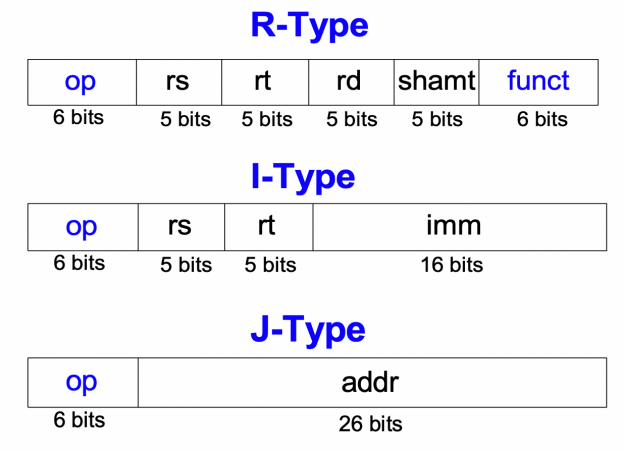
Instruction Format



Register Set

Name	Register Number	Usage
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2-3	Function return values
$a0-$a3	4-7	Function arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved variables
$t8-$t9	24-25	more temporaries
$k0-$k1	26-27	OS temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	Function return address

Instructions sorted by opcode

Opcode	Name	Description	Operation
000000 (0)	R-type	all R-type instructions	See the table below
000001 (1) (rt = 0/1)	bltz rs, label / bgez rs, label	branch less than zero/branch greater than or equal to zero	if ([rs] < 0) PC = BTA/ if ([rs] ≥ 0) PC = BTA
000010 (2)	j label	jump	PC = JTA
000011 (3)	jal label	jump and link	$ra = PC + 4, PC = JTA
000100 (4)	beq rs, rt, label	branch if equal	if ([rs] == [rt]) PC = BTA
000101 (5)	bne rs, rt, label	branch if not equal	if ([rs] != [rt]) PC = BTA
000110 (6)	blez rs, label	branch if less than or equal to zero	if ([rs] ≤ 0) PC = BTA
000111 (7)	bgtz rs, label	branch if greater than zero	if ([rs] > 0) PC = BTA
001000 (8)	addi rt, rs, imm	add immediate	[rt] = [rs] + SignImm
001001 (9)	addiu rt, rs, imm	add immediate unsigned	[rt] = [rs] + SignImm
001010 (10)	slti rt, rs, imm	set less than immediate	[rs] < SignImm ? [rt] = 1 : [rt] = 0
001011 (11)	sltiu rt, rs, imm	set less than immediate unsigned	[rs] < SignImm ? [rt] = 1 : [rt] = 0
001100 (12)	andi rt, rs, imm	and immediate	[rt] = [rs] & ZeroImm
001101 (13)	ori rt, rs, imm	or immediate	[rt] = [rs] | ZeroImm
001110 (14)	xori rt, rs, imm	xor immediate	[rt] = [rs] ^ ZeroImm
001111 (15)	lui rt, imm	load upper immediate	[rt] = {imm, 16'b0}
010000 (16) (rs = 0/4)	mfc0 rt, rd / mtc0 rt, rd	move from/to coprocessor 0	[rt] = [rd]/[rd] = [rt] (rd is in coprocessor 0)
010001 (17)	F-type	fop = 16/17: F-type instructions	See F-Type instructions Table
010001 (17) (rt = 0/1)	bc1f label/ bc1t label	fop = 8: branch if fpcond is FALSE/TRUE	if (fpcond == 0) PC = BTA/ if (fpcond == 1) PC = BTA
011100 (28) (func = 2)	mul rd, rs, rt	multiply (32-bit result)	[rd] = [rs] x [rt]
100000 (32)	lb rt, imm(rs)	load byte	[rt] = SignExt ($[Address]_{7:0}$)
100001 (33)	lh rt, imm(rs)	load halfword	[rt] = SignExt ($[Address]_{15:0}$)
100011 (35)	lw rt, imm(rs)	load word	[rt] = [Address]
100100 (36)	lbu rt, imm(rs)	load byte unsigned	[rt] = ZeroExt ($[Address]_{7:0}$)
100101 (37)	lhu rt, imm(rs)	load halfword unsigned	[rt] = ZeroExt ($[Address]_{15:0}$)
101000 (40)	sb rt, imm(rs)	store byte	$[Address]_{7:0} = [rt]_{7:0}$
101001 (41)	sh rt, imm(rs)	store halfword	$[Address]_{15:0} = [rt]_{15:0}$
101011 (43)	sw rt, imm(rs)	store word	[Address] = [rt]
110001 (49)	lwc1 ft, imm(rs)	load word to FP coprocessor 1	[ft] = [Address]
111001 (56)	swc1 ft, imm(rs)	store word to FP coprocessor 1	[Address] = [ft]

R-Type, sorted by function field

Funct	Name	Description	Operation
000000 (0)	`sll rd, rt, shamt`	shift left logical	`[rd] = [rt] << shamt`
000010 (2)	`srl rd, rt, shamt`	shift right logical	`[rd] = [rt] >> shamt`
000011 (3)	`sra rd, rt, shamt`	shift right arithmetic	`[rd] = [rt] >>> shamt`
000100 (4)	`sllv rd, rt, rs`	shift left logical variable	`[rd] = [rt] << [rs]`$_{4:0}$
000110 (6)	`srlv rd, rt, rs`	shift right logical variable	`[rd] = [rt] >> [rs]`$_{4:0}$
000111 (7)	`srav rd, rt, rs`	shift right arithmetic variable	`[rd] = [rt] >>> [rs]`$_{4:0}$
001000 (8)	`jr rs`	jump register	`PC = [rs]`
001001 (9)	`jalr rs`	jump and link register	`$ra = PC + 4, PC = [rs]`
001100 (12)	`syscall`	system call	`system call exception`
001101 (13)	`break`	break	`break exception`
010000 (16)	`mfhi rd`	move from hi	`[rd] = [hi]`
010001 (17)	`mthi rs`	move to hi	`[hi] = [rs]`
010010 (18)	`mflo rd`	move from lo	`[rd] = [lo]`
010011 (19)	`mtlo rs`	move to lo	`[lo] = [rs]`
011000 (24)	`mult rs, rt`	multiply	`{[hi], [lo]} = [rs] × [rt]`
011001 (25)	`multu rs, rt`	multiply unsigned	`{[hi], [lo]} = [rs] × [rt]`
011010 (26)	`div rs, rt`	divide	`[lo] = [rs]/[rt], [hi] = [rs]%[rt]`
011011 (27)	`divu rs, rt`	divide unsigned	`[lo] = [rs]/[rt], [hi] = [rs]%[rt]`
100000 (32)	`add rd, rs, rt`	add	`[rd] = [rs] + [rt]`
100001 (33)	`addu rd, rs, rt`	add unsigned	`[rd] = [rs] + [rt]`
100010 (34)	`sub rd, rs, rt`	subtract	`[rd] = [rs] - [rt]`
100011 (35)	`subu rd, rs, rt`	subtract unsigned	`[rd] = [rs] - [rt]`
100100 (36)	`and rd, rs, rt`	and	`[rd] = [rs] & [rt]`
100101 (37)	`or rd, rs, rt`	or	`[rd] = [rs] | [rt]`
100110 (38)	`xor rd, rs, rt`	xor	`[rd] = [rs] ^ [rt]`
100111 (39)	`nor rd, rs, rt`	nor	`[rd] = ~([rs] | [rt])`
101010 (42)	`slt rd, rs, rt`	set less than	`[rs] < [rt] ? [rd] = 1 : [rd] = 0`
101011 (43)	`sltu rd, rs, rt`	set less than unsigned	`[rs] < [rt] ? [rd] = 1 : [rd] = 0`

F-Type instructions

Funct	Name	Description	Operation
000000 (0)	`add.s fd, fs, ft` / `add.d fd, fs, ft`	FP add	`[fd] = [fs] + [ft]`
000001 (1)	`sub.s fd, fs, ft` / `sub.d fd, fs, ft`	FP subtract	`[fd] = [fs] - [ft]`
000010 (2)	`mul.s fd, fs, ft` / `mul.d fd, fs, ft`	FP multiply	`[fd] = [fs] × [ft]`
000011 (3)	`div.s fd, fs, ft` / `div.d fd, fs, ft`	FP divide	`[fd] = [fs]/[ft]`
000101 (5)	`abs.s fd, fs` / `abs.d fd, fs`	FP absolute value	`[fd] = ([fs] < 0) ? [-fs] : [fs]`
000111 (7)	`neg.s fd, fs` / `neg.d fd, fs`	FP negation	`[fd] = [-fs]`
111010 (58)	`c.seq.s fs, ft` / `c.seq.d fs, ft`	FP equality comparison	`fpcond = ([fs] == [ft])`
111100 (60)	`c.lt.s fs, ft` / `c.lt.d fs, ft`	FP less than comparison	`fpcond = ([fs] < [ft])`
111110 (62)	`c.le.s fs, ft` / `c.le.d fs, ft`	FP less than or equal comparison	`fpcond = ([fs] ≤ [ft])`

System Calls and Input output Operators

Service	Code in $v0	Arguments	Results
print_int	1	$a0 = integer to be printed	
print_float	2	$f12 = float to be printed	
print_double	3	$f12 = double to be printed	
print_string	4	$a0 = address of string in memory	
read_int	5		integer returned in $v0
read_float	6		float returned in $v0
read_double	7		double returned in $v0
read_string	8	$a0 = memory address of string input buffer $a1 = length of string buffer (n)	
sbrk	9	$a0 = amount	address in $v0
exit	10		