



# Tema 3 (II)

## Fundamentos de la programación en ensamblador



Grupo ARCOS

Estructura de Computadores  
Grado en Ingeniería Informática  
Universidad Carlos III de Madrid

# Contenidos

---

- I. Programación en ensamblador (II)
  1. Arquitectura MIPS (II)
  2. Directivas
  3. Servicios del sistema
  4. Tipo de instrucciones (II)

# ¡ATENCIÓN!

---

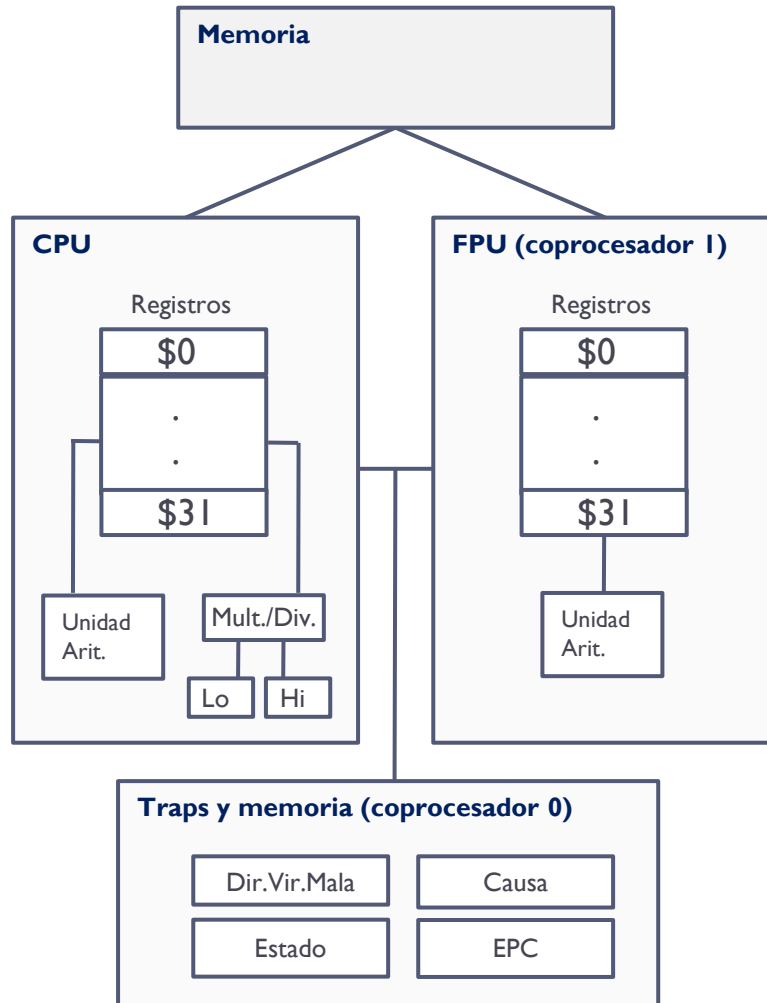
- ❑ Estas transparencias son un guión para la clase
- ❑ Los libros dados en la bibliografía junto con lo explicado en clase representa el material de estudio para el temario de la asignatura

# Contenidos

---

- I. Programación en ensamblador (II)**
  - 1. Arquitectura MIPS (II)**
  2. Directivas
  3. Servicios del sistema
  4. Tipo de instrucciones (II)

# Arquitectura MIPS



- ▶ **MIPS R2000/R3000**
  - ▶ Procesador de 32 bits
  - ▶ Tipo RISC
  - ▶ CPU + coprocesadores auxiliares
- ▶ **Coprocesador 0**
  - ▶ excepciones, interrupciones y sistema de memoria virtual
- ▶ **Coprocesador 1**
  - ▶ FPU (Unidad de Punto Flotante)

# SPIM

The screenshot shows the PCSpim simulator window. At the top, there's a menu bar (File, Simulator, Window, Help) and a toolbar. Below that, a status bar displays PC, Status, EPC, HI, Cause, LO, and BadVAddr. The main area is divided into three sections: General Registers, Code, and Memory. The General Registers section shows R0 through R31 with their names and values. The Code section shows assembly instructions with addresses and comments. The Memory section shows DATA, STACK, and KERNEL DATA. At the bottom, there's a status bar with the current PC, EPC, and Cause values.

```
PC = 00400000 EPC = 00000000 Cause = 00000000 BadVAddr= 00000000
Status = 3000ff10 HI = 00000000 LO = 00000000

General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
R5 (a1) = 00000000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7ffffffc
R6 (a2) = 00000000 R14 (t6) = 00000000 R22 (s6) = 00000000 R30 (s8) = 00000000
R7 (a3) = 00000000 R15 (t7) = 00000000 R23 (s7) = 00000000 R31 (ra) = 00000000

[0x00400000] 0x8fa40000 lw $4, 0($29) ; 175: lw $a0 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 176: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 177: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 178: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 179: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 180: jal main
[0x00400018] 0x00000000 nop ; 181: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 183: li $v0 10
[0x00400020] 0x0000000c syscall ; 184: syscall # syscall 10 (exit)
[0x00400024] 0x34020004 ori $2, $0, 4 ; 8: li $2 4
[0x00400028] 0x3c041001 lui $4, 4097 [msg_hola] ; 9: la $4 msg_hola

DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x616c6f68 0x6e756d20 0x000a6f64 0x00000000
[0x10010010]...[0x10040000] 0x00000000

STACK
[0x7ffffffc] 0x00000000

KERNEL DATA
[0x90000000] 0x78452020 0x74706563 0x206e6f69 0x636f2000
[0x90000010] 0x72727563 0x61206465 0x6920646e 0x726f6e67

Memory and registers cleared and the simulator reinitialized.

SPIM Version Version 7.3 of August 26, 2006
Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Loaded: C:\Program Files\PCSpim\Exceptions.s
Z:\work\docencia\grado\info-ec\2008-2009\transparencias\hola.s.txt successfully loaded

For Help, press F1 PC=0x00400000 EPC=0x00000000 Cause=0x00000000
```

Banco de registros

\$0, \$1, \$2, ...

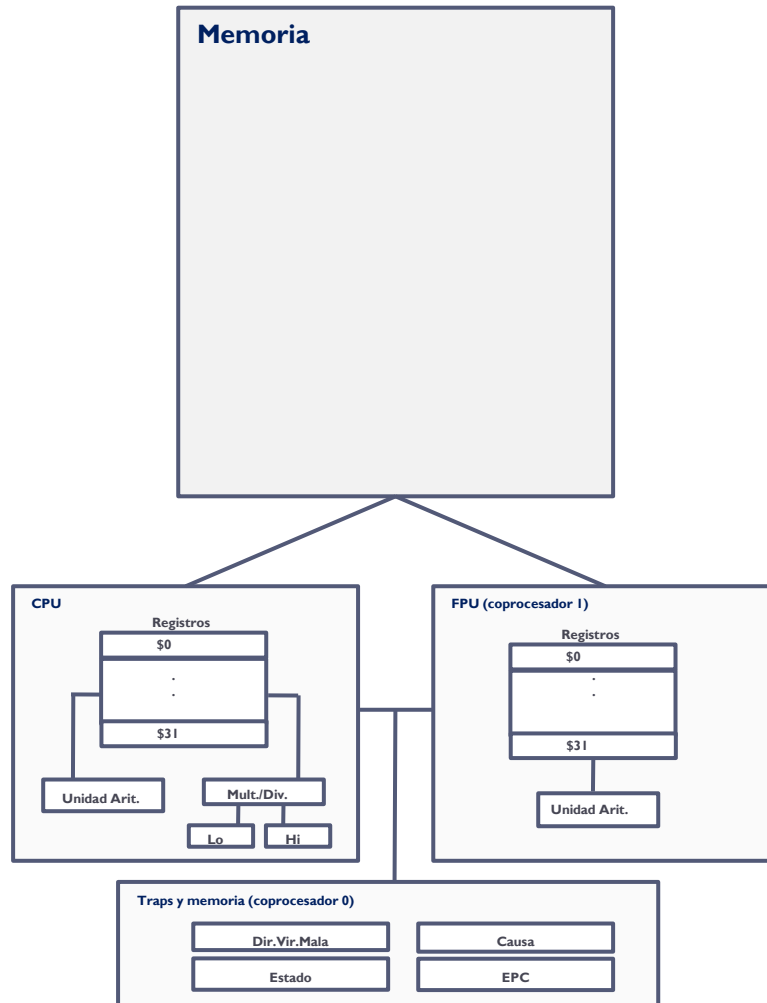
\$f0, \$f1, ...

# Banco de registros (enteros)

Nombre registro	Número	Uso
zero	0	Constante 0
at	1	Reservado para el ensamblador
v0, v1	2, 3	Resultado de una rutina (o expresión)
a0, ..., a3	4, ..., 7	Argumento de entrada para rutinas
t0, ..., t7	8, ..., 15	Temporal ( <u>NO</u> se conserva entre llamadas)
s0, ..., s7	16, ..., 23	Temporal (se conserva entre llamadas)
t8, t9	24, 25	Temporal ( <u>NO</u> se conserva entre llamadas)
k0, k1	26, 27	Reservado para el sistema operativo
gp	28	Puntero al área global
sp	29	Puntero a pila
fp	30	Puntero a marco de pila
ra	31	Dirección de retorno (rutinas)

- ▶ Hay 32 registros
  - ▶ 4 bytes de tamaño (una palabra)
  - ▶ Se nombran con un \$ al principio
- ▶ Convenio de uso
  - ▶ Reservados
  - ▶ Argumentos
  - ▶ Resultados
  - ▶ Temporales
  - ▶ Punteros

# Arquitectura MIPS



## ▶ Memoria

- ▶ 4 GB ( $2^{32}$ )
- ▶ Direccionada por bytes

## ▶ MIPS R2000/R3000

- ▶ Procesador de 32 bits
- ▶ Tipo RISC
- ▶ CPU + coprocesadores auxiliares



# SPIM

The screenshot shows the PCSpim simulator window. At the top, there is a menu bar (File, Simulator, Window, Help) and a toolbar. Below that, the status bar displays: PC = 00400000, EPC = 00000000, Cause = 00000000, BadVAddr = 00000000, Status = 3000ff10, HI = 00000000, LO = 00000000.

The main window is divided into several sections:

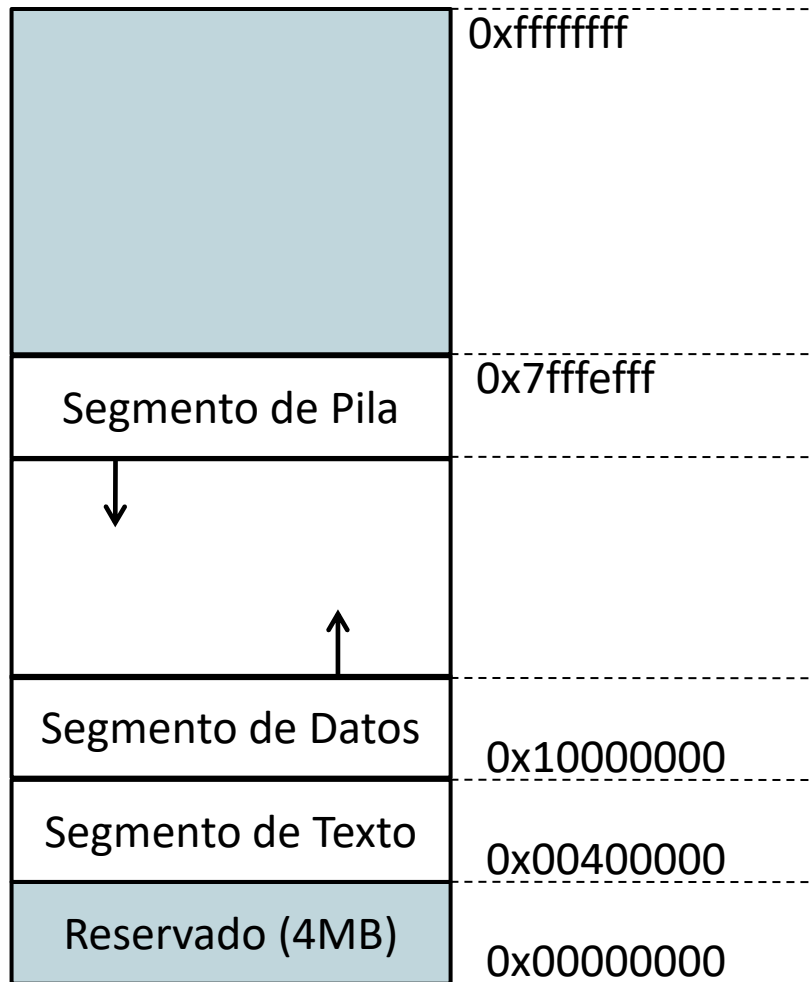
- General Registers:** A table showing the values of registers R0 through R31. For example, R0 (r0) = 00000000, R1 (at) = 00000000, R2 (v0) = 00000000, R3 (v1) = 00000000, R4 (a0) = 00000000, R5 (a1) = 00000000, R6 (a2) = 00000000, R7 (a3) = 00000000, R8 (t0) = 00000000, R9 (t1) = 00000000, R10 (t2) = 00000000, R11 (t3) = 00000000, R12 (t4) = 00000000, R13 (t5) = 00000000, R14 (t6) = 00000000, R15 (t7) = 00000000, R16 (s0) = 00000000, R17 (s1) = 00000000, R18 (s2) = 00000000, R19 (s3) = 00000000, R20 (s4) = 00000000, R21 (s5) = 00000000, R22 (s6) = 00000000, R23 (s7) = 00000000, R24 (t8) = 00000000, R25 (t9) = 00000000, R26 (k0) = 00000000, R27 (k1) = 00000000, R28 (gp) = 10008000, R29 (sp) = 7ffffffc, R30 (s8) = 00000000, R31 (ra) = 00000000.
- Assembly Code:** A list of instructions with their addresses and comments. The current instruction at address 0x00400000 is highlighted in blue: `lw $4, 0($29) ; 175: lw $a0 0($sp) # argc`. Other instructions include `addiu $5, $29, 4 ; 176: addiu $a1 $sp 4 # argv`, `addiu $6, $5, 4 ; 177: addiu $a2 $a1 4 # envp`, `sll $2, $4, 2 ; 178: sll $v0 $a0 2`, `addu $6, $6, $2 ; 179: addu $a2 $a2 $v0`, `jal 0x00400024 [main] ; 180: jal main`, `nop ; 181: nop`, `ori $2, $0, 10 ; 183: li $v0 10`, `syscall ; 184: syscall # syscall 10 (exit)`, `ori $2, $0, 4 ; 8: li $2 4`, and `lui $4, 4097 [msg_hola] ; 9: la $4 msg_hola`.
- DATA:** Memory addresses and their contents. For example, `[0x10000000]...[0x10010000] 0x00000000`, `[0x10010000] 0x616c6f68 0x6e756d20 0x000a6f64 0x00000000`, and `[0x10010010]...[0x10040000] 0x00000000`.
- STACK:** Memory address `[0x7ffffffc] 0x00000000`.
- KERNEL DATA:** Memory addresses and their contents. For example, `[0x90000000] 0x78452020 0x74706563 0x206e6f69 0x636f2000` and `[0x90000010] 0x72727563 0x61206465 0x6920646e 0x726f6e67`.

At the bottom of the window, there is a status bar with the text: "Memory and registers cleared and the simulator reinitialized." and "SPIM Version Version 7.3 of August 26, 2006 Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu). All Rights Reserved. DOS and Windows ports by David A. Carley (dac@cs.wisc.edu). Copyright 1997 by Morgan Kaufmann Publishers, Inc. See the file README for a full copyright notice. Loaded: C:\Program Files\PCSpim\Exceptions.s Z:\work\docencia\grado\info-ec\2008-2009\transparencias\hola.s.txt successfully loaded".

Memoria

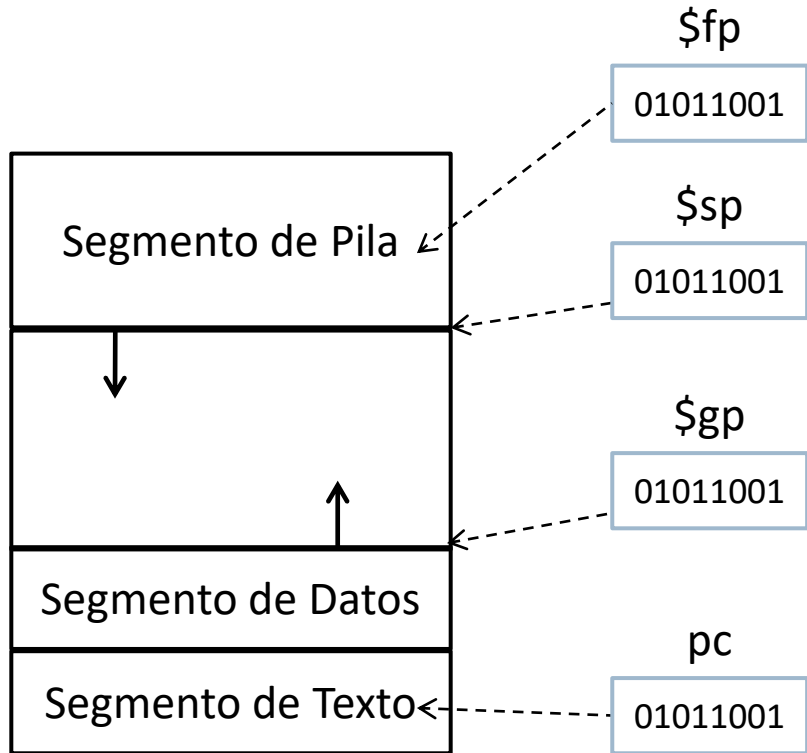


# Memoria



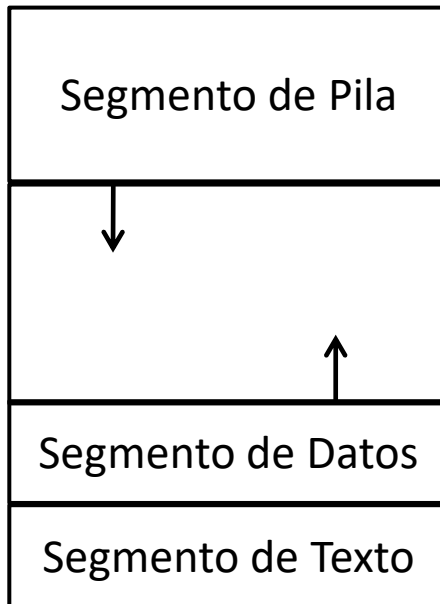
- ▶ Hay 4 GB de memoria direccionables en total
- ▶ Parte de esa memoria la utilizan los distintos segmentos de un proceso
- ▶ Otra parte de la memoria está reservada:
  - ▶ Un mini-sistema operativo reside en los primeros 4 MB de memoria

# Mapa de memoria de un proceso



- ▶ Los procesos dividen el espacio de memoria en segmentos lógicos para organizar el contenido:
  - ▶ Segmento de pila
    - ▶ Variables locales
    - ▶ Contexto de funciones
  - ▶ Segmento de datos
    - ▶ Datos estáticos
  - ▶ Segmento de código (texto)
    - ▶ Código

# Ejercicio

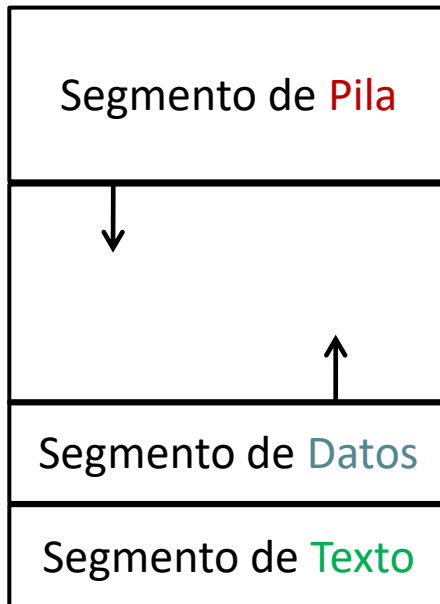


```
// variables globales
int a;

main ()
{
    // variables locales
    int b;

    // código
    return a + b;
}
```

# Ejercicio (solución)



```
// variables globales
int a;

main ()
{
    // variables locales
    int b;

    // código
    return a + b;
}
```

# SPIM

The screenshot shows the PCSpim simulator interface with the following sections:

- Registers:** A table of 32 general registers (R0-R31) with their names and values. For example, R0 (r0) = 00000000, R16 (s0) = 00000000, R24 (t8) = 00000000, etc.
- Code Segment:** A list of instructions with their addresses and comments. For example, [0x00400000] 0x8fa40000 lw \$4, 0(\$29) ; 175: lw \$a0 0(\$sp) # argc.
- Data Segment:** A list of memory addresses and their contents. For example, [0x10000000]...[0x10010000] 0x00000000.
- Stack:** A list of memory addresses and their contents. For example, [0x7ffffc] 0x00000000.
- Kernel Data:** A list of memory addresses and their contents. For example, [0x90000000] 0x78452020 0x74706563 0x206e6f69 0x636f2000.

At the bottom, there is a status bar with the text: "For Help, press F1" and "PC=0x00400000 EPC=0x00000000 Cause=0x00000000".

Segmento de código

Segmento de datos

Segmento de pila

Otros



# Contenidos

---

## I. Programación en ensamblador (II)

1. Arquitectura MIPS (II)
2. **Directivas**
3. Servicios del sistema
4. Tipo de instrucciones (II)

# Ejemplo: Hola mundo...

---

hola.s

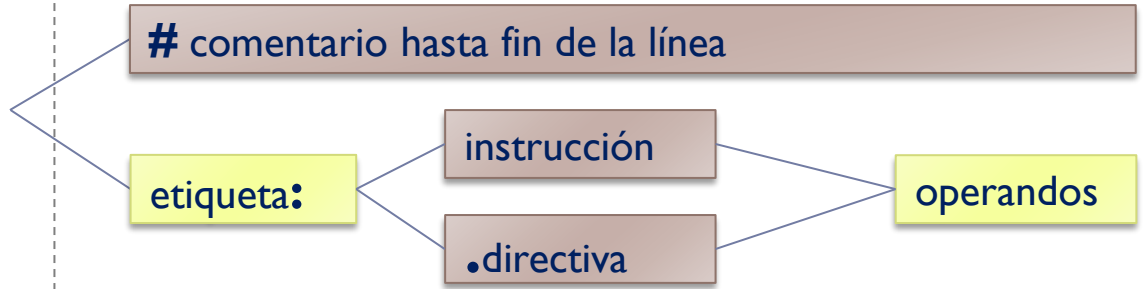
```
.data
    msg_hola: .asciiz "hola mundo\n"

.text
    .globl main
main:
    # printf("hola mundo\n") ;
    li $v0 4
    la $a0 msg_hola
    syscall
```



# Ejemplo: Hola mundo...

```
hola.s
.data
msg_hola: .asciiz "hola mundo\n"
.text
.globl main
main:
# printf("hola mundo\n");
li $v0 4
la $a0 msg_hola
syscall
```



# Ejemplo: Hola mundo...

---

hola.s

**.data**

```
msg_hola: .ascii "hola mundo\n"
```

**.text**

```
.globl main
```

```
main:
```

```
    # printf("hola mundo\n") ;
```

```
    li $v0 4
```

```
    la $a0 msg_hola
```

```
    syscall
```

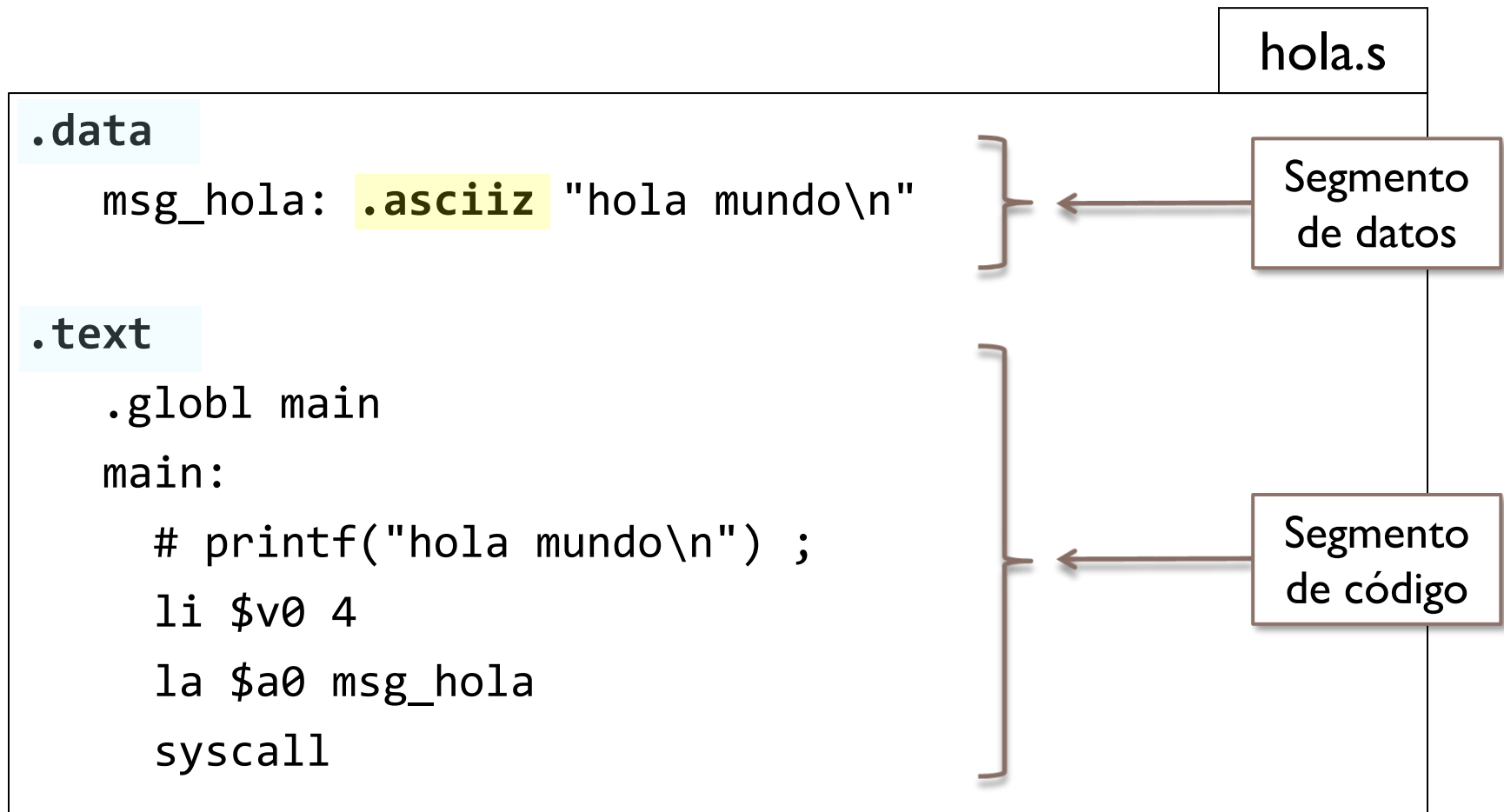


# Programa en ensamblador: directivas de preproceso

---

Directivas	Uso
<b>.data</b>	Siguientes elementos van al segmento de dato
<b>.text</b>	Siguientes elementos van al segmento de código
<b>.ascii</b> “ <i>tira de caracteres</i> ”	Almacena cadena caracteres NO terminada en carácter nulo
<b>.asciiz</b> “ <i>tira de caracteres</i> ”	Almacena cadena caracteres terminada en carácter nulo
<b>.byte</b> 1, 2, 3	Almacena bytes en memoria consecutivamente
<b>.half</b> 300, 301, 302	Almacena medias palabras en memoria consecutivamente
<b>.word</b> 800000, 800001	Almacena palabras en memoria consecutivamente
<b>.float</b> 1.23, 2.13	Almacena float en memoria consecutivamente
<b>.double</b> 3.0e21	Almacena double en memoria consecutivamente
<b>.space</b> 10	Reserva un espacio de 10 bytes en el segmento actual
<b>.extern</b> <i>etiqueta n</i>	Declara que <i>etiqueta</i> es global de tamaño <i>n</i>
<b>.globl</b> <i>etiqueta</i>	Declara <i>etiqueta</i> como global
<b>.align</b> <i>n</i>	Alinea el siguiente dato en un límite de $2^n$

# Ejemplo: Hola mundo...



# SPIM

The screenshot shows the SPIM simulator window with the following content:

```
PC = 00400000 EPC = 00000000 Cause = 00000000 BadVAddr= 00000000
Status = 3000fff1 HI = 00000000 LO = 00000000

General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
R5 (a1) = 00000000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7ffffffc
R6 (a2) = 00000000 R14 (t6) = 00000000 R22 (s6) = 00000000 R30 (s8) = 00000000
R7 (a3) = 00000000 R15 (t7) = 00000000 R23 (s7) = 00000000 R31 (ra) = 00000000

[0x00400000] 0x8fa40000 lw $4, 0($29) ; 175: lw $a0 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 176: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 177: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 178: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 179: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 180: jal main
[0x00400018] 0x00000000 nop ; 181: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 183: li $v0 10
[0x00400020] 0x0000000c syscall ; 184: syscall # syscall 10 (exit)
[0x00400024] 0x34020004 ori $2, $0, 4 ; 8: li $2 4
[0x00400028] 0x3c041001 lui $4, 4097 [msg_hola] ; 9: la $4 msg_hola

DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x616c6f68 0x6e756d20 0x000a6f64 0x00000000
[0x10010010]...[0x10040000] 0x00000000

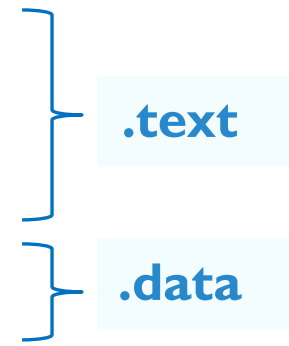
STACK
[0x7ffffffc] 0x00000000

KERNEL DATA
[0x90000000] 0x78452020 0x74706563 0x206e6f69 0x636f2000
[0x90000010] 0x72727563 0x61206465 0x6920646e 0x726f6e67

Memory and registers cleared and the simulator reinitialized.

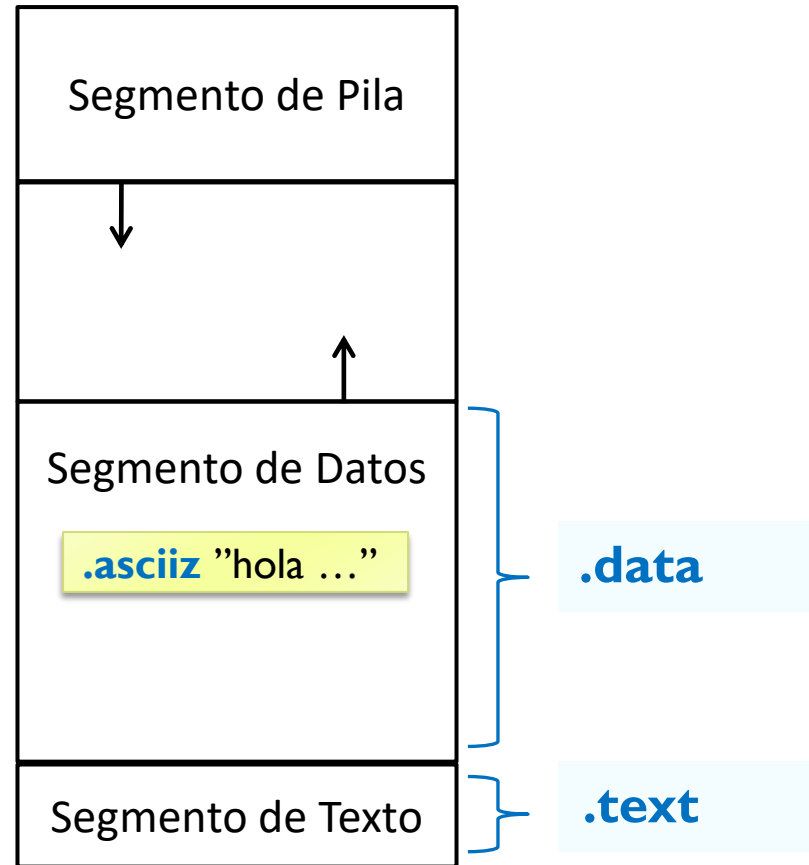
SPIM Version Version 7.3 of August 26, 2006
Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Loaded: C:\Program Files\PCSpim\Exceptions.s
Z:\work\docencia\grado\info-ec\2008-2009\transparencias\hola.s.txt successfully loaded

For Help, press F1 PC=0x00400000 EPC=0x00000000 Cause=0x00000000
```



# Programa en ensamblador: directivas de preproceso

```
hola.s
.data
msg_hola: .ascii "hola mundo\n"
.text
.globl main
main:
# printf("hola mundo\n");
li $v0 4
la $a0 msg_hola
syscall
```



# Programa en ensamblador: directivas de preproceso

---

Directivas	Uso
<code>.data</code>	Siguientes elementos van al segmento de dato
<code>.text</code>	Siguientes elementos van al segmento de código
<code>.ascii</code> “tira de caracteres”	Almacena cadena caracteres NO terminada en carácter nulo
<code>.asciiz</code> “tira de caracteres”	Almacena cadena caracteres terminada en carácter nulo
<code>.byte</code> 1, 2, 3	Almacena bytes en memoria consecutivamente
<code>.half</code> 300, 301, 302	Almacena medias palabras en memoria consecutivamente
<code>.word</code> 800000, 800001	Almacena palabras en memoria consecutivamente
<code>.float</code> 1.23, 2.13	Almacena float en memoria consecutivamente
<code>.double</code> 3.0e21	Almacena double en memoria consecutivamente
<code>.space</code> 10	Reserva un espacio de 10 bytes en el segmento actual
<code>.extern</code> etiqueta <i>n</i>	Declara que <i>etiqueta</i> es global de tamaño <i>n</i>
<code>.globl</code> etiqueta	Declara <i>etiqueta</i> como global
<code>.align</code> <i>n</i>	Alinea el siguiente dato en un límite de $2^n$

# Representación de tipo de datos básicos (1/3)

```
// boolean
bool_t b1 ;
bool_t b2 = false ;

// caracter
char c1 ;
char c2 = 'x' ;

// enteros
int res1 ;
int op1 = -10 ;

// coma flotante
float f0 ;
float f1 = 1.2 ;
double d2 = 3.0e10 ;
```

```
.data
# boolean
b1:  .space 1
b2:  .byte 0          # 1 byte

# caracter
c1:  .byte
c2:  .byte 'x'       # 1 bytes

# enteros
res1: .space 4
op1:  .word -10      # 4 bytes

# coma flotante
f0:  .float
f1:  .float 1.2      # 4 bytes
d2:  .double 3.0e10 # 8 bytes
```



# Representación de tipo de datos básicos (2/3)

```
// vectores
int vec[5] ;
int mat[2][3] = {{11,12,13},
                 {21,22,23}};
```

**.data**

```
# vectores
vec:  .space 20      # 5 elem.*4 bytes
mat:  .word 11, 12, 13
      .word 21, 22, 23
```

	...	
mat:	11	0x0108
	12	0x010c
	13	0x0110
	21	0x0114
	22	0x0118
	23	0x011c
	...	

# Representación de tipo de datos básicos (3/3)

```
// tira de caracteres (strings)
char c1[10] ;
char ac1[] = "hola" ;
```

**.data**

```
# strings
c1:   .space 10      # 10 byte
ac1:  .asciiz "hola" # 5 bytes (!)
ac2:  .ascii  "hola" # 4 bytes
```

	...	
ac1:	'h'	0x0108
	'o'	0x0109
	'l'	0x010a
	'a'	0x010b
	0	0x010c
	...	0x010d

	...	
ac2:	'h'	0x0108
	'o'	0x0109
	'l'	0x010a
	'a'	0x010b
	...	0x010c
	...	0x010d

# Ejercicio

---

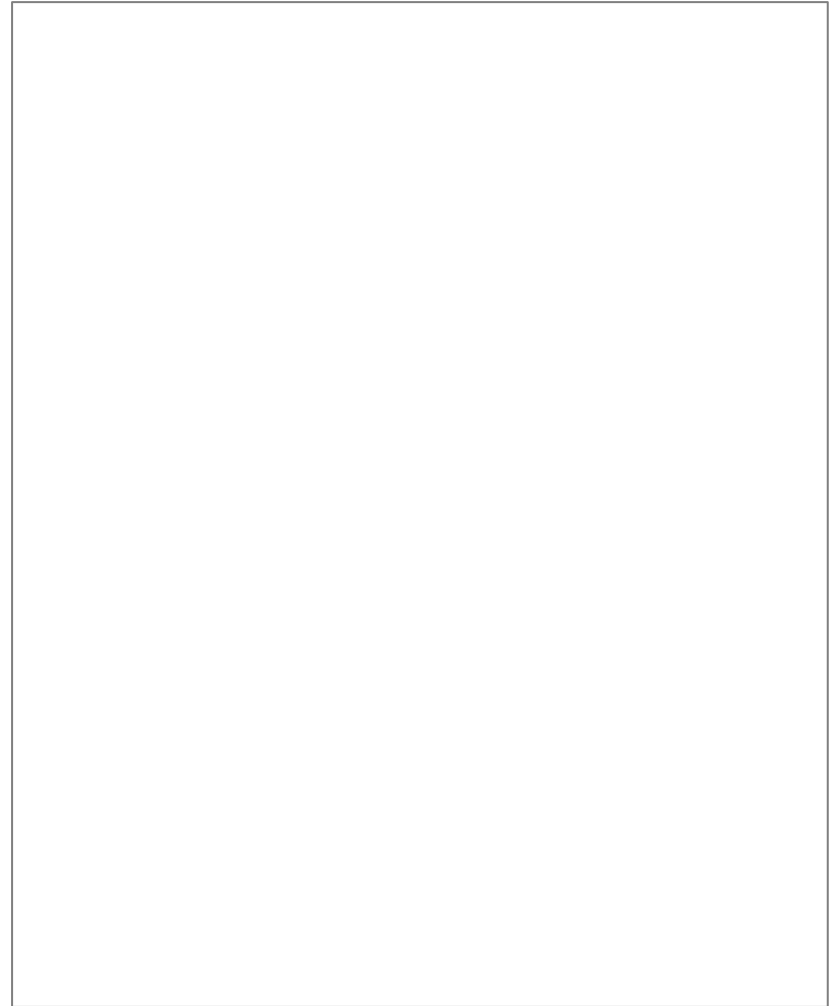


```
// variables globales

char v1;
int v2 ;
float v3 = 3.14 ;

char v4[10] ;
char v5 = "ec" ;

int v6[] = { 20, 22 } ;
```



# Ejercicio (solución)

---



```
// variables globales
```

```
char v1;
```

```
int v2 ;
```

```
float v3 = 3.14 ;
```

```
char v4[10] ;
```

```
char v5 = "ec" ;
```

```
int v6[] = { 20, 22 } ;
```

```
.data
```

```
v1: .byte
```

```
v2: .word
```

```
v3: .float 3.14
```

```
v4: .space 10
```

```
v5: .ascii "ec"
```

```
v6: .word 20, 22
```

# Ejercicio



v1:	?	0x0100
	?	0x0101
	?	0x0102
	?	0x0103
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	?	
	...	

```
.data  
  
v1: .byte  
v2: .word  
v3: .float 3.14  
  
v4: .space 10  
v5: .ascii "ec"  
  
v6: .word 20, 22
```

# Ejercicio (solución)



v1:	?	0x0100
v2:	?	0x0101
	?	0x0102
	?	0x0103
	?	0x0104
v3:	(3.14)	0x0105
	(3.14)	0x0106
	(3.14)	0x0107
	(3.14)	0x0108
v4:	?	0x0109
	...	...
	?	0x0102
v5:	'e'	0x0113
	'c'	0x0114
	0	0x0115
v6:	(20)	0x0116
	...	0x0117
	(22)	0x0120
	...	0x0121

```
.data  
  
v1: .byte  
v2: .word  
v3: .float 3.14  
  
v4: .space 10  
v5: .ascii "ec"  
  
v6: .word 20, 22
```

# Contenidos

---

## I. Programación en ensamblador (II)

1. Arquitectura MIPS (II)
2. Directivas
3. **Servicios del sistema**
4. Tipo de instrucciones (II)

# Ejemplo: Hola mundo...

hola.s

```
.data
    msg_hola: .asciiz "hola mundo\n"

.text
.globl main
main:
    # printf("hola mundo\n") ;
    li $2, 4
    la $4, msg_hola
    syscall
```





# Programa en ensamblador: llamadas al sistema

---

<b>Servicio</b>	<b>Código de llamada</b>	<b>Argumentos</b>	<b>Resultado</b>
print_int	1	\$a0 = integer	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer en \$v0
read_float	6		float en \$f0
read_double	7		double en \$f0
read_string	8	\$a0=buffer, \$a1=longitud	
sbrk	9	\$a0=cantidad	dirección en \$v0
exit	10		

# Ejemplo: Hola mundo...

---

hola.s

```
.data
    msg_hola: .asciiz "hola mundo\n"

.text
.globl main
main:
    # printf("hola mundo\n") ;
    li $v0 4
    la $a0 msg_hola
    syscall
```



# SPIM

The screenshot shows the PCSpim MIPS simulator interface. The main window displays assembly code with comments and register values. The registers are listed as follows:

Register	Value
PC	00400030
Status	3000ff10
EPC	00000000
HI	00000000
Cause	00000000
LO	00000000
BadVAddr	00000000

The assembly code is as follows:

```
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 176: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 177: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 178: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 179: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 180: jal main
[0x00400018] 0x00000000 nop ; 181: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 183: li $v0 10
[0x00400020] 0x0000000c syscall ; 184: syscall # syscall 10 (exit)
[0x00400024] 0x34020004 ori $2, $0, 4 ; 8: li $2 4
[0x00400028] 0x3c041001 lui $4, 4097 [msg_hola] ; 9: la $4 msg_hola
[0x0040002c] 0x0000000c syscall ; 10: syscall
```

The DATA section is as follows:

```
DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x616c6f68 0x6e756d20 0x000a6f64 0x00000000
[0x10010010]...[0x10040000] 0x00000000
```

The STACK section is as follows:

```
STACK
[0x7ffffeffc] 0x00000000
```

The KERNEL DATA section is as follows:

```
KERNEL DATA
[0x90000000] 0x78452020 0x74706563 0x206e6f69 0x636f2000
[0x90000010] 0x72727563 0x61206465 0x6920646e 0x726f6e67
```

The console window shows the output: "hola mundo".

# Ejercicio



```
// variables globales
int valor ;
```

```
main ()
{
    readInt(&valor) ;
    valor = valor + 1 ;
    printInt(valor) ;
}
```

Servicio	Código de llamada	Argumentos	Resultado
<b>print_int</b>	<b>1</b>	<b>\$a0 = integer</b>	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
<b>read_int</b>	<b>5</b>		<b>integer en \$v0</b>
read_float	6		float en \$f0
read_double	7		double en \$f0
read_string	8	\$a0=buffer, \$a1=long.	
sbrk	9	\$a0=cantidad	dirección en \$v0
exit	10		

# Ejercicio (solución)



```
// variables globales
```

```
int valor ;
```

```
main ()
```

```
{
```

```
    readInt(&valor) ;
```

```
    valor = valor + 1 ;
```

```
    printInt(valor) ;
```

```
}
```

```
.data
```

```
    valor: .word
```

```
.text
```

```
.globl main
```

```
main:
```

```
    # readInt(&valor)
```

```
    li $v0 5
```

```
    syscall
```

```
    sw $v0 valor
```

```
    # valor = valor + 1
```

```
    lw  $v0 valor
```

```
    add $v0 $v0 1
```

```
    sw  $v0 valor
```

```
    # printInt
```

```
    li  $v0 1
```

```
    lw  $a0 valor
```

```
    syscall
```

Servicio	Código	Argumentos	Resultado
print_int	1	\$a0 = integer	
read_int	5		integer en \$v0

# Ejercicio (solución simple)



```
// variables globales
```

```
int valor ;
```

```
main ()
```

```
{
```

```
    readInt(&valor) ;
```

```
    valor = valor + 1 ;
```

```
    printInt(valor) ;
```

```
}
```

```
.data
```

```
    # ahorro accesos a memoria
```

```
    # si no son necesarios
```

```
.text
```

```
.globl main
```

```
main:
```

```
    # readInt(&valor)
```

```
    li $v0 5
```

```
    syscall
```

```
    # valor = valor + 1
```

```
    add $a0 $v0 1
```

```
    # printInt
```

```
    li $v0 1
```

```
    syscall
```

Servicio	Código	Argumentos	Resultado
print_int	1	\$a0 = integer	
read_int	5		integer en \$v0

# Contenidos

---

- I. Programación en ensamblador (II)**
  - Arquitectura MIPS (II)
  - Directivas
  - Servicios del sistema
  - 4. Tipo de instrucciones (II)**

# Instrucciones y pseudoinstrucciones

---

- ▶ Las pseudo-instrucciones son instrucciones que no existen en silicio (en el procesador) pero que forman parte del ensamblador.
  - ▶ Ej.: `li $v0 4`  
`move $t1 $t0`
- ▶ En el proceso de ensamblado se sustituyen por la secuencia de instrucciones que si están en silicio y que realizan la misma funcionalidad.
  - ▶ Ej.: `ori $2 $0 $10 # li $v0 4`  
`addu $9, $0, $8 # move $t1 $t0`



# SPIM

Instrucciones en hexadecimal  
Ej: 0x3402000a

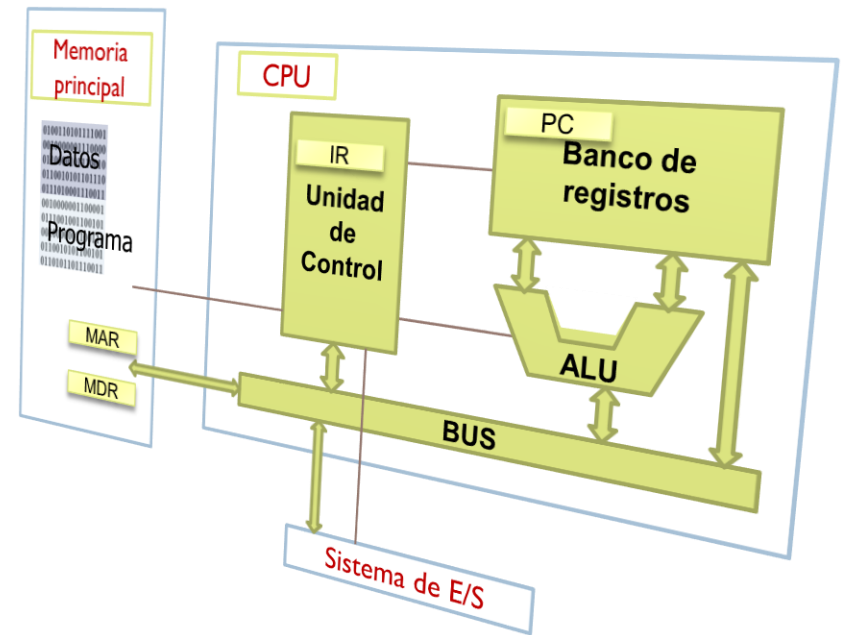
Instrucciones en ensamblador  
Ej: ori \$2 \$0 10

Pseudo-Instrucciones en ensamblador  
Ej: li \$v0 10

The screenshot shows the PCSpim simulator window. At the top, there's a menu bar (File, Simulator, Window, Help) and a toolbar. Below that, a status bar displays PC, Status, EPC, HI, Cause, and BadVAddr. A section titled 'General Registers' lists registers R0 through R31 with their current values. The main window displays assembly code with addresses from 0x00400000 to 0x00400028. The code includes instructions like lw, addiu, sll, addu, jal, nop, ori, syscall, and lui. A blue highlight is on the instruction at address 0x0040001c: ori \$2, \$0, 10. Below the code, there are sections for DATA, STACK, and KERNEL DATA. At the bottom, there's a message: 'Memory and registers cleared and the simulator reinitialized.' and version information for SPIM 7.3.

} Segment de código

# Tipo de instrucciones



- ▶ Transferencias de datos (2)
- ▶ Entrada/Salida
- ▶ Control de flujo

# Transferencia de datos bytes

- ▶ **Transferencias de datos**
- ▶ Entrada/Salida
- ▶ Control de flujo

- ▶ Copia un **byte** de **memoria** a un **registro** o viceversa

- ▶ Para bytes:

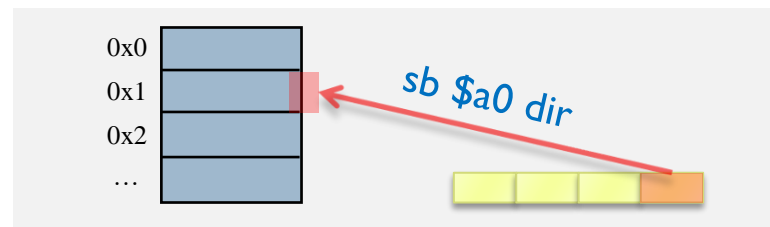
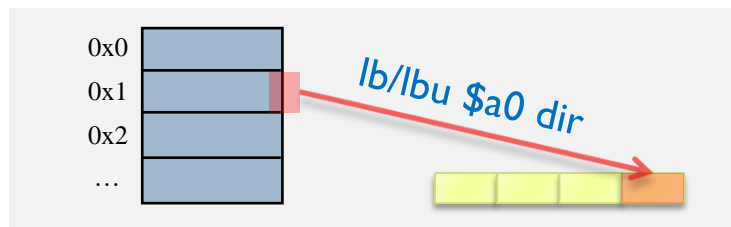
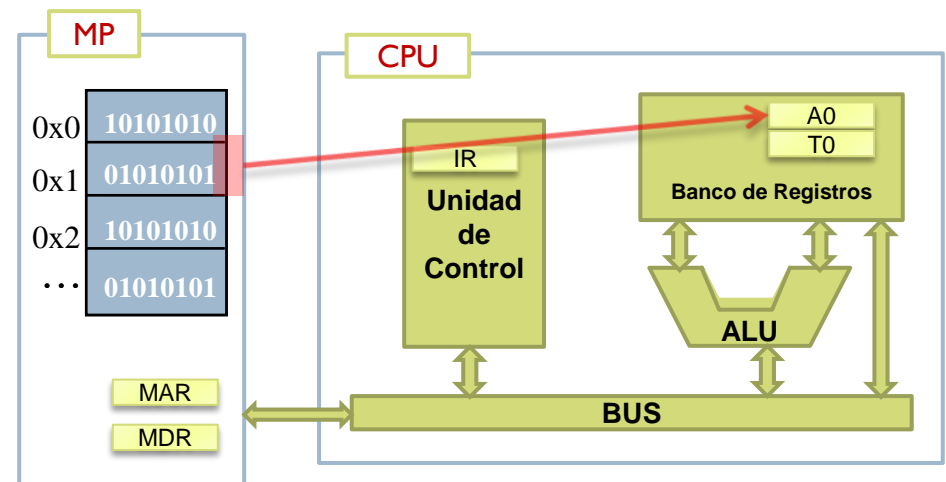
- ▶ Memoria a registro

**lb** \$a0 dir

**lbu** \$a0 dir

- ▶ Registro a memoria

**sb** \$t0 dir



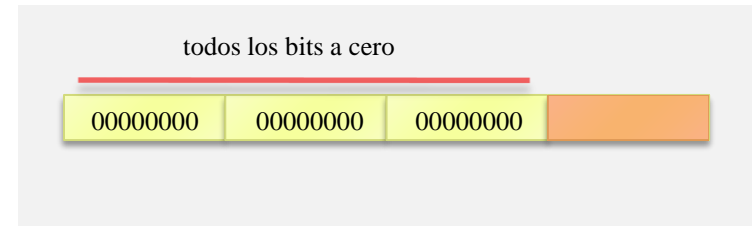
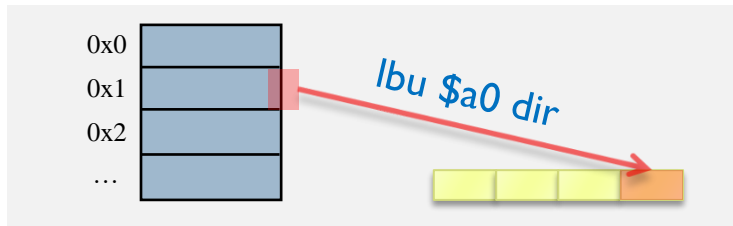
# Transferencia de datos

## Extensión de signo

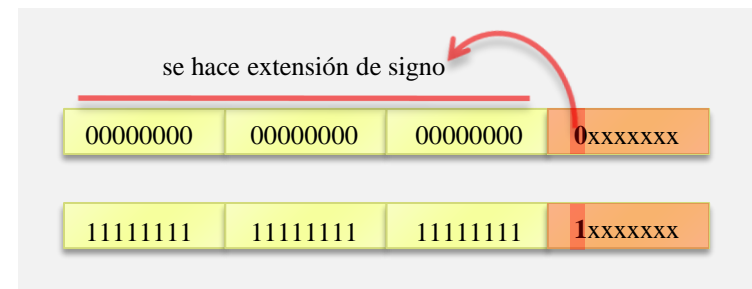
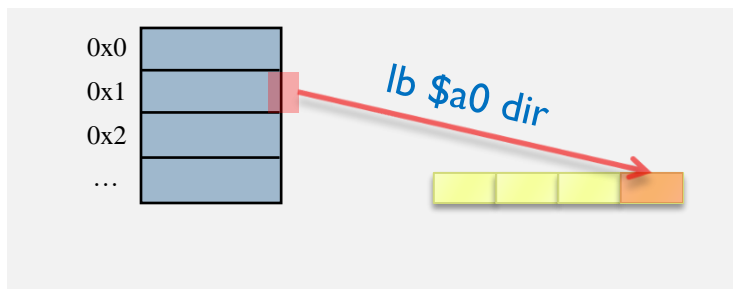
- ▶ Transferencias de datos
- ▶ Entrada/Salida
- ▶ Control de flujo

- ▶ Hay dos posibilidades a la hora de traer un byte de memoria a registro:

- ▶ A) Transferir **sin signo**, por ejemplo: `lbu $a0 dir`



- ▶ B) Transferir **con signo**, por ejemplo: `lb $a0 dir`



# Transferencia de datos

## Direccionamiento

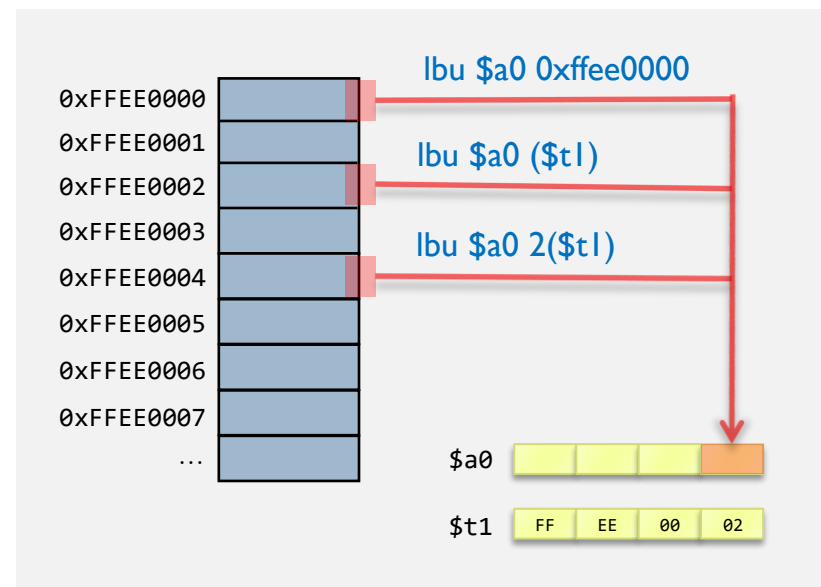
- ▶ **Transferencias de datos**
- ▶ Entrada/Salida
- ▶ Control de flujo

- ▶ Hay tres posibilidades a la hora de indicar la posición de memoria:

- ▶ **A) Directo:**  
`lbu $a0 0x0FFEE0000`

- ▶ **B) Indirecto a registro:**  
`lbu $a0 ($t1)`

- ▶ **C) Relativo a registro:**  
`lbu $a0 2($t1)`



# SPIM (uso de lbu y lb)

```
PCSpim
File Simulator Window Help
[Icons]
Status = 3000ff10 HI = 00000000 LO = 00000000
Registers
R0 (r0) = 00000000 R8 (t0) = e5e5e5e5 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = e5e50000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
R5 (a1) = 7ffff000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7fffeffc

[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 179: jal main
[0x00400018] 0x00000000 nop ; 180: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 182: li $v0 10
[0x00400020] 0x0000000c syscall ; 183: syscall # syscall
[0x00400024] 0x3c01e5e5 lui $1, -6683 ; 12: li $t0 0xe5e5e5e5
[0x00400028] 0x3428e5e5 ori $8, $1, -6683
[0x0040002c] 0x3c011001 lui $1, 4097 ; 13: lbu $t0 b1
[0x00400030] 0x90280000 lbu $8, 0($1)

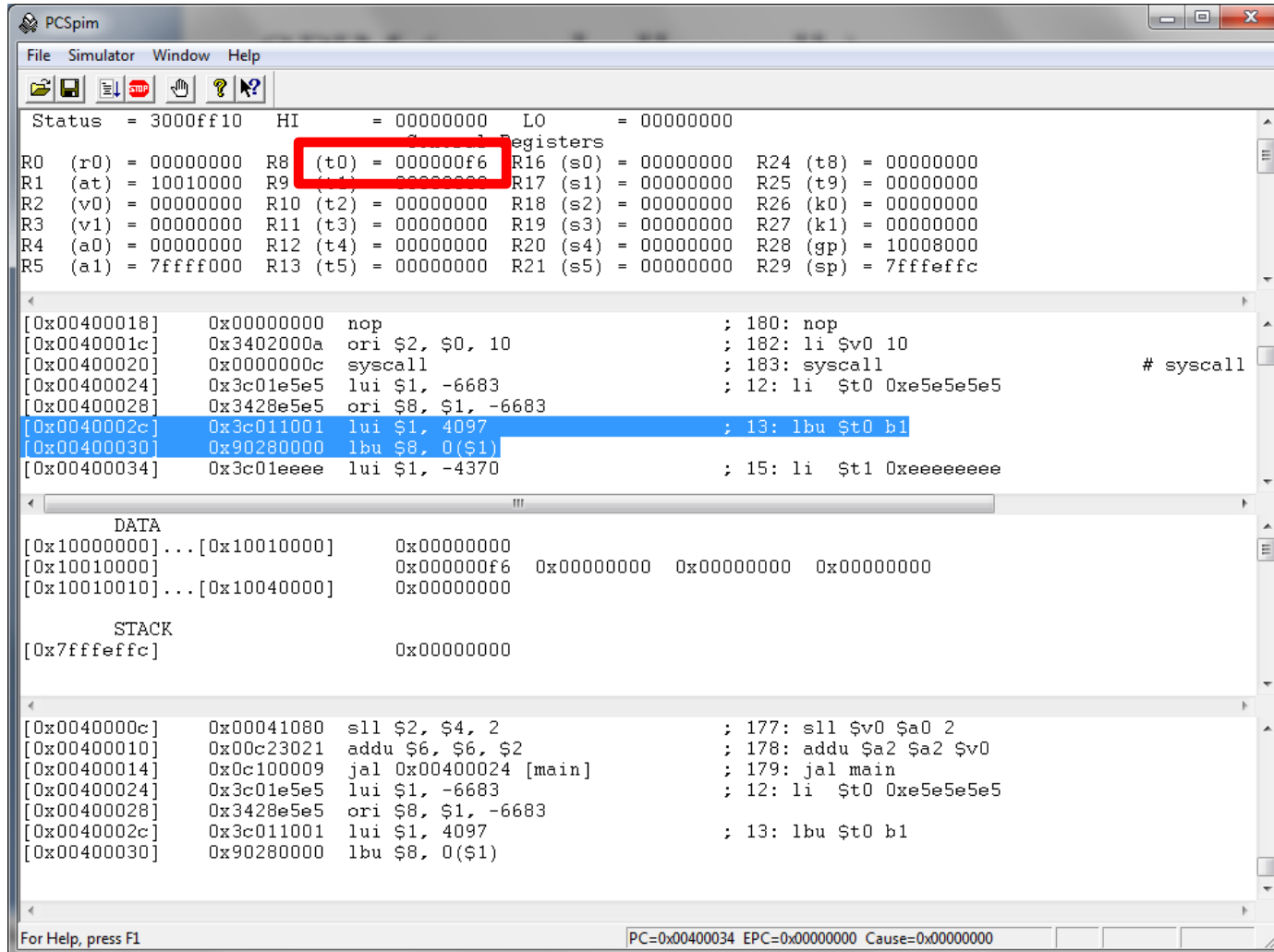
DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x000000f6 0x00000000 0x00000000 0x00000000
[0x10010010]...[0x10040000] 0x00000000

STACK
[0x7fffeffc] 0x00000000

[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 175: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 176: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 177: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 178: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 179: jal main
[0x00400024] 0x3c01e5e5 lui $1, -6683 ; 12: li $t0 0xe5e5e5e5
[0x00400028] 0x3428e5e5 ori $8, $1, -6683

For Help, press F1 PC=0x0040002c EPC=0x00000000 Cause=0x00000000
```

# SPIM (uso de lbu y lb)



```
PCSpim
File Simulator Window Help
[Icons]
Status = 3000ff10 HI = 00000000 LO = 00000000
Registers
R0 (r0) = 00000000 R8 (t0) = 000000f6 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 10010000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
R5 (a1) = 7ffff000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7ffffefc

[0x00400018] 0x00000000 nop ; 180: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 182: li $v0 10
[0x00400020] 0x0000000c syscall ; 183: syscall # syscall
[0x00400024] 0x3c01e5e5 lui $1, -6683 ; 12: li $t0 0xe5e5e5e5
[0x00400028] 0x3428e5e5 ori $8, $1, -6683
[0x0040002c] 0x3c011001 lui $1, 4097 ; 13: lbu $t0 b1
[0x00400030] 0x90280000 lbu $8, 0($1)
[0x00400034] 0x3c01e5ee lui $1, -4370 ; 15: li $t1 0xe5e5e5e5

DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x000000f6 0x00000000 0x00000000 0x00000000
[0x10010010]...[0x10040000] 0x00000000

STACK
[0x7ffffefc] 0x00000000

[0x0040000c] 0x00041080 sll $2, $4, 2 ; 177: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 178: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 179: jal main
[0x00400024] 0x3c01e5e5 lui $1, -6683 ; 12: li $t0 0xe5e5e5e5
[0x00400028] 0x3428e5e5 ori $8, $1, -6683
[0x0040002c] 0x3c011001 lui $1, 4097 ; 13: lbu $t0 b1
[0x00400030] 0x90280000 lbu $8, 0($1)

For Help, press F1 PC=0x00400034 EPC=0x00000000 Cause=0x00000000
```

# Ejemplo (boolean)



```
bool_t b1 ;  
bool_t b2 = false ;  
...
```

```
main ()  
{  
    b1 = true ;  
    ...  
}
```

```
.data  
b1: .space 1      # 1 bytes  
b2: .byte 0  
...  
  
.text  
.globl main  
main: la $t0 b1  
      li $t1 1  
      sb $t1 ($t0)  
      ...
```



# Ejemplo (strings)



```
char c1 ;  
char c2='h' ;  
char *ac1 = "hola" ;
```

```
main ()  
{  
    ac1[0] = 'm' ;  
    printf("%s",ac1) ;  
}
```

```
.data  
c1:  .space 1      # 1 byte  
c2:  .byte 'h'  
ac1: .asciiz "hola"
```

```
.text  
.globl main  
main:  
  
    li $t0 'm'  
    sb $t0 ac1+0  
  
    li $v0 4  
    la $a0 ac1  
    syscall
```

# Ejercicio

---



Realice un programa que calcule el número de caracteres que tiene una tira de caracteres cuya dirección se encuentra en el registro \$a0



```
.data
    string1: .ascii "hola"

.text
.globl main
main:
    la    $a0, string1
```

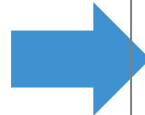
# Ejercicio (solución)



Realice un programa que calcule el número de caracteres que tiene una tira de caracteres cuya dirección se encuentra en el registro \$a0



```
int i;  
  
i=0;  
while($a0[i]) {  
    i = i + 1 ;  
}
```



```
.data  
    string1: .ascii "hola"  
  
.text  
.globl main  
main:  
    la    $a0 string1
```

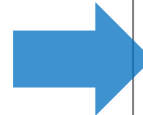
# Ejercicio (solución)



Realice un programa que calcule el número de caracteres que tiene una tira de caracteres cuya dirección se encuentra en el registro \$a0



```
int i;  
  
i=0;  
while($a0[i]) {  
    i = i + 1 ;  
}
```



```
.data  
    string1: .asciiz "hola"  
  
.text  
.globl main  
main:  
    la    $a0 string1  
  
    move $t0 $a0  
    li   $v0 0  
while1: lb    $t1 ($t0)  
    beqz $t1 fin1  
    add  $v0 $v0 1  
    add  $t0 $t0 1  
    b   while1  
  
fin1:
```

# Consejos

---



- ▶ No programar directamente en ensamblador
  - ▶ Mejor **primero hacer diseño** en DFD, Java/C/Pascal...
  - ▶ Ir traduciendo poco a poco el diseño a ensamblador
- ▶ **Comentar** suficientemente el código y datos
  - ▶ Por línea o por grupo de líneas comentar qué parte del diseño implementa.
- ▶ **Probar** con suficientes casos de prueba
  - ▶ Probar que el programa final funciona adecuadamente a las especificaciones dadas

# Transferencia de datos palabras

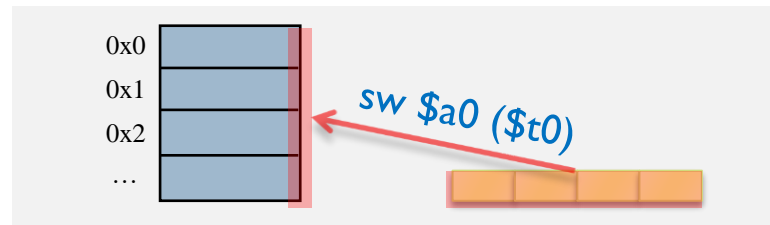
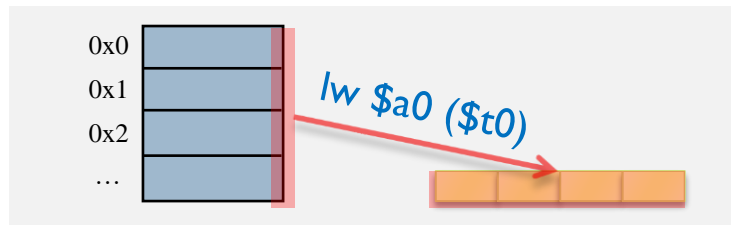
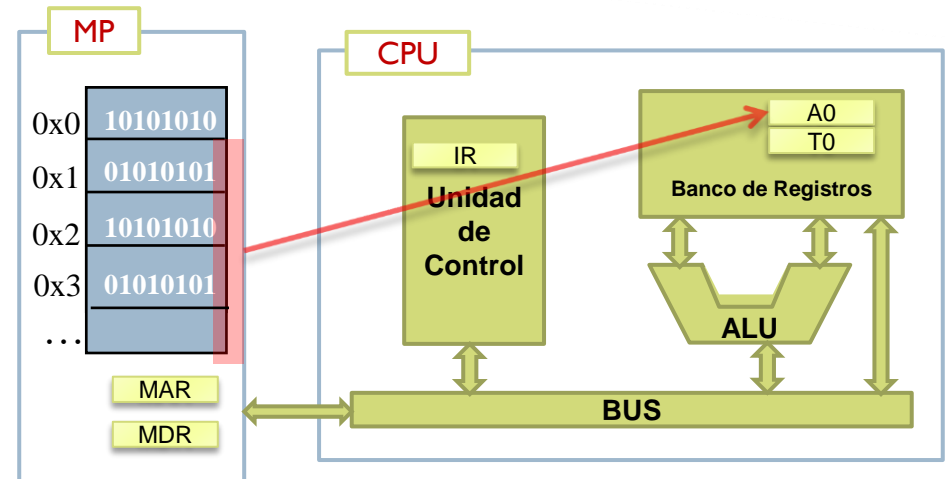
- ▶ Transferencias de datos
- ▶ Entrada/Salida
- ▶ Control de flujo

▶ Copia una **palabra** de **memoria** a un **registro** o viceversa

▶ Ejemplos:

▶ Memoria a registro  
`lw $a0 ($t0)`

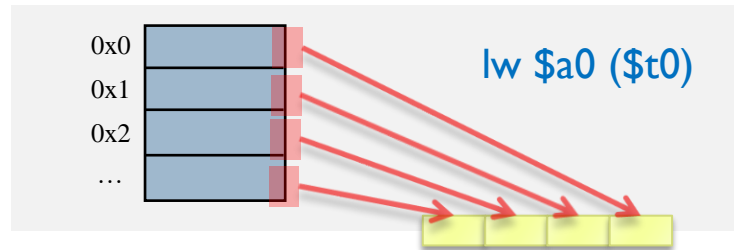
▶ Registro a memoria  
`sw $t0 ($t0)`



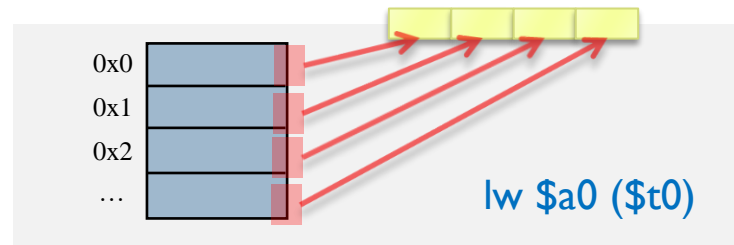
# Transferencia de datos ordenamiento de bytes

## ▶ Hay dos tipos de ordenamiento de bytes:

### ▶ Little-endian (Dirección 'pequeña' termina la palabra...)



### ▶ Big-endian (Dirección 'grande' termina la palabra...)



# Ejemplo

---

endian.s

```
.data
```

```
    b1: .byte 0x00, 0x11, 0x22, 0x33
```

```
.text
```

```
.globl main
```

```
main:
```

```
    lw  $t0 b1
```

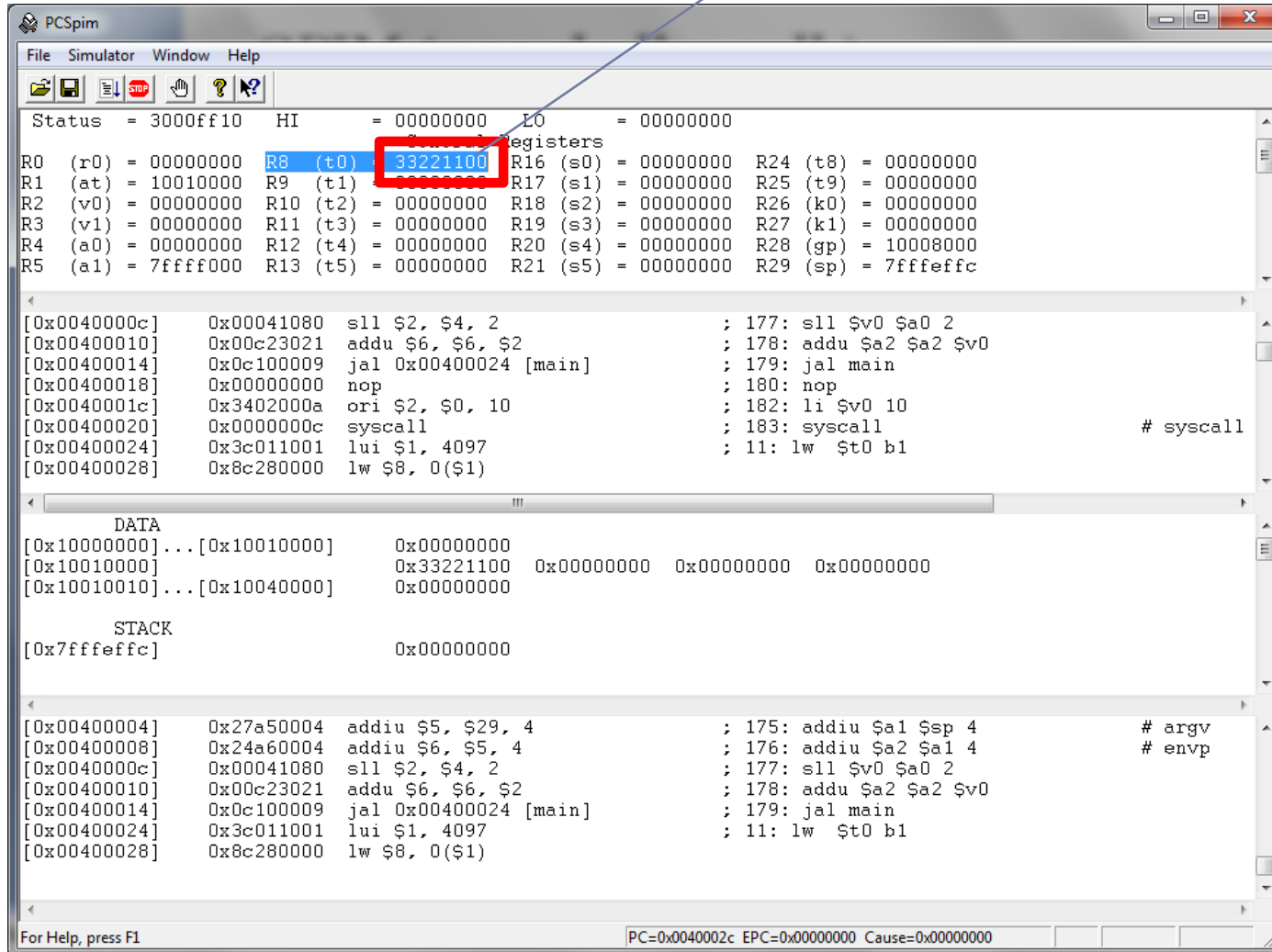




# SPIM

b1: .byte 0x00, 0x11, 0x22, 0x33

Little-endian: dirección 'pequeña' termina la palabra  
Big-endian: dirección 'grande' termina la palabra



```
PCSpim
File Simulator Window Help
[Icons]
Status = 3000ff10 HI = 00000000 LO = 00000000
Registers
R0 (r0) = 00000000 R8 (t0) = 33221100 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 10010000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 10008000
R5 (a1) = 7ffff000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7ffffeffc

[0x0040000c] 0x00041080 sll $2, $4, 2 ; 177: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 178: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 179: jal main
[0x00400018] 0x00000000 nop ; 180: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 182: li $v0 10
[0x00400020] 0x0000000c syscall ; 183: syscall # syscall
[0x00400024] 0x3c011001 lui $1, 4097 ; 11: lw $t0 b1
[0x00400028] 0x8c280000 lw $8, 0($1)

DATA
[0x10000000]...[0x10010000] 0x00000000
[0x10010000] 0x33221100 0x00000000 0x00000000 0x00000000
[0x10010010]...[0x10040000] 0x00000000

STACK
[0x7ffffeffc] 0x00000000

[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 175: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 176: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 177: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 178: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 179: jal main
[0x00400024] 0x3c011001 lui $1, 4097 ; 11: lw $t0 b1
[0x00400028] 0x8c280000 lw $8, 0($1)

For Help, press F1 PC=0x0040002c EPC=0x00000000 Cause=0x00000000
```

# Transferencia de datos palabras

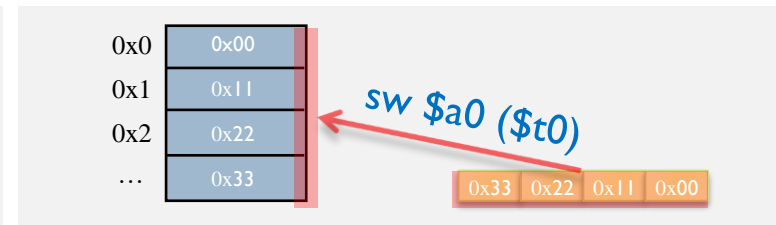
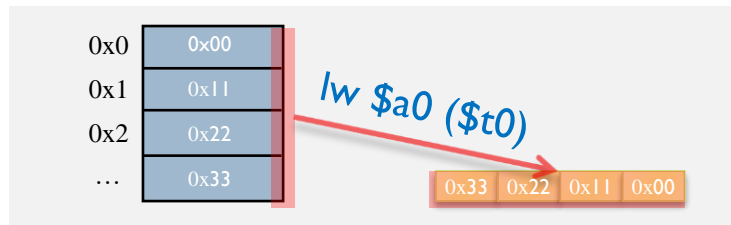
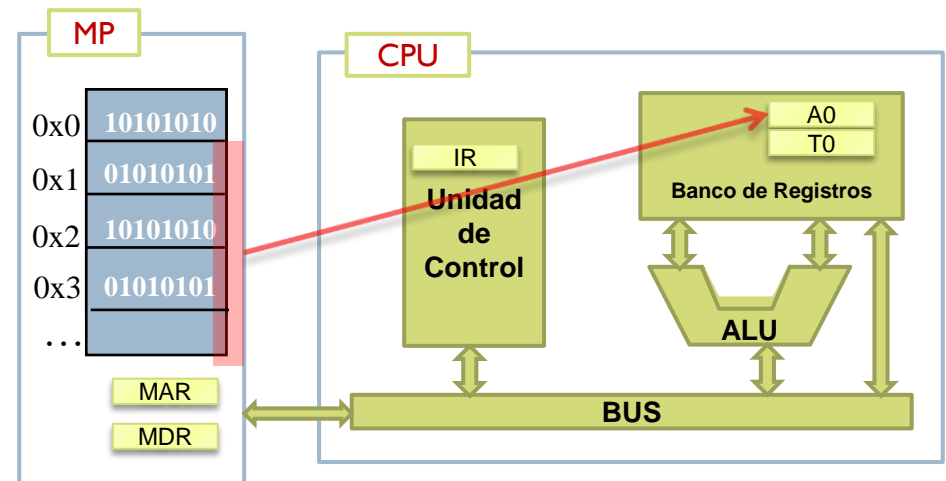
- ▶ Transferencias de datos
- ▶ Entrada/Salida
- ▶ Control de flujo

▶ Copia una **palabra** de **memoria** a un **registro** o viceversa

▶ Ejemplos:

▶ Memoria a registro  
`lw $a0 ($t0)`

▶ Registro a memoria  
`sw $t0 ($t0)`



# Ejemplo (enteros)



```
int resultado ;  
int op1 = 100 ;  
int op2 = -10 ;
```

...

```
main ()
```

```
{  
    resultado = op1+op2;
```

...

```
}
```

```
.data
```

```
resultado: .space 4 # 4 bytes  
op1:      .word 100  
op2:      .word -10
```

...

```
.text
```

```
.globl main
```

```
main: lw $t1 op1  
      lw $t2 op2  
      add $t3 $t1 $t2  
      la $t4 resultado  
      sw $t3 ($t4)
```

...

# Ejemplo (enteros)



```
int vec[5] ;  
int mat[2][3] = {{11,12,13},  
                {21,22,23}};
```

...

```
main ()
```

```
{  
  m[1][2] = m[1][1] + m[2][1];
```

...

```
}
```

```
.data
```

```
vec: .space 20    #5 elem.*4 bytes  
mat: .word 11, 12, 13  
     .word 21, 22, 23
```

...

```
.text
```

```
.globl main
```

```
main:  lw  $t1 mat+0  
      lw  $t2 mat+12  
      add $t3 $t1 $t2  
      sw  $t3 mat+4
```

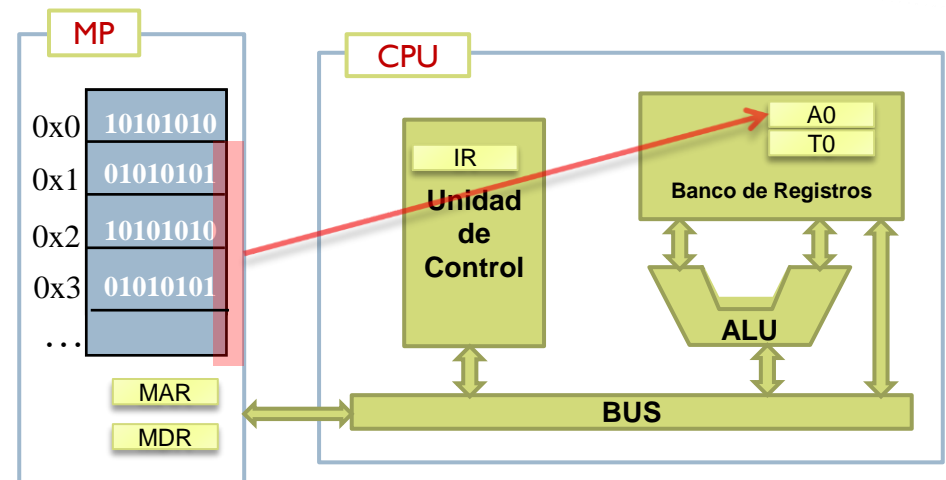
...

# Transferencia de datos alineamiento y tamaño de trabajo

- ▶ **Transferencias de datos**
- ▶ Entrada/Salida
- ▶ Control de flujo

## ▶ Peculiaridades:

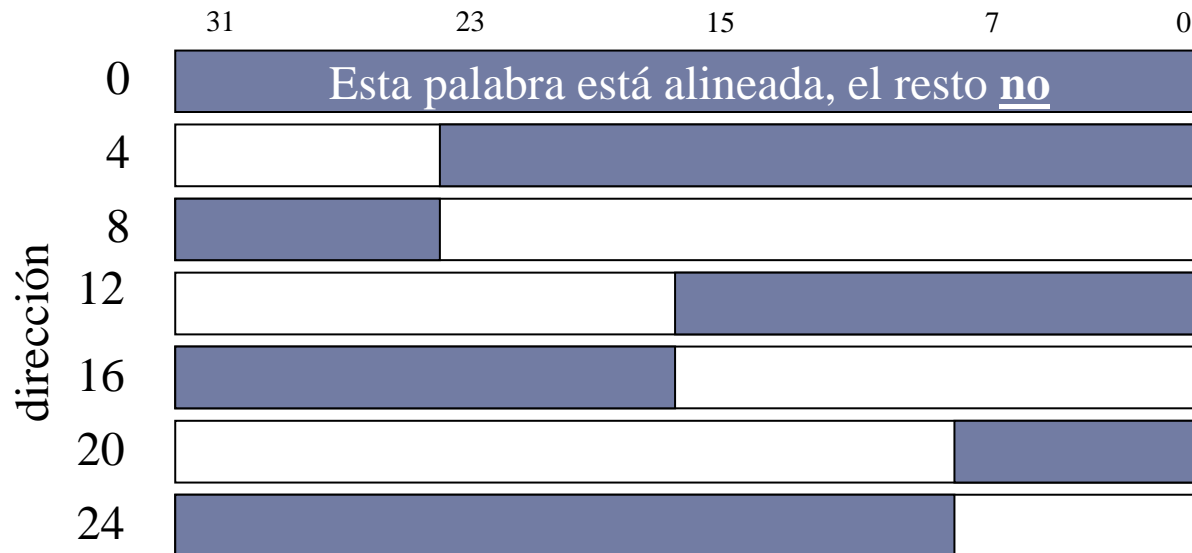
- ▶ Alineamiento de los elementos en memoria
- ▶ Tamaño de trabajo por defecto



# Alineamiento

- ▶ El **alineamiento** supone que la **dirección** sea **múltiplo del tamaño de la palabra**:

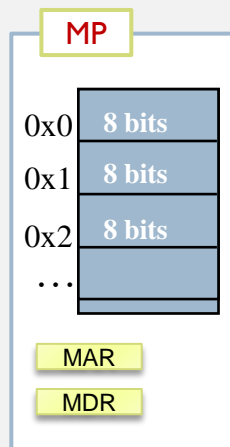
```
.data  
b1: .byte 0x0f  
w1: .word 10
```



# Direccionamiento a nivel de palabra o de byte

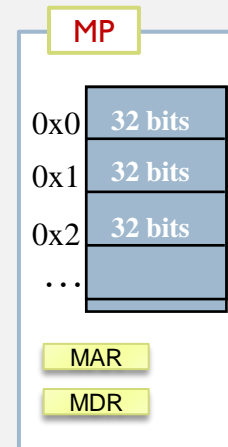
- ▶ La **memoria principal** es similar a un gran vector de una dimensión
- ▶ Una **dirección de memoria** es el índice del vector
- ▶ Hay dos **tipos de direccionamiento**:

- ▶ Direccionamiento a nivel de byte



- ▶ Cada elemento de la memoria es un **byte**
- ▶ Transferir una **palabra** supone transferir **4 bytes**

- ▶ Direccionamiento a nivel de palabra

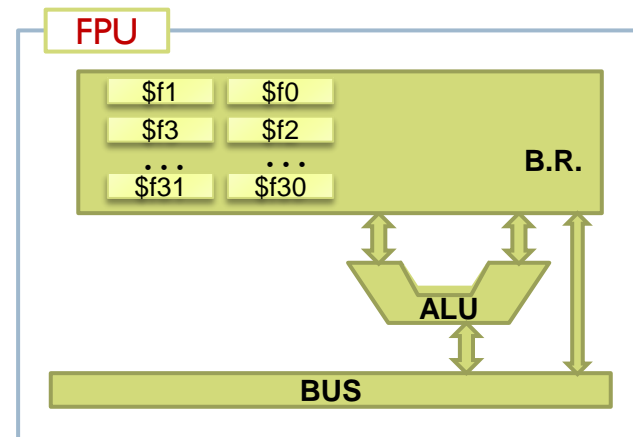


- ▶ Cada elemento de la memoria es una **palabra**
- ▶ En MIPS una palabra son **4 bytes**
- ▶ **lb** supone transferir una **palabra** y quedarse con un **byte**

# Transferencia de datos banco de registros (flotantes)

- ▶ **Transferencias de datos**
- ▶ Entrada/Salida
- ▶ Control de flujo

- ▶ El coprocesador I tiene 32 registros de 32 bits (4 bytes) cada uno
  - ▶ Es posible trabajar con simple o doble precisión
- ▶ Simple precisión (32 bits):
  - ▶ Del \$f0 al \$f31
  - ▶ Ej.: `add.s $f0 $f1 $f5`  
 $f0 = f1 + f5$
  - ▶ Otras operaciones:
    - ▶ `add.s, sub.s, mul.s, div.s, abs.s`
- ▶ Doble precisión (64 bits):
  - ▶ Se utilizan por parejas
  - ▶ Ej.: `add.d $f0 $f2 $f8`  
 $(f0, f1) = (f2, f3) + (f8, f9)$
  - ▶ Otras operaciones:
    - ▶ `add.d, sub.d, mul.d, div.d, abs.d`





# Transferencia de datos

## ieee 754

- ▶ Transferencias de datos
- ▶ Entrada/Salida
- ▶ Control de flujo

▶ Copia una **número** de **memoria** a un **registro** o viceversa

▶ Instrucciones:

▶ Memoria a registro

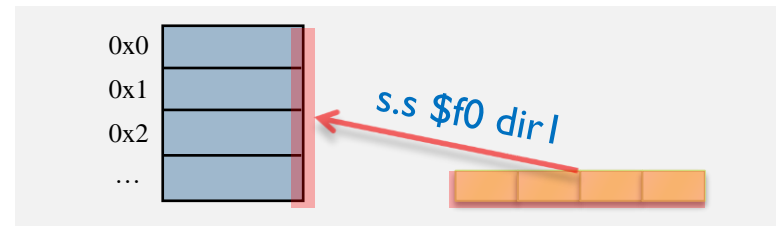
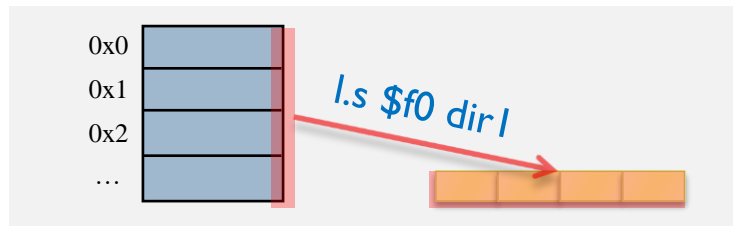
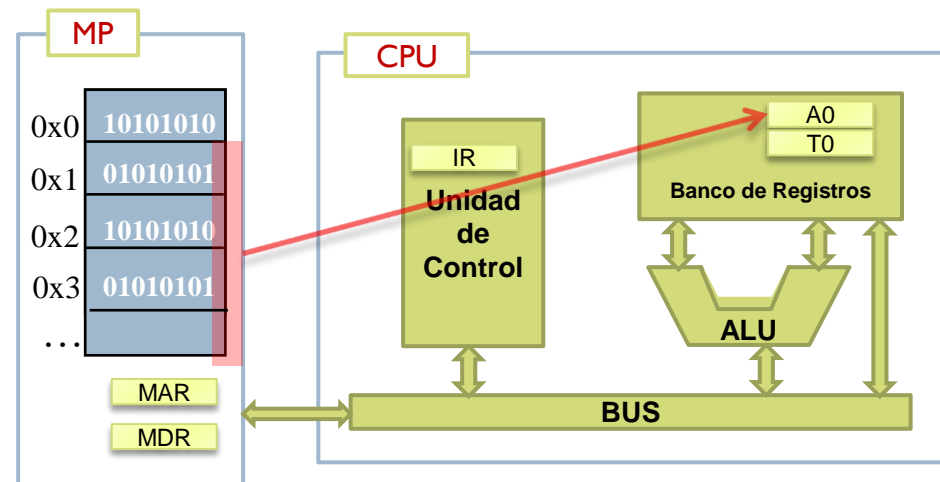
*l.s \$f0 dir1*

*l.d \$f2 dir2*

▶ Registro a memoria

*s.s \$f0 dir1*

*s.d \$f0 dir2*



# Ejercicio

---



```
float resultado ;  
float n1 = 100.0 ;  
float n2 = 10.234 ;  
  
main ()  
{  
    resultado = n1 + n2 ;  
}
```

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

# Ejercicio (solución)

---



```
float resultado ;
float n1 = 100.0 ;
float n2 = 10.234 ;

main ()
{
    resultado = n1 + n2 ;
}
```

```
.data
resultado: .float
n1:        .float 100.0
n2:        .float 10.234

.text
.globl main
main:

    l.s    $f0 n1
    l.s    $f1 n2
    add.s  $f2 $f0 $f1
    s.s    $f2 resultado
```

- ▶ Transferencias de datos
- ▶ **Entrada/Salida**
- ▶ Control de flujo

# Entrada/Salida

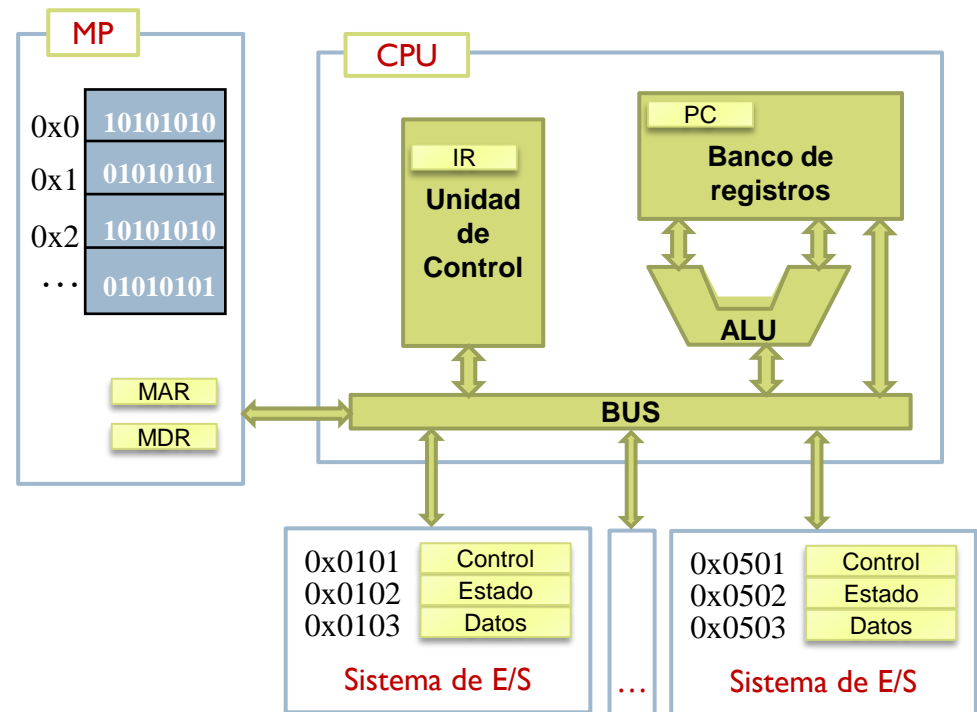
## ▶ Interacción con periféricos

### ▶ Dos tipos:

- ▶ E/S mapeada a memoria

```
lw $t0 0x0502
sw $t0 0x0501
```

- ▶ E/S por puertos
- ```
in $t0 0x0502
out $t0 0x0501
```



- ▶ Transferencias de datos
- ▶ Entrada/Salida
- ▶ Control de flujo

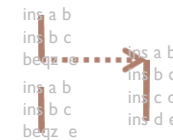
# Control de Flujo (1/2)

- ▶ Cambio de la secuencia de instrucciones a ejecutar (programadas)

- ▶ Distintos tipos:

- ▶ Bifurcación o salto condicional:

- ▶ Saltar a la posición x, si valor  $\leq$  a y
- ▶ Ej: `bne $t0 $t1 0xE00012`



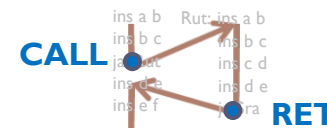
- ▶ Bifurcación o salto incondicional:

- ▶ El salto se realiza siempre
- ▶ Ej: `j 0x10002E`



- ▶ Llamada a procedimiento:

- ▶ Salto con vuelta
- ▶ Ej: `jal 0x20001E ..... jr $ra`

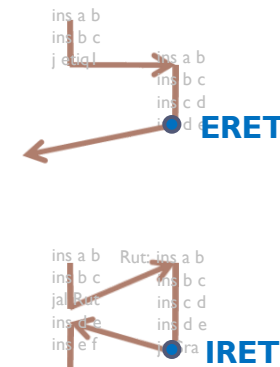


- ▶ Transferencias de datos
- ▶ Entrada/Salida
- ▶ **Control de flujo**

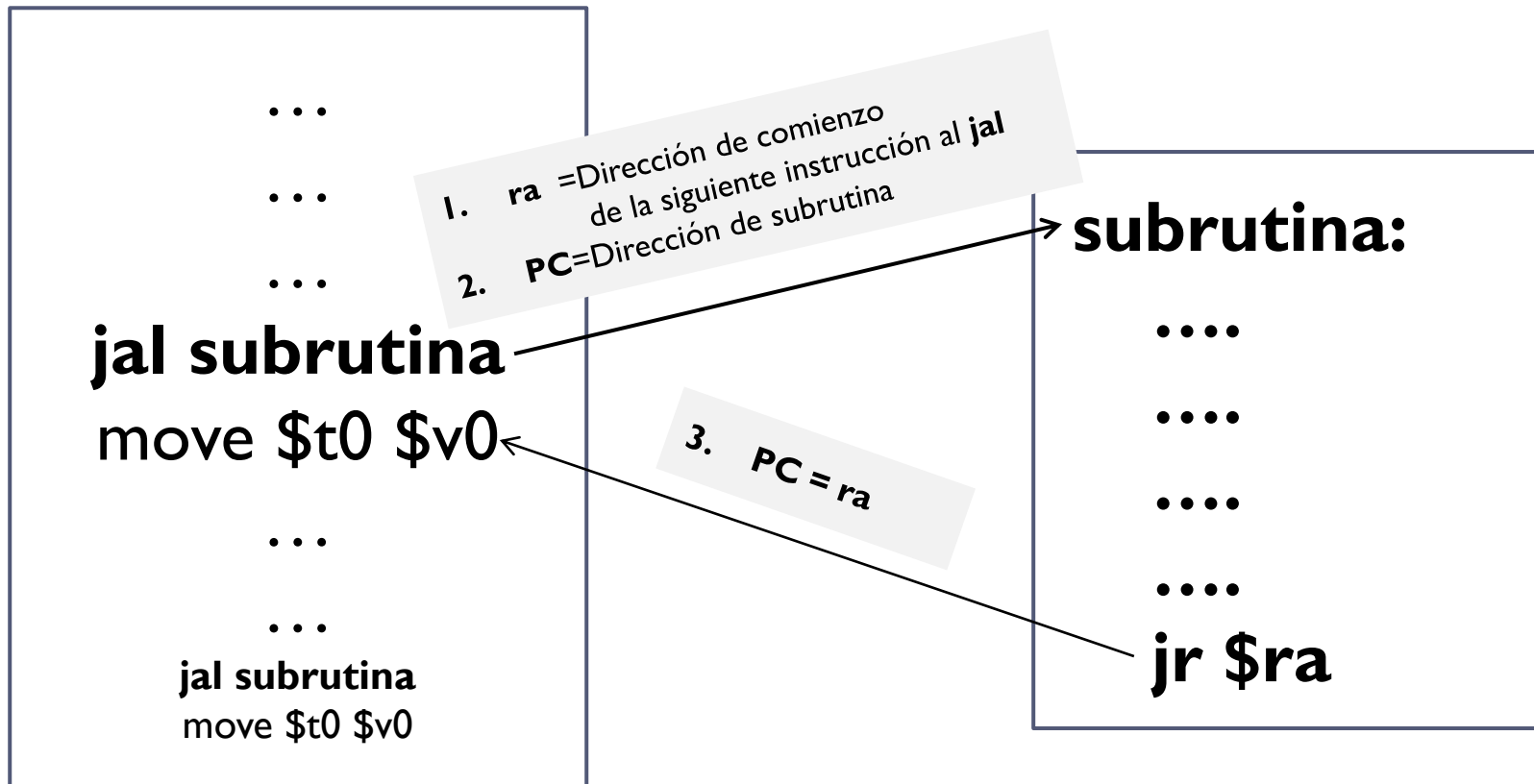
# Control de Flujo (2/2)

---

- ▶ Cambio de la secuencia de instrucciones a ejecutar (no programadas)
- ▶ Distintos tipos:
  - ▶ Excepción:
    - ▶ División 0/0
    - Ej: <excepción> ..... **eret**
  - ▶ Interrupción:
    - ▶ Un periférico necesita atención de CPU
    - Ej: <interrupción> ..... **iret**



# Instrucciones `jal` y `jr`



# Ejemplo básico de jal+jr



```
void di_hola ( void )
{
    printf("hola\n") ;
}
```

```
main ()
{
    di_hola() ;
}
```

```
.data
msg: .asciiz "hola\n"
```

```
.text
.globl main
```

```
di_hola: la $a0 msg
         li $v0 4
         syscall
         jr $ra
```

```
main:   jal di_hola

         li $a0 10
         syscall
```





# Tema 3 (II)

## Fundamentos de la programación en ensamblador



Grupo ARCOS

Estructura de Computadores  
Grado en Ingeniería Informática  
Universidad Carlos III de Madrid