# Guía MIPS32: Registros, Instrucciones

2 de junio de 2025

## 1. Registros MIPS

| Nombre | Número | ¿se mantiene? | Explicación |
|---|---|---|---|
| $zero | 0 | Sí (siempre 0) | Siempre contiene el valor 0. No se puede modificar. |
| $at | 1 | No | Registro temporal reservado para el ensamblador (pseudoinstrucciones) |
| $v0-$v1 | 2-3 | No | Valores de retorno de funciones (return values) |
| $a0-$a3 | 4-7 | No | Argumentos para llamadas a funciones (arguments) |
| $t0-$t7 | 8-15 | No | Registros temporales (no preservados entre llamadas) |
| $s0-$s7 | 16-23 | Sí | Registros guardados (preservados entre llamadas) |
| $k0-$k1 | 26-27 | No | Reservados para el núcleo del sistema operativo |
| $gp | 28 | Sí | Puntero global (global pointer) |
| $sp | 29 | Sí | Puntero de pila (stack pointer) |
| $fp | 30 | Sí | Puntero de marco (frame pointer) |
| $ra | 31 | Sí | Dirección de retorno (return address) |

- **Número**: Cada registro tiene un identificador numérico (0-31) que se usa en la codificación binaria de las instrucciones. Por ejemplo, $t0 es el registro 8.
- **¿Se mantiene?**: Indica si el registro debe preservar su valor después de una llamada a función (convención de llamadas):
  - **Sí**: La función llamada debe dejar el registro con el mismo valor que tenía al entrar (usualmente se guarda en la pila).
  - **No**: La función llamada puede modificar el registro sin restricciones.

# 2. Instrucciones Aritméticas

## 2.1. Aritmética Básica

```
add $t0, $t1, $t2    # $t0 = $t1 + $t2 (suma con signo)
addu $t0, $t1, $t2   # $t0 = $t1 + $t2 (suma sin signo)
sub $t0, $t1, $t2    # $t0 = $t1 - $t2 (resta con signo)
subu $t0, $t1, $t2   # $t0 = $t1 - $t2 (resta sin signo)
addi $t0, $t1, 5     # $t0 = $t1 + 5 (suma inmediata)
```

## 2.2. Multiplicación y División

```
mult $t0, $t1        # Hi/Lo = $t0 * $t1 (producto de 32 bits)
div $t0, $t1         # Lo = $t0/$t1, Hi = $t0 % $t1 (division con
    signo)
mflo $t2             # $t2 = Lo (copia registro bajo)
mfhi $t3             # $t3 = Hi (copia registro alto)
```

# 3. Instrucciones de Comparación y Saltos

## 3.1. Comparaciones

```
slt $t0, $t1, $t2    # $t0 = 1 si $t1 < $t2 (con signo), 0 en otro
    caso
sltu $t0, $t1, $t2   # $t0 = 1 si $t1 < $t2 (sin signo), 0 en otro
    caso
slti $t0, $t1, 10    # $t0 = 1 si $t1 < 10 (inmediato con signo)
```

## 3.2. Saltos Condicionales

```
beq $t0, $t1, etiqueta  # Salta si $t0 == $t1
bne $t0, $t1, etiqueta  # Salta si $t0 != $t1
beqz $t0, etiqueta      # Salta si $t0 == 0 (pseudoinstrucci n)
bgt $t0, $t1, etiqueta  # Salta si $t0 > $t1 (pseudoinstrucci n)
blt $t0, $t1, etiqueta  # Salta si $t0 < $t1 (pseudoinstrucci n)
bge $t0, $t1, etiqueta  # Salta si $t0 >= $t1 (pseudoinstrucci n)
ble $t0, $t1, etiqueta  # Salta si $t0 <= $t1 (pseudoinstrucci n)
```

## 3.3. Saltos Incondicionales

```
j etiqueta           # Salto incondicional
jal etiqueta         # Salto y enlace (para llamadas a funci n)
jr $ra               # Salto a direcci n en $ra (retorno de
    funci n)
```

# 4. Manejo de Memoria

## 4.1. Concepto de Palabra

En MIPS, una **palabra** (word) son 4 bytes (32 bits). Las direcciones de memoria deben estar alineadas a palabras (múltiplos de 4).

## 4.2. Carga y Almacenamiento

```
lw $t0, 0($sp)      # Carga palabra de ($sp + 0) en $t0
sw $t0, 4($sp)      # Almacena $t0 en ($sp + 4)
lb $t0, 0($sp)      # Carga byte (sign-extend) en $t0
lbu $t0, 0($sp)     # Carga byte (zero-extend) en $t0
sb $t0, 0($sp)      # Almacena byte (los 8 bits bajos de $t0)
```

# 5. Entrada/Salida Básica

## 5.1. Lectura y Escritura de Enteros

```
# Ejemplo: Leer un entero y mostrarlo
.data
    mensaje: .asciiz "Ingrese un numero: "
    resultado: .asciiz "El numero ingresado es: "

.text
main:
    # Mostrar mensaje
    li $v0, 4           # C digo para imprimir string
    la $a0, mensaje      # Carga direcci n del mensaje
    syscall

    # Leer entero
    li $v0, 5           # C digo para leer entero
    syscall
    move $t0, $v0       # Guardar el valor le do en $t0

    # Mostrar resultado
    li $v0, 4
    la $a0, resultado
    syscall

    li $v0, 1           # C digo para imprimir entero
    move $a0, $t0       # Cargar el valor a imprimir
    syscall

    # Terminar programa
    li $v0, 10
    syscall
```

## 5.2. Manejo de Caracteres

```
.data
    char: .byte 'A'     # Caracter ASCII
    buffer: .space 20   # Buffer para cadena

.text
main:
    # Leer un car cter
    li $v0, 12          # Codigo para leer caracter
    syscall
    sb $v0, char        # Almacenar caracter leido

    # Mostrar car cter
    li $v0, 11          # Codigo para imprimir caracter
    lb $a0, char        # Cargar caracter
    syscall

    # Leer cadena
    li $v0, 8           # Codigo para leer cadena
    la $a0, buffer      # Direccion del buffer
    li $a1, 20          # Longitud maxima
    syscall

    # Mostrar cadena
    li $v0, 4
    la $a0, buffer
    syscall
```

## 5.3. Números Decimales (Punto Flotante)

```
.data
    flotante: .float 3.1416
    doble: .double 2.71828
    mensaje: .asciiz "El valor es: "

.text
main:
    # Leer float
    li $v0, 6           # Codigo para leer float
    syscall             # Resultado en $f0

    # Mostrar float
    li $v0, 4
    la $a0, mensaje
    syscall

    li $v0, 2           # Codigo para imprimir float
    mov.s $f12, $f0     # Cargar valor a imprimir
    syscall

    # Leer double
    li $v0, 7           # Codigo para leer double
    syscall             # Resultado en $f0-$f1

```

```
25      # Mostrar double
26      li $v0, 4
27      la $a0, mensaje
28      syscall
29
30      li $v0, 3           # Codigo para imprimir double
31      mov.d $f12, $f0     # Cargar valor a imprimir
32      syscall
```