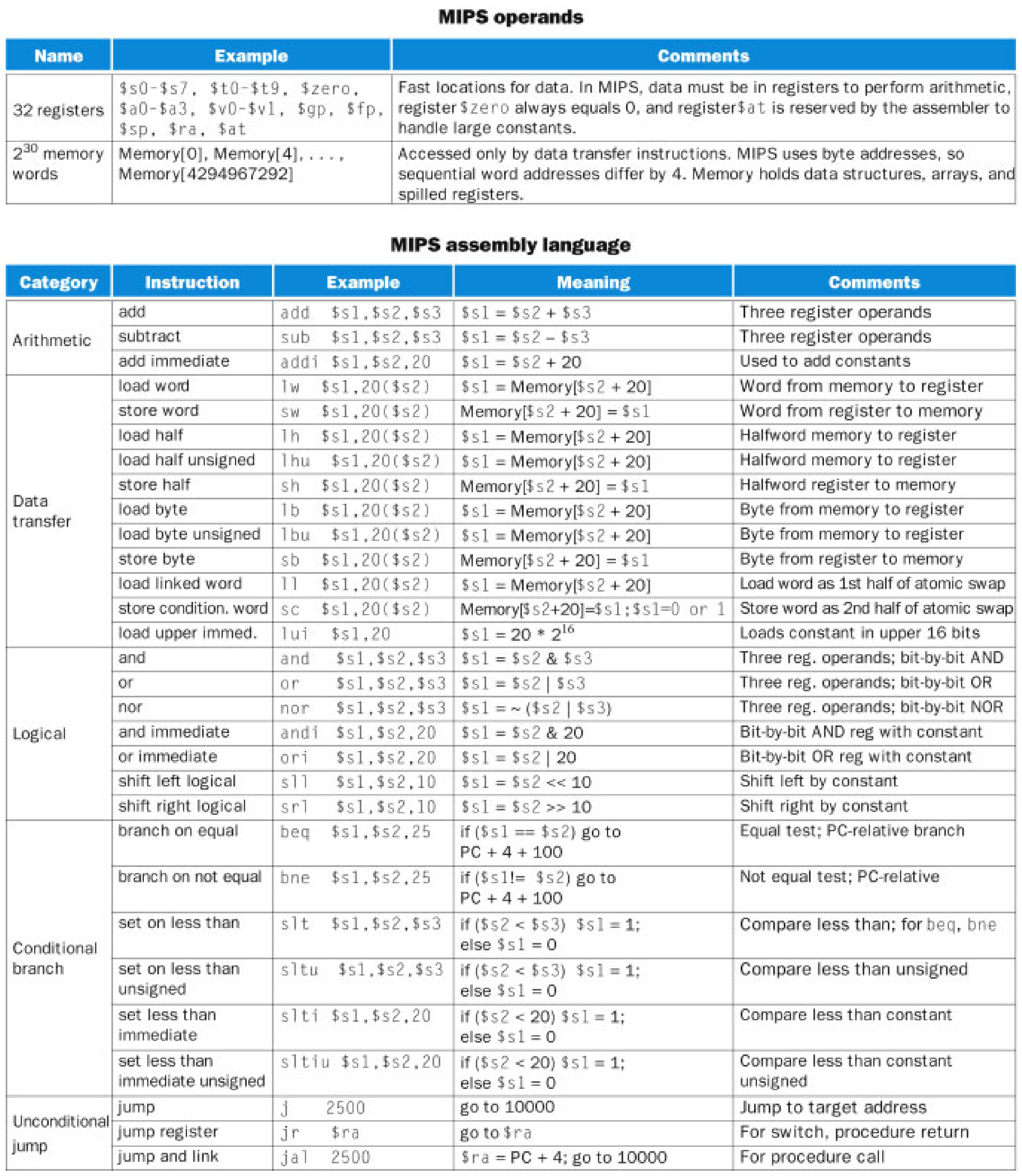

MIPS operands

Name	Example	Comments
32 registers	`$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at`	Fast locations for data. In MIPS, data must be in registers to perform arithmetic, register `$zero` always equals 0, and register `$at` is reserved by the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	`$s1` = `$s2` + `$s3`	Three register operands
	subtract	`sub $s1,$s2,$s3`	`$s1` = `$s2` – `$s3`	Three register operands
	add immediate	`addi $s1,$s2,20`	`$s1` = `$s2` + 20	Used to add constants
Data transfer	load word	`lw $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Word from memory to register
	store word	`sw $s1,20($s2)`	Memory[`$s2` + 20] = `$s1`	Word from register to memory
	load half	`lh $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Halfword memory to register
	load half unsigned	`lhu $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Halfword memory to register
	store half	`sh $s1,20($s2)`	Memory[`$s2` + 20] = `$s1`	Halfword register to memory
	load byte	`lb $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Byte from memory to register
	load byte unsigned	`lbu $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Byte from memory to register
	store byte	`sb $s1,20($s2)`	Memory[`$s2` + 20] = `$s1`	Byte from register to memory
	load linked word	`ll $s1,20($s2)`	`$s1` = Memory[`$s2` + 20]	Load word as 1st half of atomic swap
	store condition. word	`sc $s1,20($s2)`	Memory[`$s2`+20]=`$s1`;`$s1`=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	`lui $s1,20`	`$s1` = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	`$s1` = `$s2` & `$s3`	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	`$s1` = `$s2` | `$s3`	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	`$s1` = ~ (`$s2` | `$s3`)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,20`	`$s1` = `$s2` & 20	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,20`	`$s1` = `$s2` | 20	Bit-by-bit OR reg with 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
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; for `beq`, `bne`
	set on less than unsigned	`sltu $s1,$s2,$s3`	if (`$s2` < `$s3`) `$s1` = 1; else `$s1` = 0	Compare less than unsigned
	set less than immediate	`slti $s1,$s2,20`	if (`$s2` < 20) `$s1` = 1; else `$s1` = 0	Compare less than constant
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if (`$s2` < 20) `$s1` = 1; else `$s1` = 0	Compare less than constant unsigned
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

load link ll $rt, 0($rs)

> loads the content of memory location 0($rs) into register $rt, and saves the memory address into a link register

store conditional sc $rt, 0($rs)

> stores register $rt into the memory location 0($rs) only if it is the first store after the ll instruction. In other words, the store succeeds only if the memory location has not been changed by another store instruction.
>
> Succeeds if memory location not changed since the ll instruction and sets $rt to 1
>
> Fails if memory location is changed and sets $rt to 0

MIPS Instruction Subset Encodings

Register name	Description	Register number
$zero	Constant zero	0
$v0 – $v1	Result values	2 – 3
$a0 – $a3	Arguments	4 – 7
$t0 – $t7, $t8 – $t9	Temporaries can be overwritten by callee	8 – 15, 24 – 25
$s0 – $s7	Saved must be saved/restored by callee	16 – 23
$sp	Stack pointer	29
$ra	Return address	31

instruction	op	rs	rt	rd	shamt	funct
Width	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Bits	31-26	25-21	20-16	15-11	10-6	5-0
add $t0, $s1, $s2	0*	$s1 reg 17	$s2 reg 18	$t0 8	0	32
sub $s0, $s1, $s2	0	$s1 reg 17	$s2 reg 18	$s0 reg 16	0	34
and $s0, $s1, $s2	0	$s1 reg 17	$s2 reg 18	$s0 reg 16	0	36
or $s0, $s1, $s2	0	$s1 reg 17	$s2 reg 18	$s0 reg 16	0	37
nor $s0, $s1, $s2	0	$s1 reg 17	$s2 reg 18	$s0 reg 16	0	39
addi $s1, $s2, 100	8	$s2 reg 18	$s1 reg 17	100		
sll $t2, $s0, 4	0	0	$s0 reg 16	$t2 reg 10	4	0
srl $t2, $s0, 4	0	0	$s0 reg 16	$t2 reg 10	4	2
lw $t0, 1200($t1)	35	$t1 reg 9	$t0 reg 8	1200		
sw $t0, 1200($t1)	43	$t1 reg 9	$t0 reg 8	1200		
j 12345	2	absolute address 12345				
bne $s0, $s1, exit	5	$s0 16	$s1 17	Number of words to label from the next word		
beq $s0, $s1, exit	4	$s0 16	$s1 17	Number of words to label from the next word		
slt $s0, $s1, $s2	0	$s1 17	$s2 18	$s0 16	0	42
ll $s1, 4($s0)	48	$s0 16	$s1 17	4		
sc $s1, 4($s0)	56	$s0 16	$s1 17	4		

* all values are in decimal