Important Note:

Do not write anything on the provided document. The document is to be returned.

MIPS Instruction Sheet

Arithmetic and Logical Instructions

Instruction	Operation
add $d, $s, $t	$d = $s + $t
addu $d, $s, $t	$d = $s + $t
addi $t, $s, i	$t = $s + SE(i)
addiu $t, $s, i	$t = $s + SE(i)
and $d, $s, $t	$d = $s & $t
andi $t, $s, i	$t = $s & ZE(i)
div $s, $t	lo = $s / $t; hi = $s % $t
divu $s, $t	lo = $s / $t; hi = $s % $t
mult $s, $t	hi:lo = $s * $t
multu $s, $t	hi:lo = $s * $t
nor $d, $s, $t	$d = ~($s | $t)
or $d, $s, $t	$d = $s | $t
ori $t, $s, i	$t = $s | ZE(i)
sll $d, $t, a	$d = $t << a
sllv $d, $t, $s	$d = $t << $s
sra $d, $t, a	$d = $t >> a
srav $d, $t, $s	$d = $t >> $s
srl $d, $t, a	$d = $t >>> a
srlv $d, $t, $s	$d = $t >>> $s
sub $d, $s, $t	$d = $s - $t
subu $d, $s, $t	$d = $s - $t
xor $d, $s, $t	$d = $s ^ $t
xori $d, $s, i	$d = $s ^ ZE(i)

Constant-Manipulating Instructions

Instruction	Operation
lhi $t, i	HH($t) = i
llo $t, i	LH($t) = i

Comparison Instructions

Instruction	Operation
slt $d, $s, $t	$d = ($s < $t)
sltu $d, $s, $t	$d = ($s < $t)
slti $t, $s, i	$t = ($s < SE(i))
sltiu $t, $s, i	$t = ($s < SE(i))

Branch Instructions

Instruction	Operation
beq $s, $t, label	if ($s == $t) pc += i << 2
bgtz $s, label	if ($s > 0) pc += i << 2
blez $s, label	if ($s <= 0) pc += i << 2
bne $s, $t, label	if ($s != $t) pc += i << 2

Jump Instructions

Instruction	Operation
j label	pc += i << 2
jal label	$31 = pc; pc += i << 2
jalr $s	$31 = pc; pc = $s
jr $s	pc = $s

Load Instructions

Instruction	Operation
lb $t, i($s)	$t = SE (MEM [$s + i]:1)
lbu $t, i($s)	$t = ZE (MEM [$s + i]:1)
lh $t, i($s)	$t = SE (MEM [$s + i]:2)
lhu $t, i($s)	$t = ZE (MEM [$s + i]:2)
lw $t, i($s)	$t = MEM [$s + i]:4

Store Instructions

Instruction	Operation
sb $t, i($s)	MEM [$s + i]:1 = LB ($t)
sh $t, i($s)	MEM [$s + i]:2 = LH ($t)
sw $t, i($s)	MEM [$s + i]:4 = $t

Data Movement Instructions

Instruction	Operation
mfhi $d	$d = hi
mflo $d	$d = lo
mthi $s	hi = $s
mtlo $s	lo = $s

Exception and Interrupt Instructions

Instruction	Operation
trap 1	Print integer value in $4
trap 5	Read integer value into $2
trap 10	Terminate program execution
trap 101	Print ASCII character in $4
trap 102	Read ASCII character into $2

Instruction Encodings

Register	000000ss sssttttt dddddaaa aaffffff
Immediate	ooooooss sssttttt iiiiiiii iiiiiiii
Jump	ooooooii iiiiiiii iiiiiiii iiiiiiii

Instruction Syntax

Syntax	Template	Encoding	Comments
ArithLog	f $d, $s, $t	Register	
DivMult	f $s, $t	Register	
Shift	f $d, $t, a	Register	
ShiftV	f $d, $t, $s	Register	
JumpR	f $s	Register	
MoveFrom	f $d	Register	
MoveTo	f $s	Register	
ArithLogI	o $t, $s, i	Immediate	
LoadI	o $t, immed32	Immediate	i is high or low 16 bits of immed32
Branch	o $s, $t, label	Immediate	i is calculated as (label-(current+4))>>2
BranchZ	o $s, label	Immediate	i is calculated as (label-(current+4))>>2
LoadStore	o $t, i($s)	Immediate	
Jump	o label	Jump	i is calculated as label<<2
Trap	o i	Jump	

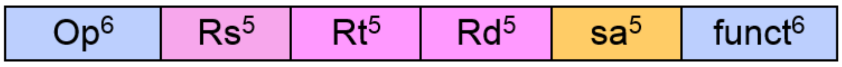
MIPS Machine Code

Instruction	Opcode/Function	Syntax
add	100000	ArithLog
addu	100001	ArithLog
addi	001000	ArithLogI
addiu	001001	ArithLogI
and	100100	ArithLog
andi	001100	ArithLogI
div	011010	DivMult
divu	011011	DivMult
mult	011000	DivMult
multu	011001	DivMult
nor	100111	ArithLog
or	100101	ArithLog
ori	001101	ArithLogI
sll	000000	Shift
sllv	000100	ShiftV
sra	000011	Shift
srav	000111	ShiftV
srl	000010	Shift
srlv	000110	ShiftV
sub	100010	ArithLog
subu	100011	ArithLog
xor	100110	ArithLog
xori	001110	ArithLogI
lhi	011001	LoadI
llo	011000	LoadI

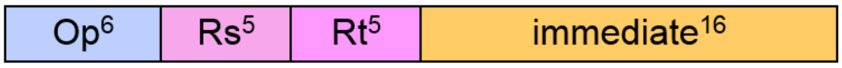
Instruction	Opcode/Function	Syntax
slt	101010	ArithLog
sltu	101001	ArithLog
slti	001010	ArithLogI
sltiu	001001	ArithLogI
beq	000100	Branch
bgtz	000111	BranchZ
blez	000110	BranchZ
bne	000101	Branch
j	000010	Jump
jal	000011	Jump
jalr	001001	JumpR
jr	001000	JumpR
lb	100000	LoadStore
lbu	100100	LoadStore
lh	100001	LoadStore
lhu	100101	LoadStore
lw	100011	LoadStore
sb	101000	LoadStore
sh	101001	LoadStore
sw	101011	LoadStore
mfhi	010000	MoveFrom
mflo	010010	MoveFrom
mthi	010001	MoveTo
mtlo	010011	MoveTo
trap	011010	Trap

Instruction Format

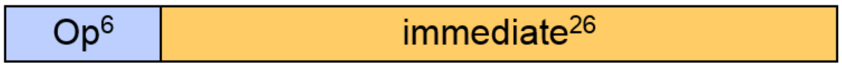
- Register (R-Type)
  - Register-to-register instructions
  - Op: operation code specifies the format of the instruction



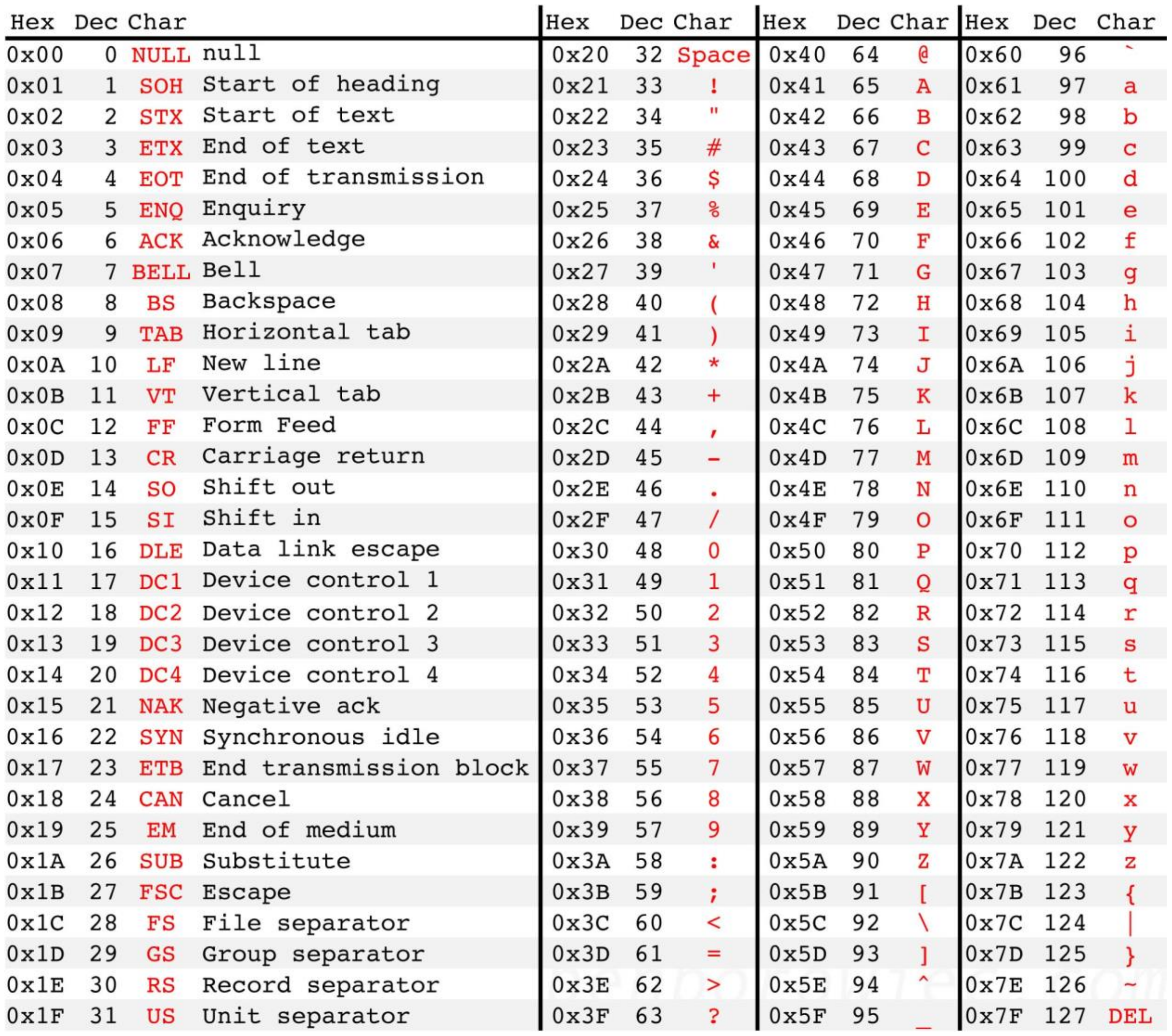
- Immediate (I-Type)
  - 16-bit immediate constant is part in the instruction



- Jump (J-Type)
  - Used by jump instructions



ASCII Table



Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

SYSCALL Services

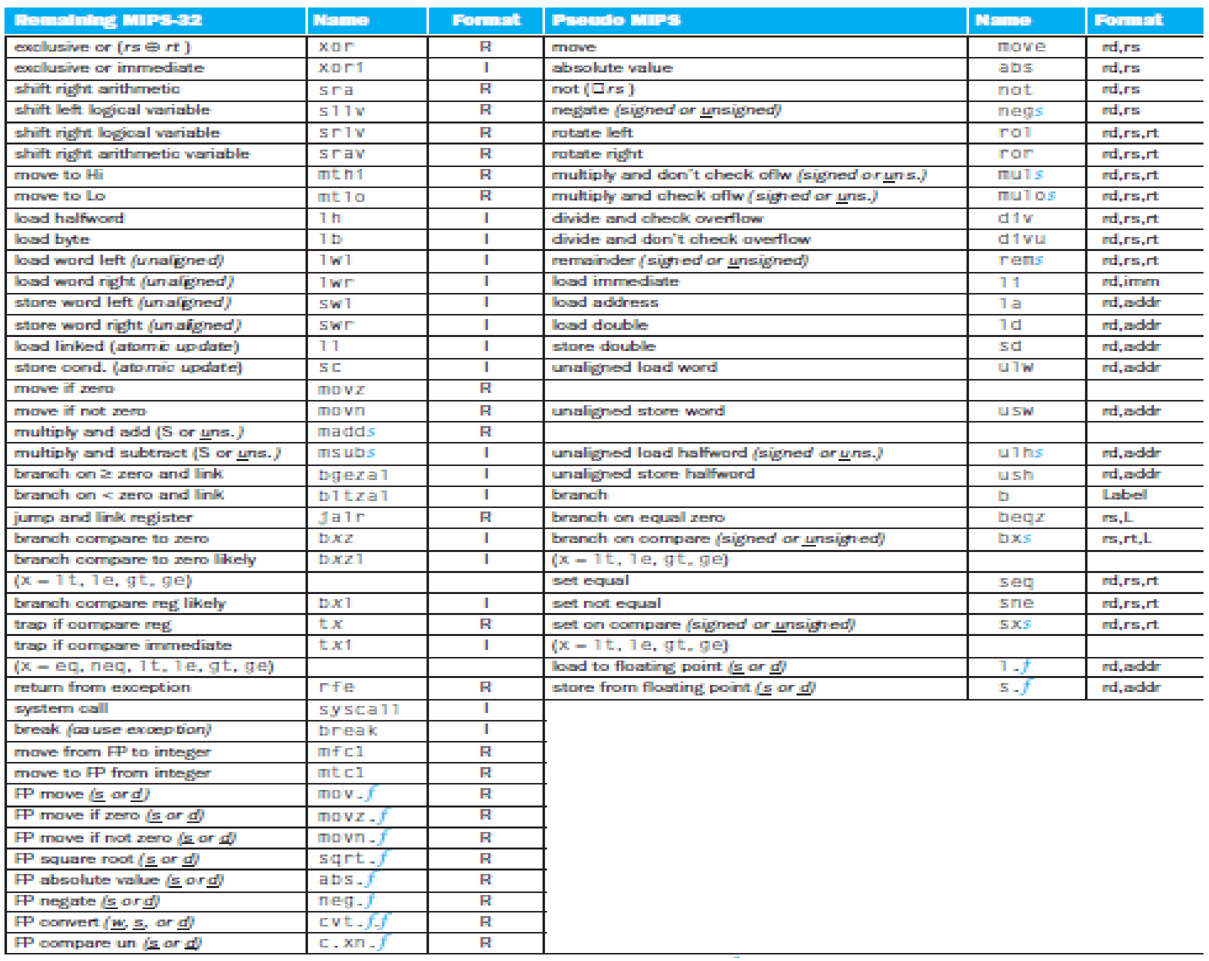
Service	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	$v0 = integer read
Read Float	6	$f0 = float read
Read Double	7	$f0 = double read
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Exit Program	10	
Print Char	11	$a0 = character to print Supported by MARS
Read Char	12	$a0 = character read

Name	Reg No.	Usage
`$zero`	0	the constant value 0
`$v0-$v1`	2-3	values for results and expression evaluation
`$a0-$a3`	4-7	arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved
`$t8-$t9`	24-25	more temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address

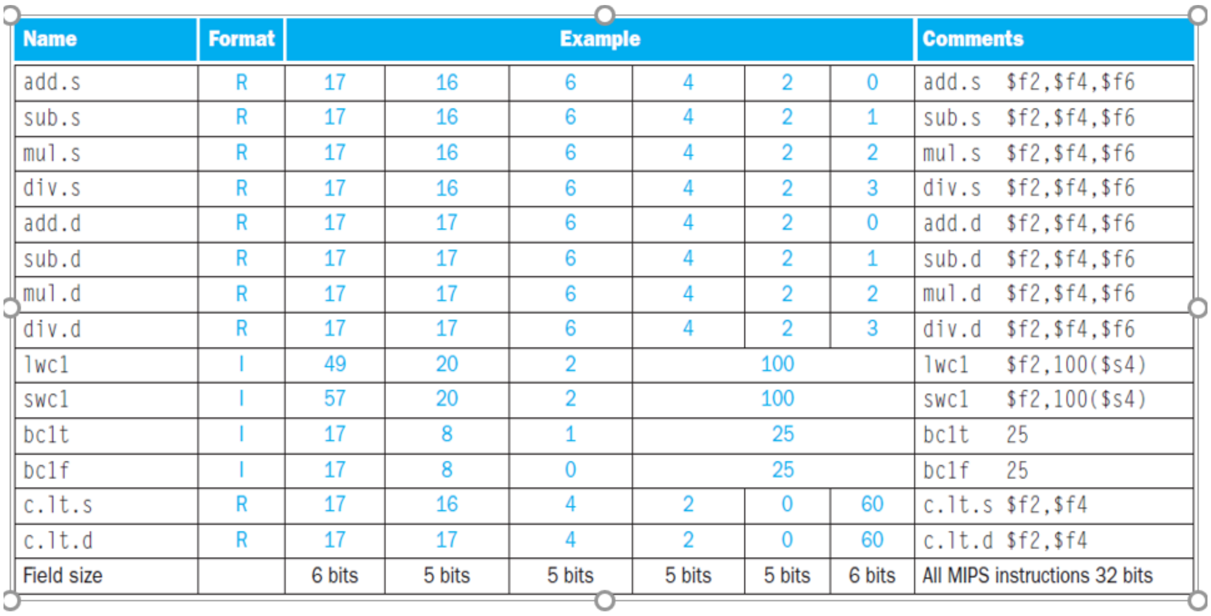
Register name	Number	Usage
$zero	0	constant 0
$at	1	reserved for assembler
$v0	2	expression evaluation and results of a function
$v1	3	expression evaluation and results of a function
$a0	4	argument 1
$a1	5	argument 2
$a2	6	argument 3
$a3	7	argument 4
$t0	8	temporary (not preserved across call)
$t1	9	temporary (not preserved across call)
$t2	10	temporary (not preserved across call)
$t3	11	temporary (not preserved across call)
$t4	12	temporary (not preserved across call)
$t5	13	temporary (not preserved across call)
$t6	14	temporary (not preserved across call)
$t7	15	temporary (not preserved across call)
$s0	16	saved temporary (preserved across call)
$s1	17	saved temporary (preserved across call)
$s2	18	saved temporary (preserved across call)
$s3	19	saved temporary (preserved across call)
$s4	20	saved temporary (preserved across call)
$s5	21	saved temporary (preserved across call)
$s6	22	saved temporary (preserved across call)
$s7	23	saved temporary (preserved across call)
$t8	24	temporary (not preserved across call)
$t9	25	temporary (not preserved across call)
$k0	26	reserved for OS kernel
$k1	27	reserved for OS kernel
$gp	28	pointer to global area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address (used by function call)

Instruction	Meaning	Format					
add.s fd, fs, ft	(fd) = (fs) + (ft)	0x11	0	ft^5	fs^5	fd^5	0
add.d fd, fs, ft	(fd) = (fs) + (ft)	0x11	1	ft^5	fs^5	fd^5	0
sub.s fd, fs, ft	(fd) = (fs) – (ft)	0x11	0	ft^5	fs^5	fd^5	1
sub.d fd, fs, ft	(fd) = (fs) – (ft)	0x11	1	ft^5	fs^5	fd^5	1
mul.s fd, fs, ft	(fd) = (fs) × (ft)	0x11	0	ft^5	fs^5	fd^5	2
mul.d fd, fs, ft	(fd) = (fs) × (ft)	0x11	1	ft^5	fs^5	fd^5	2
div.s fd, fs, ft	(fd) = (fs) / (ft)	0x11	0	ft^5	fs^5	fd^5	3
div.d fd, fs, ft	(fd) = (fs) / (ft)	0x11	1	ft^5	fs^5	fd^5	3
sqrt.s fd, fs	(fd) = sqrt (fs)	0x11	0	0	fs^5	fd^5	4
sqrt.d fd, fs	(fd) = sqrt (fs)	0x11	1	0	fs^5	fd^5	4
abs.s fd, fs	(fd) = abs (fs)	0x11	0	0	fs^5	fd^5	5
abs.d fd, fs	(fd) = abs (fs)	0x11	1	0	fs^5	fd^5	5
neg.s fd, fs	(fd) = – (fs)	0x11	0	0	fs^5	fd^5	7
neg.d fd, fs	(fd) = – (fs)	0x11	1	0	fs^5	fd^5	7

R	**0**	Rs	Rt	Rd	ShA	func
I	**1, 4-31**	Rs	Rt	Immediate (16 bit)		
J	**2**	Immediate (26-bit)				
FP	**17**	As per Rule				
LWC1	**49**	Rs	Ft	Immediate (16 bit)		
LDC1	**53**	Rs	Ft	Immediate (16 bit)		
SWC1	**57**	Rs	Ft	Immediate (16 bit)		
SDC1	**61**	Rs	Ft	Immediate (16 bit)		



Remaining MIPS-32	Name	Format	Pseudo MIPS	Name	Format
exclusive or (*rs* ⊕ *rt*)	xor	R	move	move	rd,rs
exclusive or immediate	xori	I	absolute value	abs	rd,rs
shift right arithmetic	sra	R	not (□rs)	not	rd,rs
shift left logical variable	sllv	R	negate *(signed or unsigned)*	negs	rd,rs
shift right logical variable	srlv	R	rotate left	rol	rd,rs,rt
shift right arithmetic variable	srav	R	rotate right	ror	rd,rs,rt
move to Hi	mthi	R	multiply and don't check oflw *(signed or uns.)*	muls	rd,rs,rt
move to Lo	mtlo	R	multiply and check oflw *(signed or uns.)*	mulos	rd,rs,rt
load halfword	lh	I	divide and check overflow	div	rd,rs,rt
load byte	lb	I	divide and don't check overflow	divu	rd,rs,rt
load word left *(unaligned)*	lwl	I	remainder *(signed or unsigned)*	rems	rd,rs,rt
load word right *(unaligned)*	lwr	I	load immediate	li	rd,imm
store word left *(unaligned)*	swl	I	load address	la	rd,addr
store word right *(unaligned)*	swr	I	load double	ld	rd,addr
load linked (atomic update)	ll	I	store double	sd	rd,addr
store cond. (atomic update)	sc	I	unaligned load word	ulw	rd,addr
move if zero	movz	R			
move if not zero	movn	R	unaligned store word	usw	rd,addr
multiply and add (S or uns.)	madds	R			
multiply and subtract (S or uns.)	msubs	I	unaligned load halfword *(signed or uns.)*	ulhs	rd,addr
branch on ≥ zero and link	bgezal	I	unaligned store halfword	ush	rd,addr
branch on < zero and link	bltzal	I	branch	b	Label
jump and link register	jalr	R	branch on equal zero	beqz	rs,L
branch compare to zero	bxz	I	branch on compare *(signed or unsigned)*	bxs	rs,rt,L
branch compare to zero likely	bxzl	I	(x = lt, le, gt, ge)		
(x = lt, le, gt, ge)			set equal	seq	rd,rs,rt
branch compare reg likely	bxl	I	set not equal	sne	rd,rs,rt
trap if compare reg	tx	R	set on compare *(signed or unsigned)*	sxs	rd,rs,rt
trap if compare immediate	txi	I	(x = lt, le, gt, ge)		
(x = eq, neq, lt, le, gt, ge)			load to floating point *(s or d)*	l.*f*	rd,addr
return from exception	rfe	R	store from floating point *(s or d)*	s.*f*	rd,addr
system call	syscall	I			
break *(cause exception)*	break	I			
move from FP to integer	mfc1	R			
move to FP from integer	mtc1	R			
FP move *(s or d)*	mov.*f*	R			
FP move if zero *(s or d)*	movz.*f*	R			
FP move if not zero *(s or d)*	movn.*f*	R			
FP square root *(s or d)*	sqrt.*f*	R			
FP absolute value *(s or d)*	abs.*f*	R			
FP negate *(s or d)*	neg.*f*	R			
FP convert *(w, s, or d)*	cvt.*f*.*f*	R			
FP compare un *(s or d)*	c.xn.*f*	R			



Name	Format	Example						Comments
add.s	R	17	16	6	4	2	0	add.s $f2,$f4,$f6
sub.s	R	17	16	6	4	2	1	sub.s $f2,$f4,$f6
mul.s	R	17	16	6	4	2	2	mul.s $f2,$f4,$f6
div.s	R	17	16	6	4	2	3	div.s $f2,$f4,$f6
add.d	R	17	17	6	4	2	0	add.d $f2,$f4,$f6
sub.d	R	17	17	6	4	2	1	sub.d $f2,$f4,$f6
mul.d	R	17	17	6	4	2	2	mul.d $f2,$f4,$f6
div.d	R	17	17	6	4	2	3	div.d $f2,$f4,$f6
lwc1	I	49	20	2	100			lwc1 $f2,100($s4)
swc1	I	57	20	2	100			swc1 $f2,100($s4)
bc1t	I	17	8	1	25			bc1t 25
bc1f	I	17	8	0	25			bc1f 25
c.lt.s	R	17	16	4	2	0	60	c.lt.s $f2,$f4
c.lt.d	R	17	17	4	2	0	60	c.lt.d $f2,$f4
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits

R	**0**	Rs	Rt	Rd	ShA	func
I	**1, 4-31**	Rs	Rt	Immediate (16 bit)		
J	**2**	Immediate (26-bit)				
FP	**17**	As per Rule				
LWC1	**49**	Rs	Ft	Immediate (16 bit)		
LDC1	**53**	Rs	Ft	Immediate (16 bit)		
SWC1	**57**	Rs	Ft	Immediate (16 bit)		
SDC1	**61**	Rs	Ft	Immediate (16 bit)		

Instruction	Meaning	Format					
mfc1 $t0, $f2	($t0) = ($f2)	0x11	0	$t0	$f2	0	0
mtc1 $t0, $f2	($f2) = ($t0)	0x11	4	$t0	$f2	0	0
mov.s $f4, $f2	($f4) = ($f2)	0x11	0	0	$f2	$f4	6
mov.d $f4, $f2	($f4) = ($f2)	0x11	1	0	$f2	$f4	6

Instruction	Meaning	Format					
cvt.s.w fd, fs	to single from integer	0x11	0	0	fs^5	fd^5	0x20
cvt.s.d fd, fs	to single from double	0x11	1	0	fs^5	fd^5	0x20
cvt.d.w fd, fs	to double from integer	0x11	0	0	fs^5	fd^5	0x21
cvt.d.s fd, fs	to double from single	0x11	1	0	fs^5	fd^5	0x21
cvt.w.s fd, fs	to integer from single	0x11	0	0	fs^5	fd^5	0x24
cvt.w.d fd, fs	to integer from double	0x11	1	0	fs^5	fd^5	0x24