MIPS INSTRUCTIONS

ARITHMETIC AND LOGICAL INSTRUCTIONS

- **add –** Adds 2 integer values such that $t2 = $t0 + $t1 `(add  $t2, $t0, $t1)`
- **addi –** Adds 2 integer values including an immediate value (a number given directly) such that $s0 = $t0 + *value* `(add  $t2, $t0, value)`
- **add.d –** Add 2 double values such that $f12 = $f0 + $f2 `(add.d  $f12, $f0, $f2)`
- **add.s –** Add single precision. Adds 2 float values such that $f12 = $f0 + $f2 `(add.d  $f12, $f0, $f2)`
- **sub –** Subtracts 2 integer values such that $t2 = $t0 - $t1 `(sub  $t2, $t0, $t1)`
- **mul –** Multiplies 2 integer values such that $t2 = $t0 * $t1. However, you aare limited to only being able to multiply 2 16 bit numbers and thus are restricted to a smaller range of numbers. `(mul  $t2, $t0, $t1)`
- **mult –** Multiplies 2 integer values and saves them over two registers **lo** and **hi**. `(mul  $t2, $t0, $t1)`
- **div –** Divides 2 integer values such that $s0 = $t0 / $t1 or can the quotient and remainder can be respetively retreived from the lo and hi registers. `(div  $s0, $t0, $t1 or div  $t0, $t1)`
- **slt –** Set less than. Takes 2 arguments and returns 1 if $t0 < $t1, and 0 otherwise. `(slt  $s0, $t0, $t1)`
- **slti –** Set less than immediate. Takes 2 arguments and returns 1 if $t0 < i, and 0 otherwise. `(slti  $s0, $t0, i)`
- **sll –** Shift Left Logical. Can be used to do multiplication more efficiently such that $t0 = $s0 * 2^i `(sll  $t0, $s0, i)`
- **srl –** Shift Right Logical. Can be used to do division more efficiently such that $t0 = $s0 / 2^i `(srl  $t0, $s0, i)`
-

CONDITIONAL STATEMENT INSTRUCTIONS

- **beq –** Branch if equal. Equivalent to if both equal, then... Takes two comparison values, and then a function label. `(beq  $t0, $t1, label)`
- **bne –** Branch if not equal. If two comparison values are not equal, then the program proceeds onto the specified function `(bne  $t0, $t1, label)`
- **b –** Branch unconditionally. Goes to the specified function unconditionally. `(b label)`
- **bgt –** Branch if greater than. Goes to the specified function if $s0 > $s1 `(bgt  $s0, $s1, label)`
- **blt –** Branch if less than. Goes to the specified function if $s0 < $s1 `(blt  $s0, $s1, label)`
- **bgtz –** Branch if greater than zero. Goes to the specified function if $s0 > 0 `(bgt  $s0, label)`

MIPS INSTRUCTIONS

LOAD INSTRUCTIONS

- **li –** Load Immediate. Pseudo-instruction. Can load immediates up to 32 bits in size, compared to the likes of **addi** or **ori** which can only encode 16 -bit immediates. **(**`li $t0,0x12345678)`
- **la –** Load Address. Pseudo-instruction. Can be used to load integer constants just like **li**, **(**`la $t0,0x123467)` but also works with labels **(**`la $t0, Message # t0 = address of Message)`
- **lw –** Load Word. Copy word (4 bytes) at source RAM location to destination register. `(lw register_destination, RAM_source)`
- **lwc1 –** Load Word from Coprocessor 1. Copy word (4 bytes) at source RAM location to destination register. This is used for Floats rather than integers. `(lwc1 $f12, RAM_source)`
- **ldc1 –** Load Double from Coprocessor 1. This is used for doubles rather than integers. `(ldc1 $f2, RAM_source)`
- **lb –** Load Byte. Copy byte at source RAM location to low-order byte of destination register, and sign-e.g.tend to higher-order bytes `(lb register_destination, RAM_source)`
- **jal –** Jump and link. Calls a function. `(jal function_name)`
- **jr –** Jump register unconditionally. jr $ra will return you to the previous register you called the function from. `(jr register)`
- **j –** Jump unconditionally. Jumps to the target function. `(j label)`

STORE INSTRUCTIONS

- **move –** Pseudo-instruction. Moves a value from one register to another. **(**`move $a0, $t0)`
- **mflo –** Moves a the **lo** register value into the specified register **(**`mflo $s0)`
- **mfhi –** Moves a the **hi** register value into the specified register **(**`mfhi $s0)`
- **sw –** Store Word. Moves a word from a register into RAM **(**`sw $s0, offset($sp))`
- **seq –** Set if equal. If two values are equal, then the given register is set to 1 else 0. `(seq $t2, $t0, $t1)`
-

STORAGE TYPES FOR DATA DECLARACTIONS

- **.word –** Integer variables `(var1: .word 3)`
- **.asciiz –** String variables `(Message: .asciiz "Hello World)`
- **.byte –** Character variables`(chars: .byte 'a', 'b')`
- **.float –** Float variables (32 bit) `(PI: .float 3.14)`
- **.double –** Double variables (64 bit) `(PI: .double 3.14159265359)`
- **.space –**Essentially like declaring an array. You decide how many bytes the input can hold max. `(firstName: .space 20)`

MIPS INSTRUCTIONS

SYSTEM CALL CODES

Service	System Call Code	Arguments	Result
print integer	1	$a0 = value	(none)
print float	2	$f12 = float value	(none)
print double	3	$f12 = double value	(none)
print string	4	$a0 = address of string	(none)
read integer	5	(none)	$v0 = value read
read float	6	(none)	$f0 = value read
read double	7	(none)	$f0 = value read
read string	8	$a0 = address where string to be stored $a1 = number of characters to read + 1	(none)
memory allocation	9	$a0 = number of bytes of storage desired	$v0 = address of block
exit (end of program)	10	(none)	(none)
print character	11	$a0 = integer	(none)
read character	12	(none)	char in $v0

- **print int** passes an integer and prints it on the console.
- **print float** prints a single floating point number.
- **print double** prints a double precision number.
- **print string** passes a pointer to a null-terminated string
- **read int**, **read float**, and **read double** read an entire line of input up to and including a newline.
- **read string** reads up to n - 1 characters into a buffer and terminates the string with a null byte. If there are fewer characters on the current line, it reads through the newline and again null-terminates the string.
- **sbrk** returns a pointer to a block of memory containing *n* additional bytes.
- **exit** stops a program from running.
- **syscall** is used to request a service from the kernel (above table of values). I.e.:

```
li $v0, 1
add $a0, $t0, $zero
syscall
```

In this case, syscall has been used to print out an integer value. 1 is the service code for print integer. The second instruction copies from $t0 to $a0 which is the designated register where the argument is held (where the integer to be printed is).