rdest + 1.

mtcz Rsrc, CPdest Move To Coprocessor z
Move the contents of CPU register Rsrc to coprocessor z’s register CPdest.

System Call Interface

print int 1
Sa0 = integer
print float 2
$F12 = float
print double 3
$F12 = double
print string 4
Sa0 = string
read int 5
integer (in $v0)
read float 6
float (in $f0)
read double 7
double (in $f0)
read string 8
Sa0 = buffer, $a1 = length
sbkr 9
Sa0 = amount address (in $v0)
exit 10

.align n
Align the next datum on a 2 n byte boundary. For example, .align 2 aligns the next value on a word boundary. .align 0 turns off
automatic alignment of .half, .word, .float, and .double directives until the next .data or .kdata directive.

.ASCII str
Store the string in memory, but do not null-terminate it.

.ASCIIIZ str
Store the string in memory and null-terminate it.

.BYTE b1, ..., bn
Store the n values in successive bytes of memory.

.DATA <addr>
The following data items should be stored in the data segment. If the optional argument addr is present, the items are stored beginning at address addr.

.DOUBLE d1, ..., dn
Store the n floating point double precision numbers in successive memory locations.

.EXTERN sym size
Declare that the datum stored at sym is size bytes large and is a global symbol. This directive enables the assembler to store the datum in a portion of the data segment that is efficiently accessed via register $gp.

.FLOAT f1, ..., fn
Store the n floating point single precision numbers in successive memory locations.

.GLOBL sym
Declare that symbol sym is global and can be referenced from other files.

.HALF h1, ..., hn
Store the n 16-bit quantities in successive memory halfwords.

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

.TEXT <addr>
The next items are put in the user text segment. In SPIM, these items may only be instructions or words (see the .word directive below). If the optional argument addr is present, the items are stored beginning at address addr.

.WORD w1, ..., wn
Store the n 32-bit quantities in successive memory words.