MIPS operands

Name	Example	Comments
32 registers	`$s0–$s7, $t0–$t9, $gp, $fp, $zero, $sp, $ra, $at, Hi, Lo`	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register `$zero` always equals 0. Register `$at` is reserved for the assembler to handle large constants. `Hi` and `Lo` contain the results of multiply and divide.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example		Meaning	Comments
Arithmetic	add	`add`	`$s1,$s2,$s3`	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	`sub`	`$s1,$s2,$s3`	$s1 = $s2 – $s3	Three operands; overflow detected
	add immediate	`addi`	`$s1,$s2,100`	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	`addu`	`$s1,$s2,$s3`	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	`subu`	`$s1,$s2,$s3`	$s1 = $s2 – $s3	Three operands; overflow undetected
	add immediate unsigned	`addiu`	`$s1,$s2,100`	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	`mfc0`	`$s1,$epc`	$s1 = $epc	Used to copy Exception PC plus other special registers
	multiply	`mult`	`$s2,$s3`	Hi, Lo = $s2 × $s3	64-bit signed product in Hi, Lo
	multiply unsigned	`multu`	`$s2,$s3`	Hi, Lo = $s2 × $s3	64-bit unsigned product in Hi, Lo
	divide	`div`	`$s2,$s3`	Lo = $s2 / $s3, Hi = $s2 mod $s3	Lo = quotient, Hi = remainder
	divide unsigned	`divu`	`$s2,$s3`	Lo = $s2 / $s3, Hi = $s2 mod $s3	Unsigned quotient and remainder
	move from Hi	`mfhi`	`$s1`	$s1 = Hi	Used to get copy of Hi
	move from Lo	`mflo`	`$s1`	$s1 = Lo	Used to get copy of Lo
Logical	and	`and`	`$s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; logical AND
	or	`or`	`$s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; logical OR
	and immediate	`andi`	`$s1,$s2,100`	$s1 = $s2 & 100	Logical AND reg, constant
	or immediate	`ori`	`$s1,$s2,100`	$s1 = $s2 | 100	Logical OR reg, constant
	shift left logical	`sll`	`$s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl`	`$s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Data transfer	load word	`lw`	`$s1,100($s2)`	$s1 = Memory[$s2+100]	Word from memory to register
	store word	`sw`	`$s1,100($s2)`	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte unsigned	`lbu`	`$s1,100($s2)`	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	`sb`	`$s1,100($s2)`	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	`lui`	`$s1,100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	`beq`	`$s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne`	`$s1,$s2,25`	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt`	`$s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	`slti`	`$s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1=0	Compare < constant; two's complement
	set less than unsigned	`sltu`	`$s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1=0	Compare less than; natural numbers
	set less than immediate unsigned	`sltiu`	`$s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Unconditional jump	jump	`j`	2500	go to 10000	Jump to target address
	jump register	`jr`	`$ra`	go to $ra	For switch, procedure return
	jump and link	`jal`	2500	$ra = PC + 4; go to 10000	For procedure call

MIPS floating-point operands

Name	Example	Comments
32 floating-point registers	$f0, $f1, $f2, . . . , $f31	MIPS floating-point registers are used in pairs for double precision numbers.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS floating-point assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	add.s $f2,$f4,$f6	$f2 = $f4 + $f6	FP add (single precision)
	FP subtract single	sub.s $f2,$f4,$f6	$f2 = $f4 – $f6	FP sub (single precision)
	FP multiply single	mul.s $f2,$f4,$f6	$f2 = $f4 × $f6	FP. multiply (single precision)
	FP divide single	div.s $f2,$f4,$f6	$f2 = $f4 / $f6	FP divide (single precision)
	FP add double	add.d $f2,$f4,$f6	$f2 = $f4 + $f6	FP add (double precision)
	FP subtract double	sub.d $f2,$f4,$f6	$f2 = $f4 – $f6	FP sub (double precision)
	FP multiply double	mul.d $f2,$f4,$f6	$f2 = $f4 × $f6	FP multiply (double precision)
	FP divide double	div.d $f2,$f4,$f6	$f2 = $f4 / $f6	FP divide (double precision)
Data transfer	load word copr. 1	lwc1 $f1,100($s2)	$f1 = Memory[$s2 + 100]	32-bit data to FP register
	store word copr. 1	swc1 $f1,100($s2)	Memory[$s2 + 100] = $f1	32-bit data to memory
Conditional branch	branch on FP true	bc1t 25	if (cond == 1) go to PC + 4 + 100	PC-relative branch if FP cond.
	branch on FP false	bc1f 25	if (cond == 0) go to PC + 4 + 100	PC-relative branch if not cond.
	FP compare single (eq,ne,lt,le,gt,ge)	c.lt.s $f2,$f4	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than single precision
	FP compare double (eq,ne,lt,le,gt,ge)	c.lt.d $f2,$f4	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than double precision

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

Remaining MIPS I	Name	Format	Pseudo MIPS	Name	Format
exclusive or ($rs \oplus rt$)	xor	R	move	move	rd,rs
exclusive or immediate	xori	I	absolute value	abs	rd,rs
nor ($\neg(rs \vee rt)$)	nor	R	not ($\neg rs$)	not	rd,rs
shift right arithmetic	sra	R	negate *(signed or unsigned)*	neg*s*	rd,rs
shift left logical variable	sllv	R	rotate left	rol	rd,rs,rt
shift right logical variable	srlv	R	rotate right	ror	rd,rs,rt
shift right arith. variable	srav	R	mult. & don't check oflw *(signed or uns.)*	mul*s*	rd,rs,rt
			multiply & check oflw *(signed or uns.)*	mulo*s*	rd,rs,rt
move to Hi	mthi	R	divide and check overflow	div	rd,rs,rt
move to Lo	mtlo	R	divide and don't check overflow	divu	rd,rs,rt
load halfword	lh	I	remainder *(signed or unsigned)*	rem*s*	rd,rs,rt
load halfword unsigned	lhu	I	load immediate	li	rd,imm
store halfword	sh	I	load address	la	rd,addr
load word left *(unaligned)*	lwl	I	load double	ld	rd,addr
load word right *(unaligned)*	lwr	I	store double	sd	rd,addr
store word left *(unaligned)*	swl	I	unaligned load word	ulw	rd,addr
store word right *(unaligned)*	swr	I	unaligned store word	usw	rd,addr
branch on less than zero	bltz	I	unaligned load halfword *(signed or uns.)*	ulh*s*	rd,addr
branch on less or equal zero	blez	I	unaligned store halfword	ush	rd,addr
branch on greater than zero	bgtz	I	branch	b	Label
branch on ≥ zero	bgez	I	branch on equal zero	beqz	rs,L
branch on ≥ zero and link	bgezal	I	branch on ≥ *(signed or unsigned)*	bge*s*	rs,rt,L
branch on < zero and link	bltzal	I	branch on > *(signed or unsigned)*	bgt*s*	rs,rt,L
jump and link register	jalr	R	branch on ≤ *(signed or unsigned)*	ble*s*	rs,rt,L
return from exception	rfe	R	branch on < *(signed or unsigned)*	blt*s*	rs,rt,L
system call	syscall	R	set equal	seq	rd,rs,rt
break *(cause exception)*	break	R	set not equal	sne	rd,rs,rt
move from FP to integer	mfc1	R	set greater or equal *(signed or unsigned)*	sge*s*	rd,rs,rt
move to FP from integer	mtc1	R	set greater than *(signed or unsigned)*	sgt*s*	rd,rs,rt
FP move *(s or d)*	mov.*f*	R	set less or equal *(signed or unsigned)*	sle*s*	rd,rs,rt
FP absolute value *(s or d)*	abs.*f*	R	set less than *(signed or unsigned)*	sle*s*	rd,rs,rt
FP negate *(s or d)*	neg.*f*	R	load to floating point *(s or d)*	l.*f*	rd,addr
FP convert *(w, s, or d)*	cvt.*ff*	R	store from floating point *(s or d)*	s.*f*	rd,addr
FP compare un *(s or d)*	c.xn.*f*	R			

31 26	25 21	20 16	15 11	10 6	5 0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type

Load and Store Instructions

100000	base	dest	signed offset	LB rt, offset(rs)
100001	base	dest	signed offset	LH rt, offset(rs)
100011	base	dest	signed offset	LW rt, offset(rs)
100100	base	dest	signed offset	LBU rt, offset(rs)
100101	base	dest	signed offset	LHU rt, offset(rs)
101000	base	dest	signed offset	SB rt, offset(rs)
101001	base	dest	signed offset	SH rt, offset(rs)
101011	base	dest	signed offset	SW rt, offset(rs)

I-Type Computational Instructions

001001	src	dest	signed immediate	ADDIU rt, rs, signed-imm.
001010	src	dest	signed immediate	SLTI rt, rs, signed-imm.
001011	src	dest	signed immediate	SLTIU rt, rs, signed-imm.
001100	src	dest	zero-ext. immediate	ANDI rt, rs, zero-ext-imm.
001101	src	dest	zero-ext. immediate	ORI rt, rs, zero-ext-imm.
001110	src	dest	zero-ext. immediate	XORI rt, rs, zero-ext-imm.
001111	00000	dest	zero-ext. immediate	LUI rt, zero-ext-imm.

R-Type Computational Instructions

000000	00000	src	dest	shamt	000000	SLL rd, rt, shamt
000000	00000	src	dest	shamt	000010	SRL rd, rt, shamt
000000	00000	src	dest	shamt	000011	SRA rd, rt, shamt
000000	rshamt	src	dest	00000	000100	SLLV rd, rt, rs
000000	rshamt	src	dest	00000	000110	SRLV rd, rt, rs
000000	rshamt	src	dest	00000	000111	SRAV rd, rt, rs
000000	src1	src2	dest	00000	100001	ADDU rd, rs, rt
000000	src1	src2	dest	00000	100011	SUBU rd, rs, rt
000000	src1	src2	dest	00000	100100	AND rd, rs, rt
000000	src1	src2	dest	00000	100101	OR rd, rs, rt
000000	src1	src2	dest	00000	100110	XOR rd, rs, rt
000000	src1	src2	dest	00000	100111	NOR rd, rs, rt
000000	src1	src2	dest	00000	101010	SLT rd, rs, rt
000000	src1	src2	dest	00000	101011	SLTU rd, rs, rt

Jump and Branch Instructions

000010	target					J target
000011	target					JAL target
000000	src	00000	00000	00000	001000	JR rs
000000	src	00000	dest	00000	001001	JALR rd, rs
000100	src1	src2	signed offset			BEQ rs, rt, offset
000101	src1	src2	signed offset			BNE rs, rt, offset
000110	src	00000	signed offset			BLEZ rs, offset
000111	src	00000	signed offset			BGTZ rs, offset
000001	src	00000	signed offset			BLTZ rs, offset
000001	src	00001	signed offset			BGEZ rs, offset

	0	rs	rt	rd	shamt	funct
Bit positions	31–26	25–21	20–16	15–11	10–6	5–0

a. R-type instruction

	35 or 43	rs	rt	address
Bit positions	31–26	25–21	20–16	15–0

b. Load or store instruction

	4	rs	rt	address
Bit positions	31–26	25–21	20–16	15–0

c. Branch instruction

d. Jump Instruction	2	address

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	010
SW	00	store word	XXXXXX	add	010
Branch equal	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less than	101010	set on less than	111