

Dept. de Teoría de la Señal y Comunicaciones y Sistemas  
Telemáticos y Computación  
Área de Telemática (GSyC)

# MARS

## *MIPS Assembler and Runtime Simulator*

Katia Leal Algara

[katia.leal@urjc.es](mailto:katia.leal@urjc.es)

<http://gsyc.escet.urjc.es/~katia/>



### ¿Qué es MARS?

- ❑ Es un entorno de desarrollo interactivo (IDE) para la programación en el lenguaje ensamblador MIPS
- ❑ Orientado a la enseñanza para usarse junto con el “*Computer Organization and Design*” de Patterson y Hennessy

### *Features*

- ☐ Control de la velocidad de ejecución
- ☐ 32 registros visibles de forma simultánea
- ☐ Modificación de los valores en registros y en memoria
- ☐ Posibilidad de mostrar los datos en decimal o en hexadecimal
- ☐ Navegación por la memoria
- ☐ Editor y ensamblador integrados en el propio IDE

### Repertorio de instrucciones

- ☐ Ensambla y simula casi todas las instrucciones documentadas en el libro de texto “*Computer Organization and Design*”, Fourth Edition by Patterson and Hennessy, Elsevier - Morgan Kaufmann, 2009
- ☐ Todas las instrucciones básicas, pseudoinstrucciones, directivas y llamadas al sistemas descritas en el Apéndice B están implementadas

# Características de MARS

## CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)	OPCODE / FUNCT (Hex)
Add	add R	$R[rd] = R[rs] + R[rt]$	(1) 0 / 20 <sub>hex</sub>
Add Immediate	addi I	$R[rt] = R[rs] + \text{SignExtImm}$	(1,2) 8 <sub>hex</sub>
Add Imm. Unsigned	addiu I	$R[rt] = R[rs] + \text{SignExtImm}$	(2) 9 <sub>hex</sub>
Add Unsigned	addu R	$R[rd] = R[rs] + R[rt]$	0 / 21 <sub>hex</sub>
And	and R	$R[rd] = R[rs] \& R[rt]$	0 / 24 <sub>hex</sub>
And Immediate	andi I	$R[rt] = R[rs] \& \text{ZeroExtImm}$	(3) c <sub>hex</sub>
Branch On Equal	beq I	if( $R[rs] == R[rt]$ ) $PC = PC + 4 + \text{BranchAddr}$	(4) 4 <sub>hex</sub>
Branch On Not Equal	bne I	if( $R[rs] != R[rt]$ ) $PC = PC + 4 + \text{BranchAddr}$	(4) 5 <sub>hex</sub>
Jump	j J	$PC = \text{JumpAddr}$	(5) 2 <sub>hex</sub>
Jump And Link	jal J	$R[31] = PC + 8; PC = \text{JumpAddr}$	(5) 3 <sub>hex</sub>
Jump Register	jr R	$PC = R[rs]$	0 / 08 <sub>hex</sub>
Load Byte Unsigned	lbu I	$R[rt] = \{24'b0, M[R[rs] + \text{SignExtImm}](7:0)\}$	(2) 24 <sub>hex</sub>
Load Halfword Unsigned	lhu I	$R[rt] = \{16'b0, M[R[rs] + \text{SignExtImm}](15:0)\}$	(2) 25 <sub>hex</sub>
Load Linked	ll I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2,7) 30 <sub>hex</sub>
Load Upper Imm.	lui I	$R[rt] = \{\text{imm}, 16'b0\}$	f <sub>hex</sub>
Load Word	lw I	$R[rt] = M[R[rs] + \text{SignExtImm}]$	(2) 23 <sub>hex</sub>
Nor	nor R	$R[rd] = \sim (R[rs]   R[rt])$	0 / 27 <sub>hex</sub>
Or	or R	$R[rd] = R[rs]   R[rt]$	0 / 25 <sub>hex</sub>
Or Immediate	ori I	$R[rt] = R[rs]   \text{ZeroExtImm}$	(3) d <sub>hex</sub>
Set Less Than	slt R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	0 / 2a <sub>hex</sub>
Set Less Than Imm.	slti I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2) a <sub>hex</sub>
Set Less Than Imm. Unsigned	sltiu I	$R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$	(2,6) b <sub>hex</sub>
Set Less Than Unsig.	sltu R	$R[rd] = (R[rs] < R[rt]) ? 1 : 0$	(6) 0 / 2b <sub>hex</sub>
Shift Left Logical	sll R	$R[rd] = R[rt] \ll \text{shamt}$	0 / 00 <sub>hex</sub>
Shift Right Logical	srl R	$R[rd] = R[rt] \gg \text{shamt}$	0 / 02 <sub>hex</sub>
Store Byte	sb I	$M[R[rs] + \text{SignExtImm}](7:0) = R[rt](7:0)$	(2) 28 <sub>hex</sub>
Store Conditional	sc I	$M[R[rs] + \text{SignExtImm}] = R[rt];$ $R[rt] = (\text{atomic}) ? 1 : 0$	(2,7) 38 <sub>hex</sub>
Store Halfword	sh I	$M[R[rs] + \text{SignExtImm}](15:0) = R[rt](15:0)$	(2) 29 <sub>hex</sub>
Store Word	sw I	$M[R[rs] + \text{SignExtImm}] = R[rt]$	(2) 2b <sub>hex</sub>
Subtract	sub R	$R[rd] = R[rs] - R[rt]$	(1) 0 / 22 <sub>hex</sub>
Subtract Unsigned	subu R	$R[rd] = R[rs] - R[rt]$	0 / 23 <sub>hex</sub>

- (1) May cause overflow exception  
 (2)  $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$   
 (3)  $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$   
 (4)  $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$   
 (5)  $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$   
 (6) Operands considered unsigned numbers (vs. 2's comp.)  
 (7) Atomic test&set pair;  $R[rt] = 1$  if pair atomic, 0 if not atomic

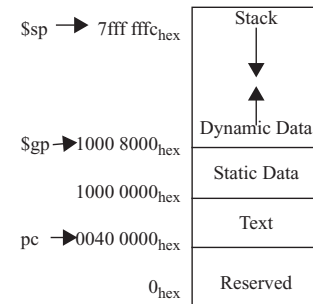
## PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if( $R[rs] < R[rt]$ ) $PC = \text{Label}$
Branch Greater Than	bgt	if( $R[rs] > R[rt]$ ) $PC = \text{Label}$
Branch Less Than or Equal	ble	if( $R[rs] \leq R[rt]$ ) $PC = \text{Label}$
Branch Greater Than or Equal	bge	if( $R[rs] \geq R[rt]$ ) $PC = \text{Label}$
Load Immediate	li	$R[rd] = \text{immediate}$
Move	move	$R[rd] = R[rs]$

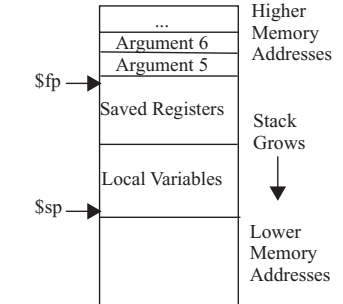
## BASIC INSTRUCTION FORMATS

R	opcode		rs		rt		rd		shamt		funct	
	31	26	25	21	20	16	15	11	10	6	5	0
I	opcode		rs		rt		immediate					
	31	26	25	21	20	16	15	0				
J	opcode		address									
	31	26	25	0								

## MEMORY ALLOCATION



## STACK FRAME



## DATA ALIGNMENT

Double Word									
Word					Word				
Halfword		Halfword			Halfword		Halfword		
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7	8	9

Value of three least significant bits of byte address (Big Endian)

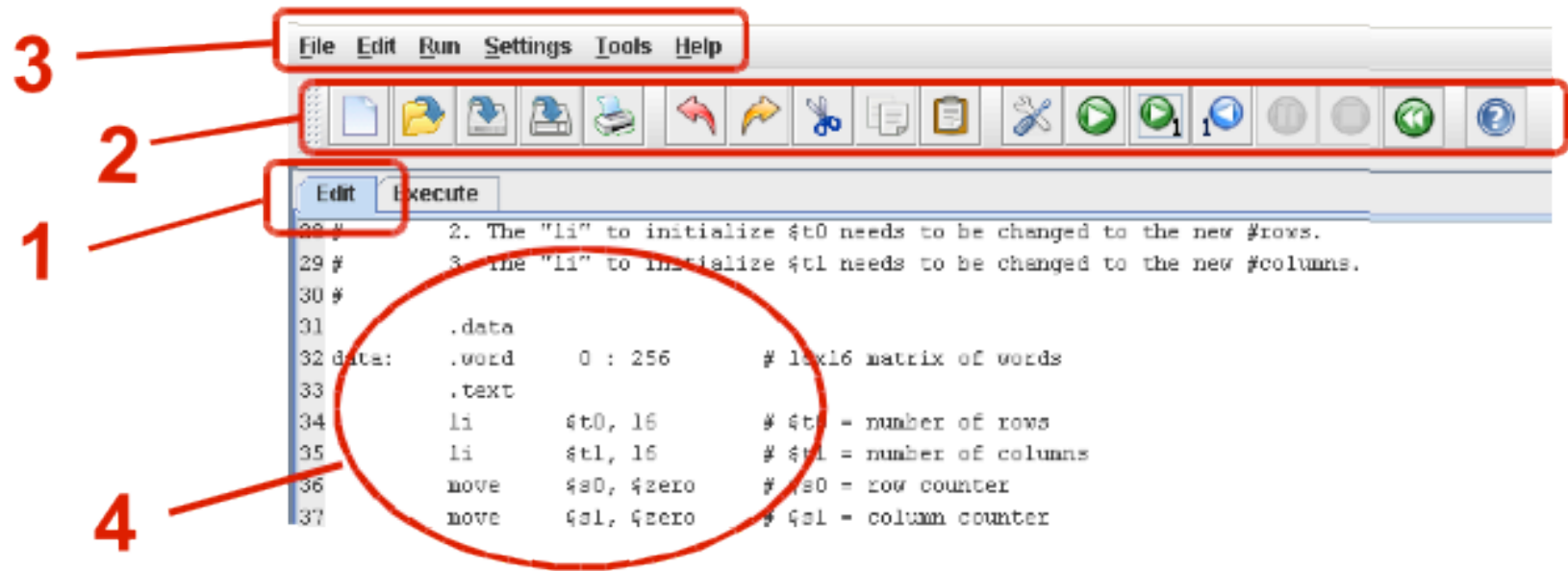
### Llamadas al sistema

- ☐ Varios servicios de sistema, principalmente para la realización de operaciones de entrada/salida, están disponibles:
  - ☐ print integer
  - ☐ print float
  - ☐ print double
  - ☐ print string
  - ☐ sbrk (allocate heap memory)
  - ☐ exit (terminate execution)
  - ☐ print character
  - ☐ read character
  - ☐ MIDI out
  - ☐ ...

(<http://courses.missouristate.edu/KenVollmar/MARS/Help/SyscallHelp.html>)


## Características de MARS

Ventana de edición: similar al Notepad de Windows



1. Edit display is indicated by highlighted tab.
- 2, 3. Typical edit and execute operations are available through icons and menus, dimmed-out when unavailable or not applicable.
4. WYSIWYG editor for MIPS assembly language code.

### Ventana de ejecución

- ☐ Para ensamblar un programa hay que pinchar en el icono 
- ☐ Si no hay errores de ensamblado, se abre la ventana de ejecución
- ☐ En caso de que el ensamblado falle, se muestra una ventana con los errores y su correspondiente número de línea

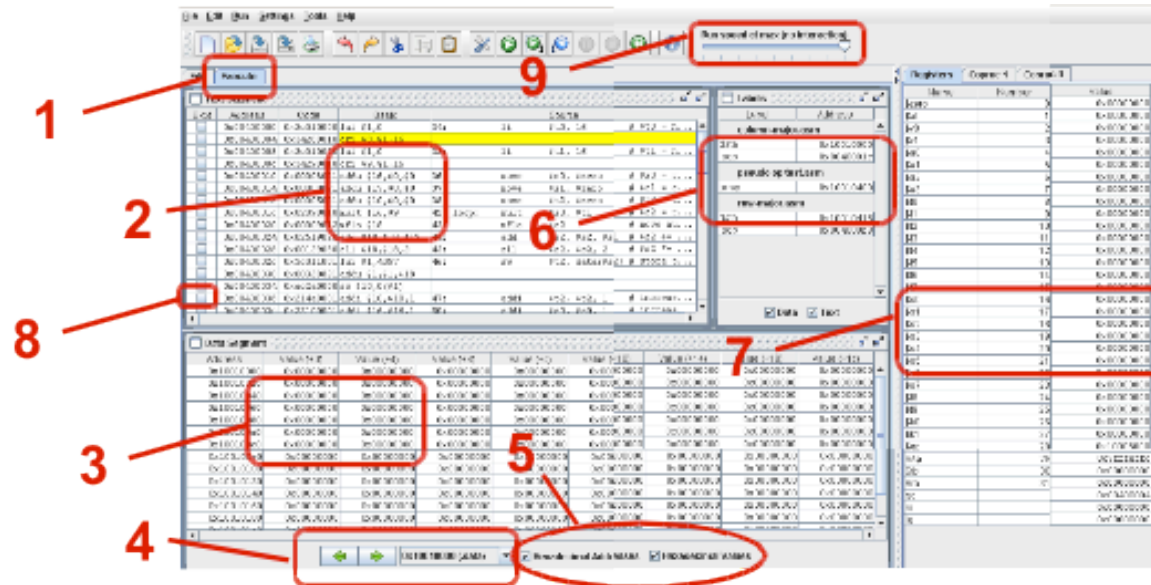


### Ventana de ejecución

- ☐ Muestra varias ventanas
- ☐ Ventana ***Text Segment***:
  - ☐ Muestra el código fuente y el binario
  - ☐ Se puede incluir un *breakpoint* en cualquier instrucción marcando el *check box* correspondiente
  - ☐ La siguiente instrucción en ser ejecutada aparece resaltada

# Características de MARS

## Ventana de ejecución



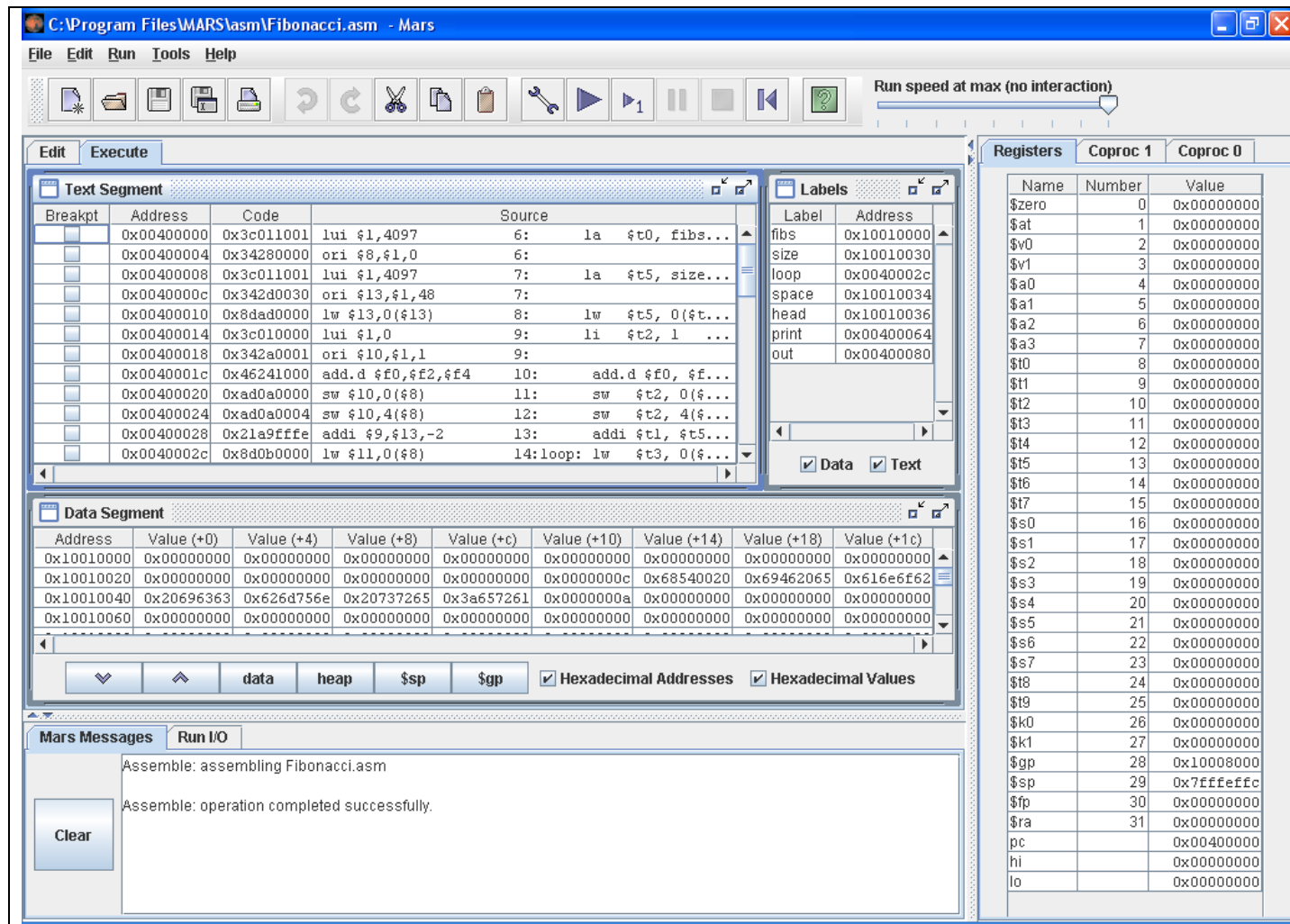
1. Execute display is indicated by highlighted tab.
2. Assembly code is displayed with its address, machine code, assembly code, and the corresponding line from the source code file. (Source code and assembly code will differ when pseudoinstructions have been used.)
3. The values stored in Memory are directly editable (similar to a spreadsheet).
4. The window onto the Memory display is controlled in several ways: previous/next arrows and a menu of common locations (e.g., top of stack).
5. The numeric base used for the display of data values and addresses (memory and registers) is selectable between decimal and hexadecimal.
6. Addresses of labels and data declarations are available. Typically, these are used only when single-stepping to verify that an address is as expected.
7. The values stored in Registers are directly editable (similar to a spreadsheet).
8. Breakpoints are set by a checkbox for each assembly instruction. These checkboxes are always displayed and available.
9. Selectable speed of execution allows the user to "watch the action" instead of the assembly program finishing directly.

### Ventana de ejecución

- ☐ Ventana ***Data Segment:***
  - ☐ Datos almacenados por el programa
  - ☐ Controles para mostrar el contenido de partes especiales de la memoria, como la pila o el *heap*
  - ☐ Posibilidad de mostrar el contenido de las posiciones de memoria en hexadecimal o en decimal
- ☐ Ventana ***Labels:*** tabla de símbolos
  - ☐ Muestra el valor de las etiquetas creadas en por el programa
- ☐ Ventana ***Registers:***
  - ☐ Muestra el contenido de los registro, tanto en decimal como en hexadecimal
  - ☐ Existen ventanas separadas para los registros de propósito general, los registros de coma flotante del *Coprocessor 1* y los registros de excepción del *Coprocessor 0*

# Características de MARS

## Ventana de ejecución



### Ventana de ejecución

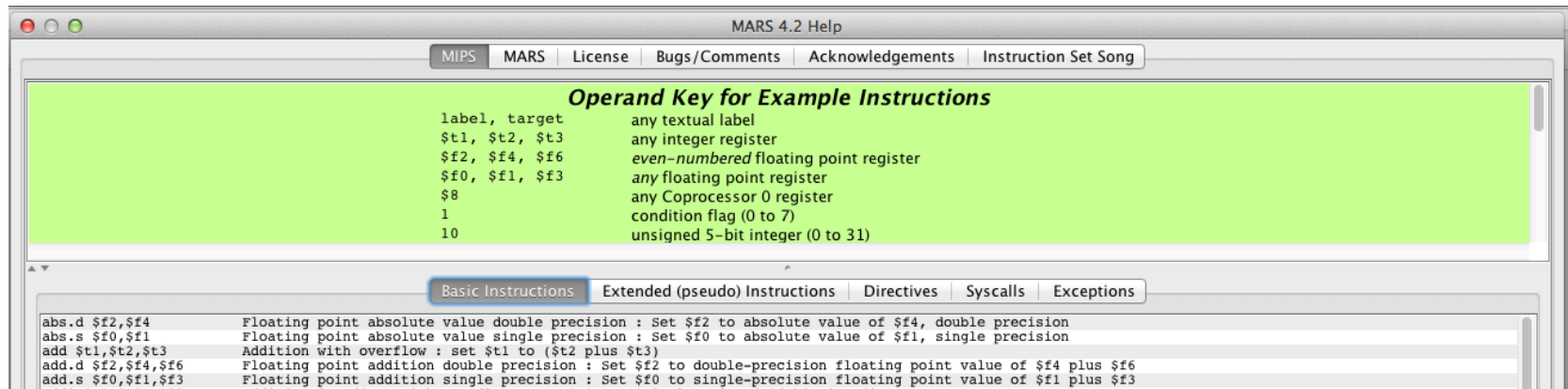
- ☐ Consola
  - ☐ Mensajes de MARS, mensajes de error de ensamblado
  - ☐ Mensajes de entrada/salida generados en tiempo de ejecución por las llamadas al sistema
  - ☐ Estas ventanas se activan cuando se escribe texto sobre ellas

### Instrucciones soportadas

- ☐ El listado completo de instrucciones, pseudo instrucciones, llamadas al sistema, directivas y excepciones soportadas se puede consultar en la ayuda de MARS (Help-F1)
- ☐ Así mismo, también se puede consultar el Apéndice B del “*Computer organization and design. The hardware/software interface*”, 4 edition. Morgan Kaufmann, 2012
- ☐ El mismo apéndice se puede descargar de forma gratuita en:  
[http://www.cs.wisc.edu/~larus/HP\\_AppA.pdf](http://www.cs.wisc.edu/~larus/HP_AppA.pdf)

# Características de MARS

## Instrucciones soportadas



### ¿Cuándo programar en ensamblador?

- ☐ Cuando el **tamaño** o la **velocidad** de un programa sean críticos
- ☐ En la mayoría de sistemas embebidos, es necesaria una respuesta rápida y fiable
- ☐ Es difícil para los programadores asegurar que un programa en lenguaje de alto nivel responde en un intervalo de tiempo determinado: tiempo de respuesta
- ☐ Reducir el tamaño de un programa reduce costes, puesto que se necesitan menos pastilla de memoria
- ☐ Se pueden identificar las partes críticas de un programa, aquellas en las que se emplea más tiempo, para **recodificarlas** en lenguaje ensamblador
- ☐ Programar en lenguaje ensamblador nos permite explotar **instrucciones especializadas**: copia de strings
- ☐ Cuando no hay disponible un lenguaje de programación para un ordenador particular

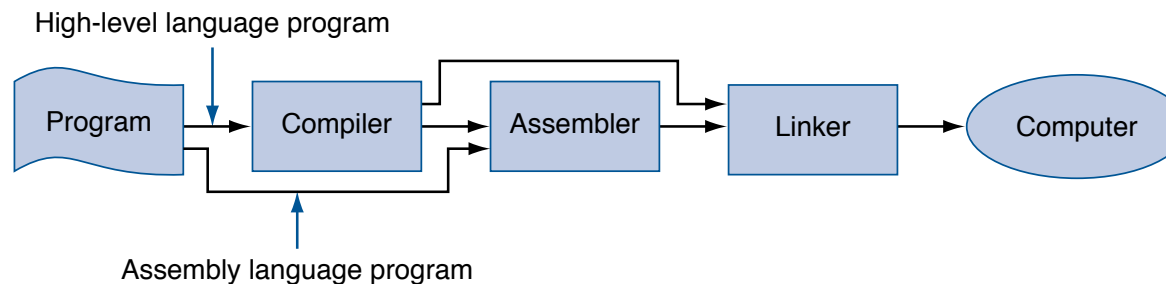


### Ensambladores

- ☐ Un programa ensamblador traduce un fichero con sentencias en lenguaje ensamblador en un fichero con instrucciones máquina y datos binarios
- ☐ El proceso de traducción tienen dos partes:
  1. Localizar **posiciones de memoria etiquetadas**, de tal forma que la dirección de un nombre simbólico se conozca cuando las instrucciones se traduzcan
  2. Traducir cada sentencia en ensamblador combinando códigos de operación, identificadores de registros y etiquetas en una **instrucción legal**
- ☐ El **código objeto** o fichero objeto no puede ser ejecutado puesto que incluye referencias a datos o procedimientos en otros ficheros
  - ☐ Variables externas o globales
  - ☐ Variables locales

## Linkadores o enlazadores

- ❑ El programa ensamblador depende de otra herramienta, el **linkador**
- ❑ El linkador combina un conjunto de ficheros objeto y librerías en un fichero ejecutable resolviendo las referencias a etiquetas externas



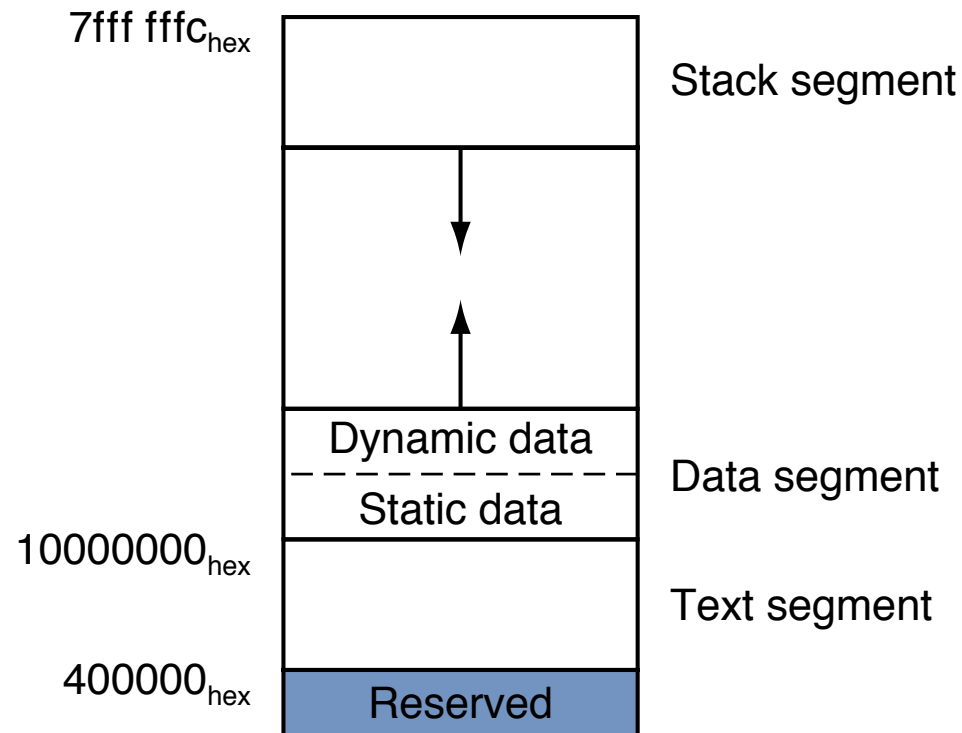
### Formato de un fichero objeto en UNIX

Object file header	Text segment	Data segment	Relocation information	Symbol table	Debugging information
--------------------	--------------	--------------	------------------------	--------------	-----------------------

- ❑ **Cabecera:** contiene el tamaño y la posición del resto de campos del fichero
- ❑ ***Text segment*:** contiene el código en lenguaje máquina de las rutinas del fichero fuente
- ❑ ***Data segment*:** contiene la representación binaria de los datos inicializados utilizados por el programa
- ❑ **Información de realojamiento:** información sobre aquellas instrucciones y datos que dependen de direcciones absolutas
- ❑ **Tabla de símbolos:** asocia direcciones con etiquetas externas y contiene una lista de referencias sin resolver
- ❑ **Información de depuración:** información para el *debugger*

## Gestión de la memoria

- ❑ Los sistemas basados en el procesador MIPS dividen la memoria en tres partes



- ❑ Limitations of MARS as of Release 4.2 include memory segments (text, data, stack, kernel text, kernel data) are limited to 4MB each starting at their respective base addresses

### Modos de direccionamiento

- ❑ MIPS es una arquitectura carga-almacenamiento, lo que significa que sólo las instrucciones de carga y almacenamiento pueden acceder a memoria
- ❑ Las instrucciones computacionales operan sólo sobre datos almacenados en registros

Format	Address computation
(register)	contents of register
imm	immediate
imm (register)	immediate + contents of register
label	address of label
label $\pm$ imm	address of label + or – immediate
label $\pm$ imm (register)	address of label + or – (immediate + contents of register)

### Ordenación de los bytes

- ☐ Los bytes de una palabra se pueden enumerar
- ☐ La convención que utiliza una máquina se denomina ordenación de los bytes
- ☐ El procesador MIPS puede operar tanto en ***big-endian*** como en ***little-endian***
- ☐ Por ejemplo, en una máquina big-endian, la directiva `.byte 0, 1, 2, 3` resulta en que una palabra de memoria contiene:

Byte #			
0	1	2	3

- ☐ Mientras que si la máquina fuera little-endian la palabra contendría:

Byte #			
3	2	1	0

### Sintaxis del ensamblador

#### ☐ **Comentarios**

- ☐ “#”
- ☐ Todo lo que hay después de este carácter se ignora

#### ☐ **Etiquetas**

- ☐ Se utilizan para referirse a posiciones de memoria o a instrucciones
- ☐ Son *case insensitive*
- ☐ Sólo se puede utilizar una etiqueta por línea
- ☐ Van seguidas de “:”

#### ☐ **Secciones más importantes:**

- ☐ **.data**
- ☐ **.text**

### Sección `.data` <addr>

- ☐ Comandos que especifican **cómo se debe rellenar la memoria** antes de que el programa comience su ejecución
- ☐ El formato de un comando `.data` es:  
`[label:] .datatype val1 [,val2 [, ...]]`
- ☐ Tipos de datos representados:
  - ☐ `.ascii str`  
Acepta strings que contienen caracteres ASCII+secuencias de escape y los almacena en memoria
  - ☐ `.asciiz str`  
Como `.ascii`, solo que termina los strings con el byte cero



### Sección `.data` <addr>

- ❑ Los caracteres especiales dentro de un string siguen la convención de C

Escaping sequence	Meaning	ASCII encoding
<code>\0</code>	Null byte	0
<code>\t</code>	Horizontal tabulation	9
<code>\n</code>	Newline character	10
<code>\"</code>	Literal quote character	34
<code>\\</code>	Literal backslash character	92

- ❑ `.byte b1, ..., bn`

Almacena  $n$  valores de 8 bits en bytes sucesivos de memoria

- ❑ `.half h1, ..., hn`

Almacena  $n$  valores de 16 bits en medias palabras sucesivas de memoria

- ❑ `.word w1, ..., wn`

Almacena  $n$  valores de 32 bits en palabras sucesivas de memoria

- ❑ `.space n`

Reserva  $n$  bytes de espacio en el segmento de datos

### Sección `.text` <addr>

- ❑ Instrucciones simbólicas que serán codificadas y ejecutadas cuando el programa comience

- ❑ Formato de un comando `.text`:

```
[label:] instruction [p1 [, p2 [, p3]]]
```

Donde `instruction` es el nombre de la instrucción y `p1`, `p2` y `p3` son los tres operandos

- ❑ El tipo y número de los operandos viene determinado por el tipo de instrucción

# Sintaxis del ensamblador

## Ejemplo

```
.data
fibs: .word 0 : 12      # "array" of 12 words to contain fib values
size: .word 12          # size of "array"
.text
main: la $t0, fibs      # load address of array
      la $t5, size      # load address of size variable
      lw $t5, 0($t5)    # load array size
      li $t2, 1         # 1 is first and second Fib. Number
      sw $t2, 0($t0)    # F[0] = 1
      sw $t2, 4($t0)    # F[1] = F[0] = 1
      addi $t1, $t5, -2 # Counter for loop, will execute (size-2) times
loop: lw $t3, 0($t0)    # Get value from array F[n]
      lw $t4, 4($t0)    # Get value from array F[n+1]
      add $t2, $t3, $t4 # $t2 = F[n] + F[n+1]
      sw $t2, 8($t0)    # Store F[n+2] = F[n] + F[n+1] in array
      addi $t0, $t0, 4  # increment address of Fib. number source
      addi $t1, $t1, -1 # decrement loop counter
      bgtz $t1, loop    # repeat if not finished yet.
      li $v0, 10        # system call for exit
      syscall          # we are out of here.
```

### Otras convenciones que veremos ...

- ☐ Llamadas al sistema
- ☐ Entrada/Salida
- ☐ Llamada a subrutina