MIPS: Logic and arithmetic instructions:

MIPS instruction		Meaning	
Mnemonic	Operands	Description	Operation
sll	$rd, $rt, num	Shift left logical	$rd←$rt≪num
srl	$rd, $rt, num	Shift right logical	$rd←$rt≫num
sra	$rd, $rt, num	Shift right arithmetic	$rd←$rt≫num+msb
sllv	$rd, $rt, $rs	Shift left logic. var.	$rd←$rt≪$rs
srlv	$rd, $rt, $rs	Shift right logic. var.	$rd←$rt≫$rs
srav	$rd, $rt, $rs	Shift right arithm. var.	$rd←$rt≫$rs+msb
num: (decimal) 0...31			
add	$rd, $rt, $rs	Add	$rd←$rt+$rs
addu	$rd, $rt, $rs	Add unsigned	$rd←$rt+$rs
sub	$rd, $rt, $rs	Subtract	$rd←$rt-$rs
subu	$rd, $rt, $rs	Subtract unsigned	$rd←$rt-$rs
addi	$rd, $rt, imm	Add immediate	$rd←$rt+imm
addiu	$rd, $rt, imm	Add imm. unsign.	$rd←$rt+imm
mul	$rd, $rt, $rs	Multiply	$rd←$rt*$rs
mult	$rt, $rs	Multiply	$hi,$lo←$rt*$rs
multu	$rt, $rs	Multiply unsigned	$hi,$lo←$rt*$rs
div	$rt, $rs	Divide	$lo←$rt/$rs
			$hi←$rt%$rs
divu	$rt, $rs	Divide unsigned	$lo←$rt/$rs
			$hi←$rt%$rs
imm: halfword			
and	$rd, $rt, $rs	AND	$rd←$rs&$rt
or	$rd, $rt, $rs	OR	$rd←$rt|$rs
nor	$rd, $rt, $rs	NOR	$rd←!($rt|$rs)
xor	$rd, $rt, $rs	XOR	$rd←$rt^$rs
andi	$rd, $rt, imm	AND immediate	$rd←$rt&imm
ori	$rd, $rt, imm	OR immediate	$rd←$rt|imm
xori	$rd, $rt, imm	XOR immediate	$rd←$rt^imm
imm: halfword			
slt	$rd, $rt, $rs	Set (1) if less than*	$rd←($rt<$rs)?1:0
sltu	$rd, $rt, $rs	slt unsigned	$rd←($rt<$rs)?1:0
slti	$rd, $rt, imm	slt immediate	$rd←($rt<imm)?1:0
sltiu	$rd, $rt, imm	slt imm. unsigned	$rd←($rt<imm)?1:0
imm: halfword. *: Unset (0) otherwise			

MIPS: Logic and arithmetic instructions:

MIPS instruction		Machine code					
Mnemonic	Operands	format	opcode/ func	$A	$B	$C	imm
sll	$rd, $rt, num	R	00/00	0	$rt	$rd	num
srl	$rd, $rt, num	R	00/02	0	$rt	$rd	num
sra	$rd, $rt, num	R	00/03	0	$rt	$rd	num
sllv	$rd, $rt, $rs	R	00/04	$rs	$rt	$rd	0
srlv	$rd, $rt, $rs	R	00/06	$rs	$rt	$rd	0
srav	$rd, $rt, $rs	R	00/07	$rs	$rt	$rd	0
num: (decimal) 0...31							
add	$rd, $rt, $rs	R	00/20	$rt	$rs	$rd	0
addu	$rd, $rt, $rs	R	00/21	$rt	$rs	$rd	0
sub	$rd, $rt, $rs	R	00/22	$rt	$rs	$rd	0
subu	$rd, $rt, $rs	R	00/23	$rt	$rs	$rd	0
addi	$rd, $rt, imm	I	08/-	$rt	$rd	-	imm
addiu	$rd, $rt, imm	I	09/-	$rt	$rd	-	imm
mul	$rd, $rt, $rs	R	1c/20	$rt	$rs	$rd	0
mult	$rt, $rs	R	00/18	$rt	$rs	0	0
multu	$rt, $rs	R	00/19	$rt	$rs	0	0
div	$rt, $rs	R	00/1a	$rt	$rs	0	0
divu	$rt, $rs	R	00/1b	$rt	$rs	0	0
imm: halfword							
and	$rd, $rt, $rs	R	00/24	$rt	$rs	$rd	0
or	$rd, $rt, $rs	R	00/25	$rt	$rs	$rd	0
nor	$rd, $rt, $rs	R	00/27	$rt	$rs	$rd	0
xor	$rd, $rt, $rs	R	00/26	$rt	$rs	$rd	0
andi	$rd, $rt, imm	I	0c/-	$rt	$rd	-	imm
ori	$rd, $rt, imm	I	0d/-	$rt	$rd	-	imm
xori	$rd, $rt, imm	I	0e/-	$rt	$rd	-	imm
imm: halfword							
slt	$rd, $rt, $rs	R	00/2a	$rt	$rs	$rd	0
sltu	$rd, $rt, $rs	R	00/2b	$rt	$rs	$rd	0
slti	$rd, $rt, imm	I	0a/-	$rt	$rd	-	imm
sltiu	$rd, $rt, imm	I	0b/-	$rt	$rd	-	imm
imm: halfword							

MIPS: Jump, branch and memory instructions:

MIPS instruction		Meaning	
Mnemonic	Operands	Description	Operation
j	addr	Jump	pc←addr
jal	addr	Jump and link	pc←addr,$ra←pc+4
jr	$rs	Jump register	pc←$rs
jalr	$rs, $rd	Jump and link register	pc←$rs,$rd←pc+4
beq	$rt, $rs, addr	Branch if equal	$rs==$rt?pc←addr
bne	$rt, $rs, addr	Branch if not equal	$rs!=$rt?pc←addr
bltz	$rt, addr	Branch if < 0	$rt<0?pc←addr
bgtz	$rt, addr	Branch if > 0	$rt>0?pc←addr
blez	$rt, addr	Branch if $<= 0$	$rt<=0?pc←addr
bgez	$rt, addr	Branch if $>= 0$	$rt>=0?pc←addr
mfhi	$rd	Move from HI	$rd←hi
mthi	$rs	Move to HI	$rs→hi
mflo	$rd	Move from LO	$rd←lo
mtlo	$rs	Move to LO	$rs→lo
lb	$rd, offset($rt)	Load byte	$rd←M[$rt+offset]
lbu	$rd, offset($rt)	Load byte unsigned	$rd←M[$rt+offset]
lh	$rd, offset($rt)	Load halfword	$rd←M[$rt+offset]
lhu	$rd, offset($rt)	Load halfword unsigned	$rd←M[$rt+offset]
lui	$rd, imm	Load upper immediate	$rd←imm≪16
lw	$rd, offset($rt)	Load word	$rd←M[$rt+offset]
sb	$rs, offset($rt)	Store byte	$rs→M[$rt+offset]
sh	$rs, offset($rt)	Store halfword	$rs→M[$rt+offset]
sw	$rs, offset($rt)	Store word	$rs→M[$rt+offset]

if condition is false: pc←pc+4. See pseudo-instructions for more branching instructions

appropriate bits only, rest unchanged. M: memory. offset: halfword

MIPS: Jump, branch and memory instructions:

MIPS instruction		Machine code					
Mnemonic	Operands	format	opcode/ func	$A	$B	$C	imm
j	addr	J	02/-	-	-	-	mask
jal	addr	J	03/-	-	-	-	mask
jr	$rs	R	00/08	$rs	0	0	0
jalr	$rs, $rd	R	00/09	$rs	0	$rd	0
beq	$rt, $rs, addr	I	04/-	$rt	$rs	-	relative
bne	$rt, $rs, addr	I	05/-	$rt	$rs	-	relative
bltz	$rt, addr	I	01/-	$rt	0	-	relative
bgtz	$rt, addr	I	07/-	$rt	0	-	relative
blez	$rt, addr	I	06/-	$rt	0	-	relative
bgez	$rt, addr	I	01/-	$rt	1	-	relative
mfhi	$rd	R	00/10	0	0	$rd	0
mthi	$rs	R	00/11	$rs	0	0	0
mflo	$rd	R	00/12	0	0	$rd	0
mtlo	$rs	R	00/13	$rs	0	0	0
lb	$rd, offset($rt)	I	20/-	$rt	$rd	-	offset
lbu	$rd, offset($rt)	I	24/-	$rt	$rd	-	offset
lh	$rd, offset($rt)	I	21/-	$rt	$rd	-	offset
lhu	$rd, offset($rt)	I	25/-	$rt	$rd	-	offset
lui	$rd, imm	I	0f/-	0	$rd	-	imm
lw	$rd, offset($rt)	I	23/-	$rt	$rd	-	offset
sb	$rs, offset($rt)	I	28/-	$rt	$rs	-	offset
sh	$rs, offset($rt)	I	29/-	$rt	$rs	-	offset
sw	$rs, offset($rt)	I	2b/-	$rt	$rs	-	offset

imm: 26-bit unsigned value. addr=page+4*imm, page=pc&$F0000000 mask=(addr&0FFFFFFC)/4

imm: 16-bit two's-complement signed value. relative=(addr-pc-4)/4

appropriate bits only, rest unchanged. offset: halfword

imm,offset: 16-bit two's-complement signed value.

MIPS: Floating-point instructions:

MIPS instruction		Meaning	
Mnemonic	Operands	Description	Operation
lwc1	$fd, offs($rt)	Load single	$fd←M[$rt+offs]
ldc1	$fd, offs($rt)	Load double	$fd←M[$rt+offs]
swc1	$fs, offs($rt)	Store single	$fs→M[$rt+offs]
sdc1	$fs, offs($rt)	Store double	$fs→M[$rt+offs]

M: memory

Mnemonic	Operands	Description	Operation
c.eq.SIZE	$fs, $ft	FPs equal	cflag←($fs==$ft)?1:0
c.lt.SIZE	$fs, $ft	FPs less than	cflag←($fs<$ft)?1:0
c.le.SIZE	$fs, $ft	FPs less or equal	cflag←($fs<=$ft)?1:0

cvt.TO.FROM	$fd, $fs	Convert	$fd = convert($fs)

add.SIZE	$fd, $ft, $fs	FP add	$fd←$ft+$fs
sub.SIZE	$fd, $ft, $fs	FP subtract	$fd←$ft-$fs
mul.SIZE	$fd, $ft, $fs	FP multiply	$fd←$ft*$fs
div.SIZE	$fd, $ft, $fs	FP divide	$fd←$ft/$fs

mov.SIZE	$fd, $fs	Copy	$fd←$fs
mfc1	$rd, $fs	Copy from co-proc.	$rd←$fs
mtc1	$rs, $fd	Copy to co-proc.	$rs→$fd

SIZE = {s,d}. {TO,FROM} = {s,d,w}

Mnemonic	Operands	Description	Operation
bc1t	addr	Branch if cflag true	cflag? pc←addr
bc1f	addr	Branch if cflag false	cflag? pc←addr

Otherwise: pc←pc+4

Program flow:

Mnemonic		Description	
syscall		System call	*
break	num	Exit with exception	

*: Specify type of syscall by $v0.

MIPS: Floating-point instructions:

MIPS instruction		Machine code				
Mnemonic	Operands	form	opcode	$A	$B	imm
lwc1	$fd, offs($rt)	I	31	$rt	$fd	offs
ldc1	$fd, offs($rt)	I	35	$rt	$fd	offs
swc1	$fs, offs($rt)	I	39	$rt	$fs	offs
sdc1	$fs, offs($rt)	I	3d	$rt	$fs	offs

offs: 16-bit two's-complement signed value.

Mnemonic	Operands	form	opc/typ	$A	$B	$C	func
c.eq.SIZE	$fs, $ft	fR	11/tp	$ft	$fs	0	32
c.lt.SIZE	$fs, $ft	fR	11/tp	$ft	$fs	0	3c
c.le.SIZE	$fs, $ft	fR	11/tp	$ft	$fs	0	3e

cvt.TO.FROM	$fd, $fs	fR	11/from	0	$fs	$fd	to

add.SIZE	$fd, $ft, $fs	fR	11/tp	$fs	$ft	$fd	00
sub.SIZE	$fd, $ft, $fs	fR	11/tp	$fs	$ft	$fd	01
mul.SIZE	$fd, $ft, $fs	fR	11/tp	$fs	$ft	$fd	02
div.SIZE	$fd, $ft, $fs	fR	11/tp	$fs	$ft	$fd	03

mov.SIZE	$fd, $fs	fR	11/tp	$fd	$fs	0	06
mfc1	$rd, $fs	fR	11/00	$rd	$fs	0	00
mtc1	$rs, $fd	fR	11/04	$rs	$fd	0	00

SIZE, TO, FROM: {tp, from} = 10 (s), 11 (d), 14 (w). to = 20 (s), 21 (d), 24 (w)

Mnemonic	Operands	form	opc/typ	$A	imm
bc1t	addr	fI	21/08	1	relative
bc1f	addr	fI	21/08	0	relative

imm: 16-bit two's-complement signed value. relative=(addr-pc-4)/4

Program flow:

Mnemonic		form	opc/func	$A	$B	$C	imm
syscall		R	00/0c	0	0	0	0
break	num	R	00/0d	0	0	0	num

MIPS: (Some) pseudo-instructions:

rol	$rd, $rt, num	Rotate left	$rd←rol($rt,num)
	srl $at, $rt, 32-num sll $rd, $rt, num or $rd, $rd, $at		
ror	$rd, $rt, num	Rotate right	$rd←ror($rt,num)
	sll $at, $rt, 32-num srl $rd, $rt, num or $rd, $rd, $at		

blt	$rt, $rs, addr	Branch if less than	$rt<$rs?pc←addr
	slt $at, $rt, $rs bne $at, $zero, addr		
bgt	$rt, $rs, addr	Branch if greater than	$rt>$rs?pc←addr
	slt $at, $rs, $rt bne $at, $zero, addr		
ble	$rt, $rs, addr	Branch if less/equal	$rt<=$rs?pc←addr
	slt $at, $rs, $rt beq $at, $zero, addr		
bge	$rt, $rs, addr	Branch if greater/equal	$rt<=$rs?pc←addr
	slt $at, $rt, $rs beq $at, $zero, addr		
beqz	$rt, addr	Branch if equal zero	$rt==0?pc←addr
	beq $rt, $zero, addr		
bnez	$rt, addr	Branch if not equal 0	$rt!=0?pc←addr
	bne $rt, $zero, addr		

move	$rd, $rs	Copy	$rd←$rs
	ori $rd, $zero, $rs		

jalr	$rs	Jump and link register	pc←$rs,$ra←pc+4
	jalr $rs, $ra		

MIPS: (Some) pseudo-instructions (cont.):

lw	$rd, addr	Load word from address	$rd←M[addr]
	lui $at, addr≫16 lw $rd, [addr AND 0x0000FFFF]($at),		
li	$rd, word	Load immediate	$rd←word
	lui $at, word≫16 ori $rd, $at, word AND 0x0000FFFF		
li	$rd, halfword	Load immediate	$rd←halfword
	ori $rd, $zero, halfword		
la	$rd, addr	Load address	$rd←addr
	lui $at, addr≫16 ori $rd, $at, addr AND 0x0000FFFF Note: equal to instruction li $rd, addr		

to be implemented by macros:

inc	$rt	Increment	$rt←$rt+1
	addi $rt, $rt, 1		
dec	$rt	Decrement	$rt←$rt-1
	subi $rt, $rt, 1		

push	$rs	Push onto stack	M[--$sp]←$rs
	addiu $sp, $sp, -4 sw $rs, 0($sp)		
pop	$rd	Pop from stack	$rd←M[$sp++]
	lw $rd, 0($sp) addiu $sp, $sp, 4		

return		Return from subroutine	pc←$ra
	jr $ra		
done		Terminate	
	li $v0, 10 syscall		

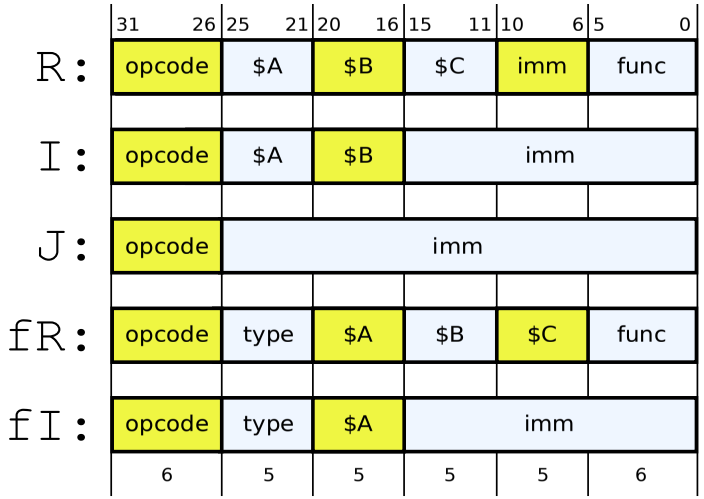
(MARS) MIPS system calls:

function	$v0	argument(s)	return value(s)
print integer	1	$a0 = integer	
print float	2	$f12 = float	
print double	3	$f12, $f13 = double	
print string	4	$a0 = address of null-terminated string	
read integer	5		$v0 integer read
read float	6		$f0 float read
read double	7		$f0,$f1 double read
read string	8	$a0 = address of buffer $a1 = max. length	
exit (terminate execution)	10		
print character	11	$a0 = character	
read character	12		$v0 character read
open file	13	$a0 = address of filename $a1 = flags (0=read, 1=overwrite,9=append) $a2 = mode (0)	$v0 file descriptor
read from file	14	$a0 = file descriptor $a1 = addr. input buffer $a2 = max length	$v0 number of chars read (0:end-of-file, <0:error)
write to file	15	$a0 = file descriptor $a1 = addr. output buffer $a2 = number of chars	$v0 number of chars written (<0: error)
close file	16	$a0 = file descriptor	
exit (terminate with value)	17	$a0 = termination result	
print integer in hexadecimal	34	$a0 = integer	
print integer in binary	35	$a0 = integer	
print integer as unsigned	36	$a0 = integer	
set random seed	40	$a0 = integer	
random int	41	$a0 = integer	$a0: next random int
random int in range	42	$a0 = integer $a1 = limit	$a0: next random int in range 0...$a1-1
random float	43	$a0 = integer	$f0: 0.0...0.999...
random double	44	$a0 = integer	$f0, $f1: 0.0...0.999...

MIPS registers:

0	$zero	8	$t0	16	$s0	24	$t8
1	$at	9	$t1	17	$s1	25	$t9
2	$v0	10	$t2	18	$s2	26	$k0
3	$v1	11	$t3	19	$s3	27	$k1
4	$a0	12	$t4	20	$s4	28	$gp
5	$a1	13	$t5	21	$s5	29	$sp
6	$a2	14	$t6	22	$s6	30	$fp
7	$a3	15	$t7	23	$s7	31	$ra

MIPS instruction format types:



Values in the tables are hexadecimal, unless otherwise specified

MARS (MIPS) assembler directives:

Directive Example	Meaning
`.data`	Start of data segment
`.text`	Start of code segment
`.globl` `.globl main`	Entry point for external reference (linker)
`.space n` `myarray: .space 12`	Reserve n bytes of space on the heap
`.ascii "string"` `mystring: .ascii "Ajax"`	Store string in heap memory
`.asciiz "string"` `mytext: .asciiz "Benfica"`	Store string+0x00 in heap memory
`.byte b1, b2,...bn`	Store byte(s) in heap memory
`.half h1, h2,...hn`	Store half-word(s) in memory
`.word w1, w2,...wn` `myvector: .word 1, 2, 4`	Store word(s) in heap memory
`.float f1, f2,...fn` `myvector: .float 1.0, 2.1, 3.6`	Store float(s) in heap memory
`.double d1, d2,...dn` `pi: .double 3.1415926E03`	Store double-precision float(s) in heap memory
`.eqv text text` `.eqv myvalue 64`	Define a substitution

`.macro ... .end_macro`
`.macro endprog`
`li $v0, 10`
`syscall`
`.end_macro`
	Define a macro*
	(not a function)

*: (does not store it in memory)