MIPS32® Instruction Set Quick Reference

Rd	—	Destination register
Rs, Rt	—	Source operand registers
Ra	—	Return address register (R31)
PC	—	Program counter
Acc	—	64-bit accumulator
Lo, Hi	—	Accumulator low (Acc$_{31:0}$) and high (Acc$_{63:32}$) parts
±	—	Signed operand or sign extension
Ø	—	Unsigned operand or zero extension
::	—	Concatenation of bit fields
R2	—	MIPS32 Release 2 instruction
dotted	—	Assembler pseudo-instruction

Please refer to *"MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set"* for complete instruction set information.

Arithmetic Operations			
ADD	Rd, Rs, Rt	Rd = Rs + Rt	(overflow trap)
ADDI	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$	(overflow trap)
ADDIU	Rd, Rs, const16	Rd = Rs + const16$^{\pm}$	
ADDU	Rd, Rs, Rt	Rd = Rs + Rt	
CLO	Rd, Rs	Rd = CountLeadingOnes(Rs)	
CLZ	Rd, Rs	Rd = CountLeadingZeros(Rs)	
LA	Rd, label	Rd = Address(label)	
LI	Rd, imm32	Rd = imm32	
LUI	Rd, const16	Rd = const16 << 16	
MOVE	Rd, Rs	Rd = Rs	
NEGU	Rd, Rs	Rd = –Rs	
SEB^{R2}	Rd, Rs	Rd = Rs$_{7:0}^{\pm}$	
SEH^{R2}	Rd, Rs	Rd = Rs$_{15:0}^{\pm}$	
SUB	Rd, Rs, Rt	Rd = Rs – Rt	(overflow trap)
SUBU	Rd, Rs, Rt	Rd = Rs – Rt	

Shift and Rotate Operations		
ROTR^{R2}	Rd, Rs, bits5	Rd = Rs$_{\text{bits5}-1:0}$ :: Rs$_{31:\text{bits5}}$
ROTRV^{R2}	Rd, Rs, Rt	Rd = Rs$_{\text{Rt}4:0-1:0}$ :: Rs$_{31:\text{Rt}4:0}$
SLL	Rd, Rs, shift5	Rd = Rs << shift5
SLLV	Rd, Rs, Rt	Rd = Rs << Rt$_{4:0}$
SRA	Rd, Rs, shift5	Rd = Rs$^{\pm}$ >> shift5
SRAV	Rd, Rs, Rt	Rd = Rs$^{\pm}$ >> Rt$_{4:0}$
SRL	Rd, Rs, shift5	Rd = Rs$^{\varnothing}$ >> shift5
SRLV	Rd, Rs, Rt	Rd = Rs$^{\varnothing}$ >> Rt$_{4:0}$

Logical and Bit-Field Operations		
AND	Rd, Rs, Rt	Rd = Rs & Rt
ANDI	Rd, Rs, const16	Rd = Rs & const16$^{\varnothing}$
EXT^{R2}	Rd, Rs, P, S	Rs = Rs$_{P+S-1:P}^{\varnothing}$
INS^{R2}	Rd, Rs, P, S	Rd$_{P+S-1:P}$ = Rs$_{S-1:0}$
NOP		No-op
NOR	Rd, Rs, Rt	Rd = ~(Rs | Rt)
NOT	Rd, Rs	Rd = ~Rs
OR	Rd, Rs, Rt	Rd = Rs | Rt
ORI	Rd, Rs, const16	Rd = Rs | const16$^{\varnothing}$
WSBH^{R2}	Rd, Rs	Rd = Rs$_{23:16}$ :: Rs$_{31:24}$ :: Rs$_{7:0}$ :: Rs$_{15:8}$
XOR	Rd, Rs, Rt	Rd = Rs ⊕ Rt
XORI	Rd, Rs, const16	Rd = Rs ⊕ const16$^{\varnothing}$

Condition Testing and Conditional Move Operations		
MOVN	Rd, Rs, Rt	if Rt ≠ 0, Rd = Rs
MOVZ	Rd, Rs, Rt	if Rt = 0, Rd = Rs
SLT	Rd, Rs, Rt	Rd = (Rs$^{\pm}$ < Rt$^{\pm}$) ? 1 : 0
SLTI	Rd, Rs, const16	Rd = (Rs$^{\pm}$ < const16$^{\pm}$) ? 1 : 0
SLTIU	Rd, Rs, const16	Rd = (Rs$^{\varnothing}$ < const16$^{\varnothing}$) ? 1 : 0
SLTU	Rd, Rs, Rt	Rd = (Rs$^{\varnothing}$ < Rt$^{\varnothing}$) ? 1 : 0

Multiply and Divide Operations		
DIV	Rs, Rt	Lo = Rs$^{\pm}$ / Rt$^{\pm}$; Hi = Rs$^{\pm}$ mod Rt$^{\pm}$
DIVU	Rs, Rt	Lo = Rs$^{\varnothing}$ / Rt$^{\varnothing}$; Hi = Rs$^{\varnothing}$ mod Rt$^{\varnothing}$
MADD	Rs, Rt	Acc += Rs$^{\pm}$ × Rt$^{\pm}$
MADDU	Rs, Rt	Acc += Rs$^{\varnothing}$ × Rt$^{\varnothing}$
MSUB	Rs, Rt	Acc −= Rs$^{\pm}$ × Rt$^{\pm}$
MSUBU	Rs, Rt	Acc −= Rs$^{\varnothing}$ × Rt$^{\varnothing}$
MUL	Rd, Rs, Rt	Rd = Rs$^{\pm}$ × Rt$^{\pm}$
MULT	Rs, Rt	Acc = Rs$^{\pm}$ × Rt$^{\pm}$
MULTU	Rs, Rt	Acc = Rs$^{\varnothing}$ × Rt$^{\varnothing}$

Accumulator Access Operations		
MFHI	Rd	Rd = Hi
MFLO	Rd	Rd = Lo
MTHI	Rs	Hi = Rs
MTLO	Rs	Lo = Rs

Jumps And Branches (Note: One Delay Slot)		
B	off18	PC += off18$^{\pm}$
BAL	off18	Ra = PC + 8, PC += off18$^{\pm}$
BEQ	Rs, Rt, off18	if Rs = Rt, PC += off18$^{\pm}$
BEQZ	Rs, off18	if Rs = 0, PC += off18$^{\pm}$
BGEZ	Rs, off18	if Rs ≥ 0, PC += off18$^{\pm}$
BGEZAL	Rs, off18	Ra = PC + 8; if Rs ≥ 0, PC += off18$^{\pm}$
BGTZ	Rs, off18	if Rs > 0, PC += off18$^{\pm}$
BLEZ	Rs, off18	if Rs ≤ 0, PC += off18$^{\pm}$
BLTZ	Rs, off18	if Rs < 0, PC += off18$^{\pm}$
BLTZAL	Rs, off18	Ra = PC + 8; if Rs < 0, PC += off18$^{\pm}$
BNE	Rs, Rt, off18	if Rs ≠ Rt, PC += off18$^{\pm}$
BNEZ	Rs, off18	if Rs ≠ 0, PC += off18$^{\pm}$
J	addr28	PC = PC$_{31:28}$ :: addr28$^{\varnothing}$
JAL	addr28	Ra = PC + 8; PC = PC$_{31:28}$ :: addr28$^{\varnothing}$
JALR	Rd, Rs	Rd = PC + 8; PC = Rs
JR	Rs	PC = Rs

Load and Store Operations		
LB	Rd, off16(Rs)	Rd = mem8(Rs + off16$^{\pm}$)$^{\pm}$
LBU	Rd, off16(Rs)	Rd = mem8(Rs + off16$^{\pm}$)$^{\varnothing}$
LH	Rd, off16(Rs)	Rd = mem16(Rs + off16$^{\pm}$)$^{\pm}$
LHU	Rd, off16(Rs)	Rd = mem16(Rs + off16$^{\pm}$)$^{\varnothing}$
LW	Rd, off16(Rs)	Rd = mem32(Rs + off16$^{\pm}$)
LWL	Rd, off16(Rs)	Rd = LoadWordLeft(Rs + off16$^{\pm}$)
LWR	Rd, off16(Rs)	Rd = LoadWordRight(Rs + off16$^{\pm}$)
SB	Rs, off16(Rt)	mem8(Rt + off16$^{\pm}$) = Rs$_{7:0}$
SH	Rs, off16(Rt)	mem16(Rt + off16$^{\pm}$) = Rs$_{15:0}$
SW	Rs, off16(Rt)	mem32(Rt + off16$^{\pm}$) = Rs
SWL	Rs, off16(Rt)	StoreWordLeft(Rt + off16$^{\pm}$, Rs)
SWR	Rs, off16(Rt)	StoreWordRight(Rt + off16$^{\pm}$, Rs)
ULW	Rd, off16(Rs)	Rd = unaligned_mem32(Rs + off16$^{\pm}$)
USW	Rs, off16(Rt)	unaligned_mem32(Rt + off16$^{\pm}$) = Rs

Atomic Read-Modify-Write Operations		
LL	Rd, off16(Rs)	Rd = mem32(Rs + off16$^{\pm}$); link
SC	Rd, off16(Rs)	if Atomic, mem32(Rs + off16$^{\pm}$) = Rd; Rd = Atomic ? 1 : 0

MIPS

OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Deci-mal	Hexa-deci-mal	ASCII Char-acter	Deci-mal	Hexa-deci-mal	ASCII Char-acter
(1)	`sll`	`add.`*f*	00 0000	0	0	NUL	64	40	@
		`sub.`*f*	00 0001	1	1	SOH	65	41	A
`j`	`srl`	`mul.`*f*	00 0010	2	2	STX	66	42	B
`jal`	`sra`	`div.`*f*	00 0011	3	3	ETX	67	43	C
`beq`	`sllv`	`sqrt.`*f*	00 0100	4	4	EOT	68	44	D
`bne`		`abs.`*f*	00 0101	5	5	ENQ	69	45	E
`blez`	`srlv`	`mov.`*f*	00 0110	6	6	ACK	70	46	F
`bgtz`	`srav`	`neg.`*f*	00 0111	7	7	BEL	71	47	G
`addi`	`jr`		00 1000	8	8	BS	72	48	H
`addiu`	`jalr`		00 1001	9	9	HT	73	49	I
`slti`	`movz`		00 1010	10	a	LF	74	4a	J
`sltiu`	`movn`		00 1011	11	b	VT	75	4b	K
`andi`	`syscall`	`round.w.`*f*	00 1100	12	c	FF	76	4c	L
`ori`	`break`	`trunc.w.`*f*	00 1101	13	d	CR	77	4d	M
`xori`		`ceil.w.`*f*	00 1110	14	e	SO	78	4e	N
`lui`	`sync`	`floor.w.`*f*	00 1111	15	f	SI	79	4f	O
	`mfhi`		01 0000	16	10	DLE	80	50	P
(2)	`mthi`		01 0001	17	11	DC1	81	51	Q
	`mflo`	`movz.`*f*	01 0010	18	12	DC2	82	52	R
	`mtlo`	`movn.`*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	`mult`		01 1000	24	18	CAN	88	58	X
	`multu`		01 1001	25	19	EM	89	59	Y
	`div`		01 1010	26	1a	SUB	90	5a	Z
	`divu`		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
`lb`	`add`	`cvt.s.`*f*	10 0000	32	20	Space	96	60	‘
`lh`	`addu`	`cvt.d.`*f*	10 0001	33	21	!	97	61	a
`lwl`	`sub`		10 0010	34	22	"	98	62	b
`lw`	`subu`		10 0011	35	23	#	99	63	c
`lbu`	`and`	`cvt.w.`*f*	10 0100	36	24	$	100	64	d
`lhu`	`or`		10 0101	37	25	%	101	65	e
`lwr`	`xor`		10 0110	38	26	&	102	66	f
	`nor`		10 0111	39	27	’	103	67	g
`sb`			10 1000	40	28	(	104	68	h
`sh`			10 1001	41	29	)	105	69	i
`swl`	`slt`		10 1010	42	2a	*	106	6a	j
`sw`	`sltu`		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
`swr`			10 1110	46	2e	.	110	6e	n
`cache`			10 1111	47	2f	/	111	6f	o
`ll`	`tge`	`c.f.`*f*	11 0000	48	30	0	112	70	p
`lwc1`	`tgeu`	`c.un.`*f*	11 0001	49	31	1	113	71	q
`lwc2`	`tlt`	`c.eq.`*f*	11 0010	50	32	2	114	72	r
`pref`	`tltu`	`c.ueq.`*f*	11 0011	51	33	3	115	73	s
	`teq`	`c.olt.`*f*	11 0100	52	34	4	116	74	t
`ldc1`		`c.ult.`*f*	11 0101	53	35	5	117	75	u
`ldc2`	`tne`	`c.ole.`*f*	11 0110	54	36	6	118	76	v
		`c.ule.`*f*	11 0111	55	37	7	119	77	w
`sc`		`c.sf.`*f*	11 1000	56	38	8	120	78	x
`swc1`		`c.ngle.`*f*	11 1001	57	39	9	121	79	y
`swc2`		`c.seq.`*f*	11 1010	58	3a	:	122	7a	z
		`c.ngl.`*f*	11 1011	59	3b	;	123	7b	{
		`c.lt.`*f*	11 1100	60	3c	<	124	7c	|
`sdc1`		`c.nge.`*f*	11 1101	61	3d	=	125	7d	}
`sdc2`		`c.le.`*f*	11 1110	62	3e	>	126	7e	~
		`c.ngt.`*f*	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) == 0

(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = `s` (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = `d` (double)

IEEE 754 FLOATING POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

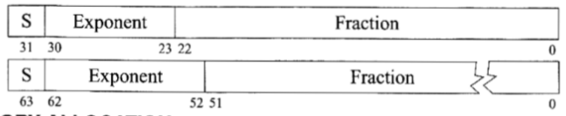
where Single Precision Bias = 127, Double Precision Bias = 1023.

IEEE 754 Symbols

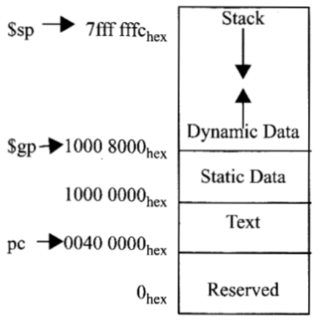
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

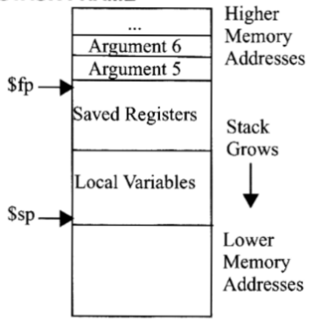
IEEE Single Precision and Double Precision Formats:



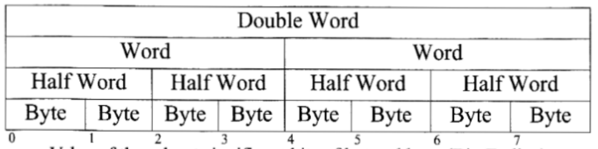
MEMORY ALLOCATION



STACK FRAME

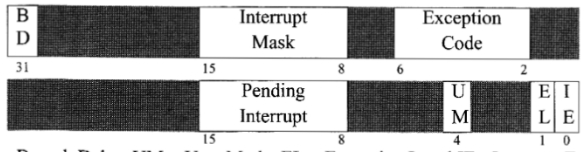


DATA ALIGNMENT



Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level,IE =Interrupt Enable

EXCEPTION CODES

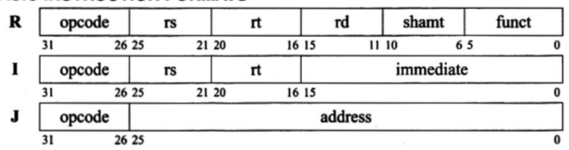
Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

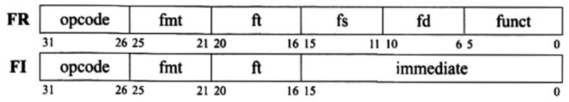
SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX	SIZE	PRE-FIX
$10^3, 2^{10}$	Kilo-	$10^{15}, 2^{50}$	Peta-	10^{-3}	milli-	10^{-15}	femto-
$10^6, 2^{20}$	Mega-	$10^{18}, 2^{60}$	Exa-	10^{-6}	micro-	10^{-18}	atto-
$10^9, 2^{30}$	Giga-	$10^{21}, 2^{70}$	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
$10^{12}, 2^{40}$	Tera-	$10^{24}, 2^{80}$	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except μ is used for micro.

BASIC INSTRUCTION FORMATS



FLOATING POINT INSTRUCTION FORMATS



PSEUDO INSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	`blt`	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	`bgt`	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	`ble`	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	`bge`	if(R[rs]>=R[rt]) PC = Label
Load Immediate	`li`	R[rd] = immediate
Move	`move`	R[rd] = R[rs]

REGISTERS		
0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call