MIPS operands

Name	Example	Comments
32 registers	`$s0, $s1, . . . , $s7` `$t0, $t1, . . . , $t7,` `$zero`	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers `$s0`–`$s7` map to 16–23 and `$t0`–`$t7` map to 8–15. MIPS register `$zero` always equals 0.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions in MIPS. 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 operands; data in registers
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 – $s3`	Three operands; data in registers
Data transfer	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Data from memory to register
	store word	`sw $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Data from register to memory
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 reg with constant
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	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,L`	`if ($s1 == $s2) go to L`	Equal test and branch
	branch on not equal	`bne $s1,$s2,L`	`if ($s1 != $s2) go to L`	Not equal test and branch
	set on less than	`slt $s1,$s2,$s3`	`if ($s2 < $s3) $s1 = 1;` `else $s1 = 0`	Compare less than; used with `beq, bne`
	set on less than immediate	`slt $s1,$s2,100`	`if ($s2 < 100) $s1 = 1;` `else $s1 = 0`	Compare less than immediate; used with `beq, bne`
Unconditional jump	jump	`j L`	go to L	Jump to target address

FIGURE 2.12 MIPS architecture revealed through Section 2.6. Highlighted portions show MIPS structures introduced in Section 2.6.

Check Yourself

C has many statements for decisions and loops while MIPS has few. Which of the following do or do not explain this imbalance? Why?

1. More decision statements make code easier to read and understand.
2. Fewer decision statements simplify the task of the underlying layer that is responsible for execution.