Common MIPS instructions

Notes: *op, funct, rd, rs, rt, imm, address, shamt* refer to fields in the instruction format
PC is assumed to point to the next instruction, **Mem** is the byte addressed main memory

Assembly Instruction	Instr Format	op op/funct	Meaning	Comments
add *$rd, $rs, $rt*	R	0/32	*$rd = $rs + $rt*	Add contents of two registers
sub *$rd, $rs, $rt*	R	0/34	*$rd = $rs - $rt*	Subtract contents of two registers
addi *$rt, $rs, imm*	I	8	*$rt = $rs + imm*	Add signed constant
addu *$rd, $rs, $rt*	R	0/33	*$rd = $rs + $rt*	Unsigned, no overflow
subu *$rd, $rs, $rt*	R	0/35	*$rd = $rs - $rt*	Unsigned, no overflow
addiu *$rt, $rs, imm*	I	9	*$rt = $rs + imm*	Unsigned, no overflow
mfc0 *$rt, $rd*	R	16	*$rt = $rd*	*rd* = coprocessor register (e.g. epc, cause, status)
mult *$rs, $rt*	R	0/24	Hi, Lo *= $rs * $rt*	64 bit signed product in Hi and Lo
multu *$rs, $rt*	R	0/25	Hi, Lo *= $rs * $rt*	64 bit unsigned product in Hi and Lo
div *$rs, $rt*	R	0/26	Lo *= $rs / $rt,* Hi *= $rs* mod *$rt*	
divu *$rs, $rt*	R	0/27	Lo *= $rs / $rt,* Hi *= $rs* mod *$rt* (unsigned)	
mfhi *$rd*	R	0/16	*$rd =* Hi	Get value of Hi
mflo *$rd*	R	0/18	*$rd =* Lo	Get value of Lo
and *$rd, $rs, $rt*	R	0/36	*$rd = $rs & $rt*	Logical AND
or *$rd, $rs, $rt*	R	0/37	*$rd = $rs | $rt*	Logical OR
andi *$rt, $rs, imm*	I	12	*$rt = $rs & imm*	Logical AND, unsigned constant
ori *$rt, $rs, imm*	I	13	*$rt = $rs | imm*	Logical OR, unsigned constant
sll *$rd, $rs, shamt*	R	0/0	*$rd = $rs << shamt*	Shift left logical (shift in zeros)
srl *$rd, $rs, shamt*	R	0/2	*$rd = $rs >> shamt*	Shift right logical (shift in zeros)
lw *$rt, imm($rs)*	I	35	*$rt = Mem[$rs + imm]*	Load word from memory
sw *$rt, imm($rs)*	I	43	*Mem[$rs + imm] = $rt*	Store word in memory
lbu *$rt, imm($rs)*	I	37	*$rt = Mem[$rs + imm]*	Load a single byte, set bits 8-31 of *$rt* to zero
sb *$rt, imm($rs)*	I	41	*Mem[$rs + imm] = $rt*	Store byte (bits 0-7 of *$rt*) in memory
lui *$rt, imm*	I	15	*$rt = imm * 2^{16}*	Load constant in bits 16-31 of register *$rt*
beq *$rs, $rt, imm*	I	4	if*($rs==$rt)* PC *=* PC *+ imm* (PC always points to next instruction)	
bne *$rs, $rt, imm*	I	5	if*($rs!=$rt)* PC *=* PC *+ imm* (PC always points to next instruction)	
slt *$rd, $rs, $rt*	R	0/42	if*($rs<$rt) $rd = 1;* else *$rd = 0*	
slti *$rt, $rs, imm*	I	10	if*($rs<imm) $rt = 1;* else *$rt = 0*	
sltu *$rd, $rs, $rt*	R	0/43	if*($rs<$rt) $rd = 1;* else *$rd = 0* (unsigned numbers)	
sltiu *$rt, $rs, imm*	I	11	if*($rs<$rt) $rd = 1;* else *$rd = 0* (unsigned numbers)	
j *destination*	J	2	PC *= address*4*	Jump to *destination*, *address = destination/4*
jal *destination*	J	3	*$ra =* PC*;* PC *= address*4* (Jump and link, *address = destination/4*)	
jr *$rs*	R	0/8	PC *= $rs*	Jump to address stored in register *$rs*
beqz $rs, label	Pseudo		If(*$rs==0*) then goto label	See also: *bnez, bgez, bgtz, blez, bltz*
bge $rs, $rt, label	Pseudo		If(*$rs≥$rt)* then goto label	See also: *bgt, ble, blt* (add *u* for *unsigned*,eg *bgeu*)
la $rd, label	Pseudo		*$rd* = label	Assign the address of the label to register *$rd*

MIPS Instruction formats

Format	Bits 31-26	Bits 25-21	Bits 20-16	Bits 15-11	Bits 10-6	Bits 5-0
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	imm		
J	op	address				

MIPS registers

register		usage	writable	caller-	callee-
-name	-nr			saved	
$zero	0	stores the value 0, **do not write to it!**	yes	-	-
$at	1	reserved for assembler	no	-	-
$v0 - $v1	2 - 3	stores function results, more than 2 results → stack	yes	x	-
$a0 - $a3	4 - 7	function arguments, if more than 4 arguments are needed → stack	yes	x	-
$t0 - $t9	8 -15, 24-25	temporary variables	yes	x	-
$s0 - $s7	16 - 23	long-living variables	yes	-	x
$k0 - $k1	26 - 27	reserved for kernel	no	-	-
$gp	28	points to middle of a 64K block in the data segm.	no	-	-
$sp	29	stack pointer (top of stack)	yes	-	-
$fp	30	frame pointer (beginning of current frame)	yes	-	x
$ra	31	return address	yes	-	x
internal, not directly accessible, registers					
Hi, Lo		stores the result of mult/div operations (use *mflo*, *mfhi* to access these registers)			
PC		contains the address of the next instruction to be fetched			
status		register 12 in coprocessor 0, stores interrupt mask and enable bits (use *mfc0*)			
cause		register 13 in coprocessor 0, stores exception type and pending interrupt bits (use *mfc0*)			
epc		register 14 in coprocessor 0, stores address of instruction causing exception (use *mfc0*)			

operating system functions

function	code in $v0	arguments	result
print_int	1	$a0	
print_float	2	$f12	
print_double	3	$f12/13	
print_string	4	$a0 contains the start address of the string	
read_int	5		$v0
read_float	6		$f0
read_double	7		$f0/1
read_string	8	$a0 contains the destination address of the string, $a1 ist maximum length	string starting at $a0
sbrk	9	$a0 contains needed size	$v0 contains start address of the memory area
exit	10		

MIPS Assembler Syntax

```
                                    # This is a comment
        .data                       # Store following data in the data segment

items:                              # This is a label connected to the next address in the
                                    # current segment
        .word 1, 2                  # Stores the values 1 and 2 in next two words

servus: .ascii “servus!!”           # Stores a not terminated string in memory
hello:  .asciiz “hello”             # Stores ‘\0’ terminated string in memory (hello+ \0’)
                                    # Note that printing the label servus will give you
                                    # the text “servus!!hello”
.text                               # Store following instructions in the text segment
.globl main                         # the label main is the entry point of our program
main:   la $t0, servus              # An instruction connected to a label (e.g. for loops)
        addi $t0, $zero, ‘a’        # assigns the ascii value of ‘a’ to $t0
```