



# TEMA 4: Lenguaje de Ordenador

## Conjunto de Instrucciones MIPS

→ Conjunto Instrucciones → MIPS  
uno de ellos lo veremos como ejemplos

Suma Resta  
add a, b, c → a = b+c      sub a, b, c → a = b-c

Principio 1 del diseño: la sencillez favorece la regularidad

### Registro de operandos

- MIPS → 32 x 32 bits archivo de registros
- 4 Byte → "Word" o "Palabra"

nombres en el ensamblador:

\$t0, \$t1, ..., \$t9 → valores temporales

\$s0, \$s1, ..., \$s7 → variables guardadas

Principio 2 de diseño: ↑ menor = ↑ más rápido

Ejemplo:

Lenguaje alto nivel:  $g = (h + i) - (j + k)$  → guardadas

Compilado Código MIPS:  $t_0 = t_1 \rightarrow$  temporales

add \$t0, \$s1, \$s2  
add \$t1, \$s3, \$s4  
sub \$s0, \$t0, \$t1

### Registros vs Memoria

- Registros → acceso más rápido
- Operaciones en Memoria → load (larga) y store (guardar)
- Compiladores → deben usar registros para variables lo máx. posible

### Operandos de Memoria

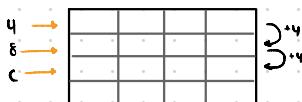
- Cargar valores memoria → registros: lw
- Guardar resultados registros → memoria: sw
- MIPS → Big Endian



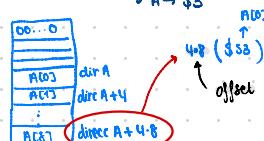
Dirección  $\square_{32}$  = dirección  $\square_8 \dots \square_0$

Como sabemos  $\neq?$  → depende instrucción

Para mayor comodidad → HEXADECIMAL



Ejemplo:  
 $g = h + A[8]$        $\left\{ \begin{array}{l} g \rightarrow \text{esta en donde } \$t1 \\ h \rightarrow \$t2 \\ A \rightarrow \$t3 \end{array} \right.$



lw \$t0, 32(\$s3) # load word  
add \$t0, \$s2, \$t0  
sw \$t0, 48(\$s3) # store word

### Operandos Inmediatos

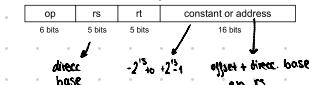
Principio 3 de Diseño: flat el caso común rápido

La constante cero → registro MIPS 0 (\$zero)

↳ útil para ≠ operaciones  
y: cambiar entre registros

add \$t2, \$s1, \$zero

### Instrucc. MIPS formato I



Principio 4 de diseño: Un buen diseño exige un buen compromiso

Ejemplo:

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32bit	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34bit	n.a.
add Immediate	I	\$imm	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	32bit	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	48bit	reg	reg	n.a.	n.a.	n.a.	address

lw \$t0, 32(\$s3)

¿\$s3 → \$s0 - \$s1: 16-23 nro \$s3 - 10)  
rd → \$t0 → registro \$ t0 - \$t1 → reg 8-15

35	19	8	32
op	rs	rd	cda. or address
100011	10011	0100	0...000000

### Instrucciones formato R

op	rs	rt	rd	shamt	6 bits
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Operation code	first register number	Second register number	Registro Destino	Shift Amount	function code
					cantidad de comités??

Ejemplo:

add \$t0, \$s1, \$s2

special	\$s1	\$s2	\$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

000000100110010010000000100000<sub>2</sub> = 02324020<sub>16</sub>

op	rs	rt	rd	shamt	funcf
0	8	9	10	0	34
↓	↓	↓	↓	↓	↓
\$t0	\$t1	\$t2			

desplazar (+8 - 10)

↓ sub (-) ↓

# TEMA 4: Lenguaje de Ordenador

## Conjunto de Instrucciones MIPS

Ejercicio: compilar el siguiente código

$$A[300] = h + A[300]$$

↓  
\$s2  
4200

lw \$t0, 1200(\$t1) → cargar elem memoria

add \$t0, \$s2, \$t0 → ~~NOT~~ notarlo!! → \$t0

sw \$t0, 1200(\$t1) → guardar elem

### operaciones NOT

$$a \text{ NOR } b = \text{NOT } (a \text{ OR } b)$$

↳ si queremos hacer un NOT:  $a \text{ NOR } \underline{\underline{0}}$

Nota → nor \$t0, \$t1, \$zero

$$\begin{array}{l} \text{St1: } 1100\ 0110 \\ \text{St0: } 0011\ 1001 \end{array}$$

### C code:

```
// set N to the smallest odd no less than N
if ( N%2 == 0 ) N++;
```

Si el  $\text{St0} \% 2 \rightarrow \text{res} \neq 0$   
ES PAR  
Terminar en 0

..... 1 → no hace nada  
..... 0 → +1

$$\begin{matrix} A1 = 1 & \text{t OR} \\ A1 = 0 & \end{matrix}$$

### Code:

lw	\$t0, 4(\$gp)	# fetch N
ori	\$t0, \$t0, 1	# turn on low order bit
sw	\$t0, 4(\$gp)	# store result in N

OR no +1

### Operaciones con Shift (desplazamiento)

Multiplicar un  $n^o \times 2 \rightarrow$  desplaza izq (SLL)

Dividir un  $n^o \div 2 \rightarrow$  desplaza dcha (SRL)

Ejemplo:

$$\$s0 \rightarrow 0000\ 1001 = 9 \Rightarrow 9 \cdot 2^4 = 144$$

$$\text{SLL de } 4 \rightarrow 1001\ 000 = 144$$

$$\text{sll } \$t2, \$s0, 4$$

0	0	16	10	4	0
op	rs	rt	rd	shamt	funct

### División con Right Shift

$$\text{Desplazo } 4 \rightarrow 2^4$$

no negativo: se mantiene el signo

$$-5 \div 4 \rightarrow \text{Desplazar } 2$$

$$11111011_2 \gg 2 = 1111110_2 = -2$$

↑↑

en  $n^o$  sin signo: se añaden ceros

$$11111011_2 \ggg 2 = 00111110_2 = +62$$

### C code:

$$\begin{array}{l} A[i] = A[i/2] + 1; \\ A[i+1] = -1; \end{array} \rightarrow \text{direcc A } \gg 28$$

```
# A[i] = A[i/2] + 1;
lw    $t0, 0($gp)      # fetch i
srl   $t0, $t0, 1       # i/2
addi  $t1, $gp, 28      # &A[0]
sll   $t0, $t0, 2       # turn i/2 into a byte offset (*4)
add   $t1, $t1, $t0      # &A[i/2]
lw    $t1, 0($t1)        # fetch A[i/2]
addi  $t1, $t1, 1       # A[i/2] + 1
lw    $t0, 0($gp)        # fetch i
sll   $t0, $t0, 2       # turn i into a byte offset
addi  $t2, $gp, 28      # &A[0]
add   $t2, $t2, $t0      # &A[i]
sw    $t1, 0($t2)        # A[i] = ...
# A[i+1] = -1;
lw    $t0, 0($gp)        # fetch i
addi  $t0, $t0, 1       # i+1
sll   $t0, $t0, 2       # turn i+1 into a byte offset
addi  $t1, $gp, 28      # &A[0]
add   $t1, $t1, $t0      # &A[i+1]
addi  $t2, $zero, -1     # -1
sw    $t2, 0($t1)        # A[i+1] = -1
```

### Operaciones AND y OR

and \$t0, \$t1, \$t2	or \$t0, \$t1, \$t2	ori \$t0, \$t1, 1
\$t2 = 1100 0011	\$t2 = 1100 0011	ori = \$t0, \$t1, 1,
\$t1 = 0100 1101	\$t1 = 0100 1101	andi \$t0, \$t1, 1,
\$t0 = 0100 0001	\$t0 = 1100 1111	=



# TEMA 4: Lenguaje de Ordenador

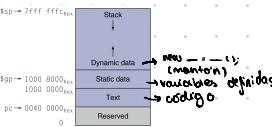
## Datos Locales en la Pila

- Datos locales asignados por calle

Algunos compiladores para controlar la pila

Utilizar Marco de Procedimiento  
(registros de activación)

## Cara de Memoria



## Resumen Registros

Name	Register number	Usage	Preserved on call?
\$zero	0	The constant value 0	n.a.
\$v0-\$v1	2-3	Values for results and expression evaluation	no
\$a0-\$a3	4-7	Arguments	no
\$t0-\$t7	8-15	Temporaries	no
\$s0-\$s7	16-23	Saved	yes
\$t8-\$t9	24-25	More temporaries	no
\$gp	28	Global pointer	yes
\$sp	29	Stack pointer	yes
\$fp	30	Frame pointer	yes
\$ra	31	Return address	yes

ASCII: 128 caracteres

Datos de Caracteres ↗ Latin-1: 256 caracteres

Unicode: conjunto de caracteres de 32 bits

Hay algunas operaciones q solo dejan añadir cte de 16 bits:



lui \$s0, 61

0000 0000 0011 1101 0000 0000 0000 0000

ori \$s0, \$s0, 2304

0000 0000 0011 1101 0000 1001 0000 0000

## Saltos direccionalizados

Saltos  $\rightarrow j$  o  $jal$



$$\text{direcc destino} = \text{PC}_{21\ldots28} + (\text{address} \times 4)$$

$$(\text{PC} = 0x0000 \& 0000 \rightarrow \text{Hexadecimal, cada n}\circ \rightarrow 4 \text{ bits})$$

J 20 → ¿Dónde va a saltar?  $20 \cdot 4 = 80 \rightarrow 1010000_2$

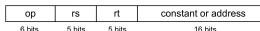
• Long bits del 21 al 28 0000

$1010000_2$

• 80 en binario y rellenar con ceros  $00\ldots00100000_2$  28 bits !!

2103

## Direcciónamiento de formas



$$\text{direcc destino} = \text{CP} + \text{offset} \times 4$$

Ejemplo: 1000 está en la direcc 80 000

Loop: \$t1 \$t1, \$s3, 2	80000	0 0 19 9 4 0
add \$t1, \$t1, \$s6	80004	0 9 22 9 0 32
lw \$t0, 0(\$t1)	80008	35 9 8 0 0
bne \$t0, \$s5, <b>Exit</b>	80012	5 8 21 2 0
addi \$s3, \$s3, 1	80016	8 19 19 1 1
j Loop	80020	2 20000
Exit: ...	80024	2 20000 X 4 = 80000

¿dónde está exit?

$2 \times 4 + \text{PC}$

PC = 80012

800 en 8 + 80012 = 80020

$20000 \times 4 = 80000$

Si el objetivo está demasiado lejos → reescribe el código

beq \$s0,\$s1, L1

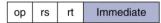
↓

bne \$s0,\$s1, L2

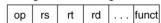
j L1

L2: ...

### 1. Immediate addressing



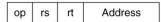
### 2. Register addressing



Registers

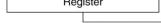
Register

### 3. Base addressing



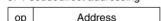
Memory

### 4. PC-relative addressing

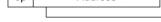


Memory

### 5. Pseudodirect addressing



Memory



Word

# TEMA 4: Lenguaje de Ordenador

## EJERCICIOS

- Considere una función denominada `SumarValor`. A esta función se le pasan tres parámetros: la dirección de inicio de un vector de números enteros, valor entero que indica el número de componentes del vector y el valor entero que hay que sumar.
- La función `SumarValor` modifica el vector, sumando el valor pasado como tercer parámetro a todas las componentes del vector. Se pide:
  - a) Indique en qué registros se han de pasar cada uno de los parámetros a la función.
  - b) Programar en ensamblador la función `SumarValor`.
  - c) Dado el siguiente fragmento de programa:

```
.data  
v: .word 7,8,3,4,5,6
```

```
.text  
.globl main  
main:
```

incluya en el `main` anterior, las sentencias en ensamblador necesarias para poder invocar a la función `SumarValor` implementada en el apartado b) de forma que sume a las componentes del vector `v` definido en la sección de datos, el número 5. Implemente a continuación de la llamada a la función, las sentencias en ensamblador que permitan imprimir todas las componentes del vector.

Añadir una función `Sumar` (`int a, int b`, que  $\rightarrow$  `a+b`)

```
SumaValor (int a[], int n, int val){  
    for (i=0 ; i < n ; i++) {  
        a[i] = a[i] + val;  
    }  
    Suma (int a, int b).  
    return a[0];
```

1º) Programamos más sencilla

Suma:  
add \$v0, \$a0, \$a1  
jr ra

2º) Programamos más compleja

`SumaValor:`

```
addi $$sp, $$sp, -12 # reservamos 3  
                     # espacios  
sw $ra, 0($sp)  
sw $a0, 4($sp)  
sw $a1, 8($sp)  
# inicializamos i  
li $t0, 0
```

bra \$t0, \$a1

se resuelven  
en suma

llamada  
a otra  
 $f(x)$

④ add \$t1, \$a0, \$zero      [registrar dato pq se  
 me pierde el usar otra f(x)]

⑤ add \$t3, \$a1, \$zero      [guardarlos en un  
 registro → más optimo]

loop:

slt \$t2, \$t1, \$a1      si  $i \leq n-1$   
bne \$t2, \$zero, EXIT      si  $t1=0 \rightarrow$  Exit

lw \$a0, 0(\$t1)      traemos \$a0

add \$a1, \$a2, \$zero      \$a2 → \$a1

jal SUMA      ] suma:  $a_1+a_2 = \$v0$   
sw \$v0, 0(\$t1)

add \$t1, \$t1, 4      ] direct+4 (= i+1)

addi \$t0, \$t0, 1

j LOOP

EXIT:

```
sw $ra, 0($sp)  
addi $sp, $sp, 12 # pop  
jr $ra
```