Mips

Samuel Primera CI: 31.129.684

1 de junio de 2025

1. Repertorio de Instrucciones

El repertorio de instrucciones de un procesador es el conjunto de operaciones que pueden ejecutarse directamente en el hardware. Estas instrucciones incluyen cálculos aritméticos, manipulaciones de memoria y control de flujo de ejecución.

Los lenguajes de programación de alto nivel como **C o Java** permiten a los desarrolladores escribir código de manera más intuitiva. Sin embargo, para que el hardware ejecute estos programas, el código debe traducirse a lenguaje ensamblador y luego a lenguaje de máquina.

El proceso de traducción ocurre en varias etapas:

1. **Código fuente**: Escrito en un lenguaje como C.
2. **Lenguaje ensamblador**: Representa instrucciones en un formato más cercano al hardware.
3. **Lenguaje de máquina**: Conjunto de bits interpretados por el procesador.

2. Operaciones en MIPS

MIPS es una arquitectura basada en el paradigma **RISC (Reduced Instruction Set Computing)**, lo que implica un repertorio de instrucciones simplificado, con una estructura clara y uniforme.

Ejemplo de suma en lenguaje ensamblador:

```
add a, b, c   # Suma b y c, almacena el resultado en a
```

En MIPS, cada instrucción:

- Ejecuta **una sola operación**.
- Tiene **exactamente tres operandos** (dos fuentes y un destino).
- Se estructura de manera predecible para optimizar el hardware.

Si se requiere la suma de cuatro variables, se emplean varias instrucciones:

```
add a, b, c   # Suma b y c
add a, a, d   # Suma el resultado anterior con d
add a, a, e   # Suma el resultado anterior con e
```

Este diseño permite una implementación eficiente en hardware, simplificando los circuitos y mejorando la velocidad de ejecución.

2.1. Operandos Mips

Nombre	Ejemplo	Comentarios
32 registros	`$s0–$s7`, `$t0–$t9`, `$zero`, `$a0–$a3`, `$v0–$v1`, `$gp`, `$fp`, `$sp`, `$ra`, `$at`	Localizaciones rápidas para los datos. En MIPS, los datos deben estar en los registros para realizar operaciones aritméticas. El registro MIPS `$zero` es siempre igual a 0. El registro `$at` está reservado por el ensamblador para manejar constantes grandes.
2^{30} palabras de memoria	Memory[0], Memory[4], . . . , Memory[4294967292]	Accesibles solamente por instrucciones de transferencia de datos. MIPS utiliza direcciones de byte, de modo que las direcciones de palabras consecutivas se diferencian en 4. La memoria guarda las estructuras de datos, las tablas y los registros desbordados (guardados).

2.2. Lenguaje Esamblador Mips

Categoría	Instrucción	Ejemplo	Significado	Comentarios
Aritmética	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 memória
	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
	load upper immed.	`lui $s1,100`	$s1 = 100 * 2^{16}	Cargar constante en los 16 bits de mayor peso
Lógica	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Tres registros operandos; AND bit-a-bit
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Tres registros operandos; OR bit-a-bit
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 |$s3)	Tres registros operandos; NOR bit-a-bit
	and immediate	`andi $s1,$s2,100`	$s1 = $s2 & 100	AND Bit-a-bit registro con constante
	or immediate	`ori $s1,$s2,100`	$s1 = $s2 | 100	OR Bit-a-bit registro con constante
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Desplazamiento a la izquierda por constante
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Desplazamiento a la derecha por constante
Salto condicional	branch on equal	`beq $s1,$s2,L`	if ($s1 == $s2) go to L PC + 4 + 100	Comprueba igualdad y bifurca relativo al PC
	branch on not equal	`bne $s1,$s2,L`	if ($s1 != $s2) go to L PC + 4 + 100	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	`slt $s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compara si es menor que una constante
Salto incondicional	jump	`j 2500`	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