Resumen de instrucciones MIPS 32

Jesus Rachadell

6 de junio de 2025

1. Introducción

Los computadores ejecutan operaciones mediante instrucciones, cuyo conjunto se denomina repertorio de instrucciones. MIPS es una arquitectura clásica basada en:

- Simplicidad: Cada instrucción realiza una sola operación.
- Rigidez: Sintaxis fija (3 operandos en operaciones básicas).
- Eficiencia: Diseño orientado a alto rendimiento y bajo consumo.

A diferencia de lenguajes de alto nivel, MIPS opera directamente sobre el hardware, reflejando el concepto de programa almacenado. Compararemos brevemente MIPS con ARM y x86, destacando su relevancia en sistemas embebidos y PCs.

2. Instrucciones

Nombre	**Ejemplo**	**Concepto**
32 registros	$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at	Registros y "variables"
2^{30} palabras de memoria	Memory[0], Memory[4], ..., Memory[4294967292]	Las direcciones de palabras consecutivas se diferencian en 4. La memoria guarda las estructuras de datos, las tablas y los registros desbordados (guardados).

Cuadro 1: Descripción de registros y memoria en MIPS.

Cuadro 2: Resumen de instrucciones MIPS

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Artimética	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Tres operandos; datos en registros
	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Tres operandos; datos en registros
	add immediate	`addi $s1, $s2, 100`	`$s1 = $s2 + 100`	Usado para sumar constantes
Transferencia de dato	load word	`lw $s1, 100($s2)`	`$s1 = Memory[$s2 + 100]`	Palabra de memoria a registro
	store word	`sw $s1, 100($s2)`	`Memory[$s2 + 100] = $s1`	Palabra de registro a memoria
	load half	`lh $s1, 100($s2)`	`$s1 = Memory[$s2 + 100]`	Media palabra de memoria a registro
	store half	`sh $s1, 100($s2)`	`Memory[$s2 + 100] = $s1`	Media palabra de registro a memoria
	load byte	`lb $s1, 100($s2)`	`$s1 = Memory[$s2 + 100]`	Byte de memoria a registro
	store byte	`sb $s1, 100($s2)`	`Memory[$s2 + 100] = $s1`	Byte de registro a memoria
Lógica	and	`and $s1, $s2, $s3`	`$s1 = $s2 & $s3`	AND bit a bit
	or	`or $s1, $s2, $s3`	`$s1 = $s2 | $s3`	OR bit a bit
	nor	`nor $s1, $s2, $s3`	`$s1 = ~($s2 | $s3)`	NOR bit a bit
	and immediate	`andi $s1, $s2, 100`	`$s1 = $s2 & 100`	AND con constante
	or immediate	`ori $s1, $s2, 100`	`$s1 = $s2 | 100`	OR con constante
	shift left logical	`sll $s1, $s2, 10`	`$s1 = $s2 ≪ 10`	Desplazamiento izquierdo
	shift right logical	`srl $s1, $s2, 10`	`$s1 = $s2 ≫ 10`	Desplazamiento derecho
Salto condicional	branch on equal	`beq $s1, $s2, L`	`if ($s1 == $s2) go to L`	Comprueba igualdad y bifurca relativo al PC
	branch on not equal	`bne $s1, $s2, L`	`if ($s1 != $s2) go to L`	Comprueba si no igual y bifurca relativo al PC
	set on less than	`slt $s1, $s2, $s3`	`if ($s2 <$s3) $s1 = 1; else $s1 = 0`	Compara si es menor que; usado con beq, bne
	set on less than immediate	`slti $s1, $s2, 100`	`if ($s2 <100) $s1 = 1; else $s1 = 0`	Compara si es menor que una constante
Salto incondicional	jump	`j 2500`[2]	`go to 10000`	Salto a la dirección destino
	jump register	`jr $ra`	`go to $ra`	Para retorno de procedimiento
	jump and link	`jal 2500`	`$ra = PC + 4; go to 10000`	Para llamada a procedimiento