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 = $epc	Copy Exception PC + special regs
	multiply	`mult`	`$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`	$s1 = Lo	Used to get copy of Lo
Data transfer	load word	`lw`	`$s1,20($s2)`	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	`sw`	`$s1,20($s2)`	Memory[$s2 + 20] = $s1	Word from register to memory
	load half unsigned	`lhu`	`$s1,20($s2)`	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	`sh`	`$s1,20($s2)`	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte unsigned	`lbu`	`$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 conditional word	`sc`	`$s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half atomic swap
	load upper immediate	`lui`	`$s1,100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	AND	`AND`	`$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,$s2,100`	$s1 = $s2 & 100	Bit-by-bit AND with constant
	OR immediate	`ORi`	`$s1,$s2,100`	$s1 = $s2 | 100	Bit-by-bit OR with constant
	shift left logical	`sll`	`$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 $s1 = 0	Compare less than; two's complement
	set less than immediate	`slti`	`$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

FIGURE 3.12 MIPS core architecture. The memory and registers of the MIPS architecture are not included for space reasons, but this section added the Hi and Lo registers to support multiply and divide. MIPS machine language is listed in the MIPS Reference Data Card at the front of this book.