Hoja de Referencia MIPS32

3 de Diciembre de 2025

I. OPERANDOS

A. *Registros*

Registro	Numero	Descripción
$z0	0	Valor constante 0
$at	1	Ensamblador Temporal
$v0 - $v1	2 - 3	Valores de resultado de funciones y evaluación de expresiones
$a0 - $a3	4 - 7	Argumentos
$t0 - $t7	8 - 15	Temporales (no se guarda el valor entre llamadas)
$s0 - $s7	16 - 23	Temporales almacenados (no se guarda el valor entre llamadas)
$t8 - $t9	24 - 25	Temporales (no se guarda el valor entre llamadas)
$k0 - $k1	26 - 27	Reservados para el núcleo del Sistema Operativo
$gp	28	Puntero al área global
$sp	29	Puntero de pila
$fp	30	Puntero de marco de pila
$ra	31	Dirección de retorno, usada por llamadas a función

B. *Palabras*

El tamaño de un registro en *MIPS* es de 32 *bits*, dada la frecuencia de grupos de este tamaño, se les ha dado un nombre, "**Palabra**".

Palabra (*word* en ingles): Unidad natural de acceso en un computador, normalmente un grupo de 32 *bits* (4 *bytes*), corresponde al tamaño de un registro en la arquitectura *MIPS*.

II. INSTRUCCIONES

A. *Formatos básicos de instrucción*

- Formato Tipo R: En este formato las instrucciones sólo utilizan registos. Los campos que componen a este formato son:

– **co**: Código de operación

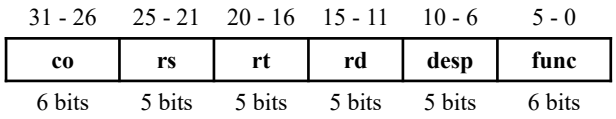
– **rs**: Primera referencia a un registro fuente

– **rt**: Segunda referencia a un registro fuente

– **rd**: Referencia a un registro destino

– **desp**: Valor numérico utilizado en las instrucciones de desplazamiento

– **func**: Campo de función utilizado para seleccionar una variante del código de operación



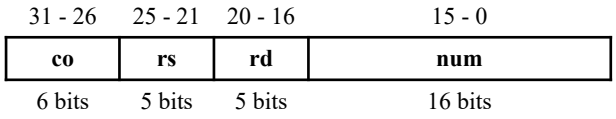
- Formato Tipo I: En este formato las instrucciones utilizan dos registos y un valor inmediato de 16 bits. Los campos que componen a este formato son:

– **co**: Código de operación

– **rs**: Primera referencia a un registro fuente

– **rd**: Referencia a un registro destino

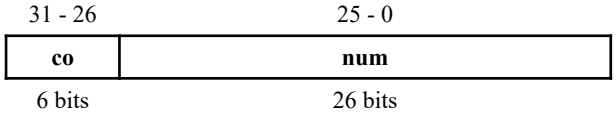
– **num**: Valor inmediato



- Formato Tipo J: Este formato solo se utiliza con instrucciones de salto incondicional. Los campos que componen a este formato son:

– **co**: Código de operación

– **num**: Valor inmediato utilizado para el cálculo de la dirección de salto



- Pseudoinstrucciones: Variación común de las instrucciones del lenguaje ensamblador, generalmente tratadas como si fueran instrucciones de pleno derecho. No estan implementadas en hardware pero su aparición en el ensamblador simplifica la traducción y programación.

B. Repertorio de instrucciones

1) Instrucciones de aritmética:

- De Suma:

– Suma (`add`): Instrucción de tipo R, suma dos operandos, `$s2` y `$s2`, almacenando el resultado en `$s3`. Tiene la siguiente estructura:

```
add $s1, $s2, $s3 # $s1 ← $s2 + $s3
```

– Suma inmediata (`addi`): Instrucción de tipo I, suma dos operandos, un registro (`$s2`), y una constante entera k, tiene la siguiente estructura (ejemplo usando $k = 100$):

```
addi $s1, $s2, 100 # $s1 ← $s2 + 100
```

– Suma sin signo (`addu`): Instrucción de tipo R, suma dos operandos, `$s2` y `$s2`, almacenando el resultado en `$s3`. Tiene la siguiente estructura:

```
addu $s1, $s2, $s3 # $s1 ← $s2 + $s3
```

– Suma inmediata sin signo (`addiu`): Instrucción de tipo I, suma dos operandos, un registro (`$s2`), y una constante entera k, tiene la siguiente estructura (ejemplo usando $k = 100$):

```
addiu $s1, $s2, 100 # $s1 ← $s2 + 100
```

- De resta

– Resta (`sub`): Instrucción de tipo R, resta dos operandos, `$s2` y `$s2`, almacenando el resultado en `$s3`. Tiene la siguiente estructura:

```
sub $s1, $s2, $s3 # $s1 ← $s2 - $s3
```

– Resta sin signo (`subu`): Instrucción de tipo R, resta dos operandos, `$s2` y `$s2`, almacenando el resultado en `$s3`. Tiene la siguiente estructura:

```
subu $s1, $s2, $s3 # $s1 ← $s2 - $s3
```

- De multiplicación y división

– Multiplicación (`mult`): Instrucción de tipo R, multiplica dos operandos, la parte baja del resultado se deja en el registro `lo`, y la parte alta en el registro `hi`

```
mult $s1, $s2 # {Hi, Lo} ← $s1 * $s2
```

– Multiplicación sin signo (`multu`): Instrucción de tipo R, multiplica dos operandos, la parte baja del resultado se deja en el registro `lo`, y la parte alta en el registro `hi`

```
multu $s1, $s2 # {Hi, Lo} ← $s1 * $s2
```

– División (`div`): Instrucción de tipo R, divide dos operandos, deja a el cociente en el registro `lo` y el resto en el registro `hi`

```
div $s1, $s2 # Lo ← $s1 / $s2, Hi ← $s1 % $s2
```

– División sin signo (`divu`): Instrucción de tipo R, divide dos operandos, deja a el cociente en el registro `lo` y el resto en el registro `hi`

```
divu $s1, $s2 # Lo ← $s1 / $s2, Hi ← $s1 % $s2
```

– Mover de parte alta (`mfhi`): Instrucción de tipo R, copia el contenido del registro `hi`, hasta uno dado

```
mfhi $s0 # $s0 ← Hi
```

– Mover de parte baja (`mflo`): Instrucción de tipo R, copia el contenido del registro `lo`, hasta uno dado

```
mflo $s0 # $s0 ← Lo
```

2) Instrucciones de acceso a memoria:

- De carga

– Carga de un byte (`lb`): Instrucción de tipo I. Carga un *Byte* de memoria a un registro, extiende el signo

```
lb $s1,k($s2) # $s1 ← Memory[$s2 + k]
```

– Carga de un byte sin signo (`lbu`): Instrucción de tipo I. Carga un *Byte* de memoria a un registro, no extiende el signo

```
lbu $s1,k($s2) # $s1 ← Memory[$s2 + k]
```

– Carga de una palabra (`lw`): Instrucción de tipo I. Carga una palabra de memoria a un registro, extiende el signo

```
lw $s1,k($s2) # $s1 ← Memory[$s2 + k]
```

– Carga de media palabra (`lh`): Instrucción de tipo I. Carga una media palabra (2 *bytes*) de memoria a un registro, extiende el signo

```
lh $s1,k($s2) # $s1 ← Memory[$s2 + k]
```

– Carga de media palabra sin signo (`lhu`): Instrucción de tipo I. Carga una media palabra (2 *bytes*) de memoria a un registro, no extiende el signo

```
lhu $s1,k($s2) # $s1 ← Memory[$s2 + k]
```

- De almacenado

– Guarda un byte (`sb`): Instrucción de tipo I. Carga en memoria un byte de un registro

```
sb $s1,k($s2) # Memory[$s2 + k] ← $s1
```

– Guarda una palabra (`sw`): Instrucción de tipo I. Carga en memoria una palabra de un registro

```
sw $s1,k($s2) # Memory[$s2 + k] ← $s1
```

– Guarda una media palabra (`sh`): Instrucción de tipo I. Carga en memoria una media palabra de un registro

```
sh $s1,k($s2) # Memory[$s2 + k] ← $s1
```

3) Instrucciones de lógica:

– And (`and`): Instrucción de tipo R, guarda en `$s0` el resultado de efectuar la operación lógica `$s1` ∧ `$s2`, tiene la siguiente estructura:

```
and $s0, $s1, $s2 # $s0 ← $s1 ∧ $s2
```

– And inmediato (`andi`): Instrucción de tipo I, guarda en `$s0` el resultado de efectuar la operación lógica `$s1` ∧ *k*, donde *k* es una constante dada, tiene la siguiente estructura:

```
andi $s0, $s1, k # $s0 ← $s1 ∧ k
```

– Or (`or`): Instrucción de tipo R, guarda en `$s0` el resultado de efectuar la operación lógica `$s1` ∨ `$s2`, tiene la siguiente estructura:

```
or $s0, $s1, $s2 # $s0 ← $s1 ∨ $s2
```

– Or inmediato (`ori`): Instrucción de tipo I, guarda en `$s0` el resultado de efectuar la operación lógica `$s1` ∨ *k*, donde *k* es una constante dada, tiene la siguiente estructura:

```
ori $s0, $s1, k # $s0 ← $s1 ∧ k
```

– Or exclusivo (`xor`): Instrucción de tipo R, guarda en `$s0` el resultado de efectuar la operación lógica `$s1` ⊻ `$s2`, tiene la siguiente estructura:

```
xor $s0, $s1, $s2 # $s0 ← $s1 ⊻ $s2
```

– Or inmediato exclusivo (`xori`): Instrucción de tipo I, guarda en `$s0` el resultado de efectuar la operación lógica `$s1` ⊻ *k*, donde *k* es una constante dada, tiene la siguiente estructura:

```
xori $s0, $s1, k # $s0 ← $s1 ⊻ k
```

– Nor (`nor`): Instrucción de tipo R, guarda en `$s0` el resultado de efectuar la operación lógica ¬(`$s1` ∨ `$s2`), tiene la siguiente estructura:

```
nor $s0, $s1, $s2 # $s0 ← ¬($s1 ∨ $s2)
```

– Desplazamiento lógico a la izquierda (`sll`): Instrucción de tipo R, mueve los *bits* del registro dado *k* posiciones a la izquierda, aquellas posiciones que son anuladas se rellenan con ceros, tiene la siguiente estructura:

```
sll $s1, $s2, k # $s1 ← $s2 << k
```

– Desplazamiento lógico a la derecha (`srl`): Instrucción de tipo R, mueve los *bits* del registro dado *k* posiciones a la derecha. En los numeros sin signo, las posiciones de bits que la operación de desplazamiento ha anulado son de relleno cero. En el caso de los números con signo, el bit de signo se usa para rellenar las posiciones de bits vacías, tiene la siguiente estructura:

```
srl $s1, $s2, k # $s1 ← $s2 >> k
```

4) Condicionales:

– Salto si igual (`beq`): Instrucción de tipo I, comprueba si dos registros son iguales, en cuyo caso, salta a otra parte del programa (puede ser o una etiqueta, o una posicion relativa a la llamada), tiene la siguiente estructura:

```
beq $s1, $s2, label # if ($s1 = $s2) go to label
```

– Salto si distinto (`bnq`): Instrucción de tipo I, comprueba si dos registros son distintos, en cuyo caso, salta a otra parte del programa (puede ser o una etiqueta, o una posicion relativa a la llamada), tiene la siguiente estructura:

```
beq $s1, $s2, label # if ($s1 ≠ $s2) go to label
```

– Salto si menor que (`blt`): Pseudoinstrucción, compara dos registros y si el primero es menor que el segundo, salta a la etiqueta

```
blt $s1, $s2, label # if ($s1 < $s2) go to label
```

– Salto si mayor que (`bgt`): Pseudoinstrucción, compara dos registros y si el primero es mayor que el segundo, salta a la etiqueta

```
bgt $s1, $s2, label # if ($s1 > $s2) go to label
```

– Salto si menor que o igual (`ble`): Pseudoinstrucción, compara dos registros y si el primero es menor o igual que el segundo, salta a la etiqueta

```
ble $s1, $s2, label # if ($s1 ≤ $s2) go to label
```

– Salto si mayor que o igual (`bge`): Pseudoinstrucción, compara dos registros y si el primero es mayor o igual que el segundo, salta a la etiqueta

```
bge $s1, $s2, label # if ($s1 ≥ $s2) go to label
```

– Fijar si menor que (`slt`): Instrucción de tipo R, compara dos registros `$s2` y `$s3`, si $s2 < $s3, a `$s1` le asigna 1, si no, le asigna 0, tiene la siguiente estructura:

```
slt $s1, $s2, $s3 # if ($s2 < $s3) $s1 ← 1 else $s2 ← 0
```

– Fijar si menor que inmediato (`slti`): Instrucción de tipo I, compara un registro `$s2` y una constante *k*, si $s2 < *k*, a `$s1` le asigna 1, si no, le asigna 0, tiene la siguiente estructura:

```
slti $s1, $s2, k # if ($s2 < k) $s1 ← 1 else $s2 ← 0
```

– Fijar si menor que sin signo (`sltu`): Instrucción de tipo R, compara dos registros `$s2` y `$s3`, si `$s2` < `$s3`, a `$s1` descartando los signos, le asigna 1, si no, le asigna 0, tiene la siguiente estructura:

```
sltu $s1, $s2, $s3 # if ($s2 < $s3) $s1 ← 1 else $s2 ← 0
```

– Fijar si menor que sin signo inmediato (`sltui`): Instrucción de tipo I, compara un registro `$s2` y una constante *k*, si `$s2` < *k*, a `$s1` descartando los signos, le asigna 1, si no, le asigna 0, tiene la siguiente estructura:

```
sltui $s1,$s2,$s3 # if ($s2 < $s3) $s1 ← 1 else $s1 ← 0
```

5) Saltos incondicionales:

– Salto incondicional (`j`): Instrucción de tipo J, salta a la dirección destino, tiene la siguiente estructura:

```
j label # go to label
```

– Saltar y enlanzar (`jal`): Instrucción de tipo J, se usa para llamar a subrutinas/funciones, hace el salto a la etiqueta guarda la posicion de la llamada en `$ra`, tiene la siguiente estructura:[1]

```
jal label # $ra ← PC + 4; go to label
```

– Salto con registro (`jr`): Instrucción de tipo J, se usa para volver de una llamada a una función/subrutina, tiene la siguiente estructura:

```
jr $ra # go to $ra
```

[1] PC significa "Program Counter" y es un registro que contiene la dirección de la instrucción que está siendo ejecutada en el programa