

PRÁCTICA 3. INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR - MARS

dtic



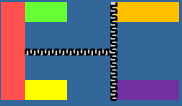


PRÁCTICA 3. INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR- MARS

Índice

- ⊙ Introducción arquitectura MIPS
- ⊙ Programación en ensamblador
- ⊙ Juego de instrucciones MIPS
- ⊙ Llamadas a procedimientos
- ⊙ El simulador MARS
- ⊙ Ejemplo sencillo
- ⊙ Ejercicios propuestos



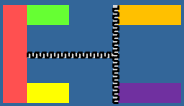


Introducción MIPS

Introducción MIPS

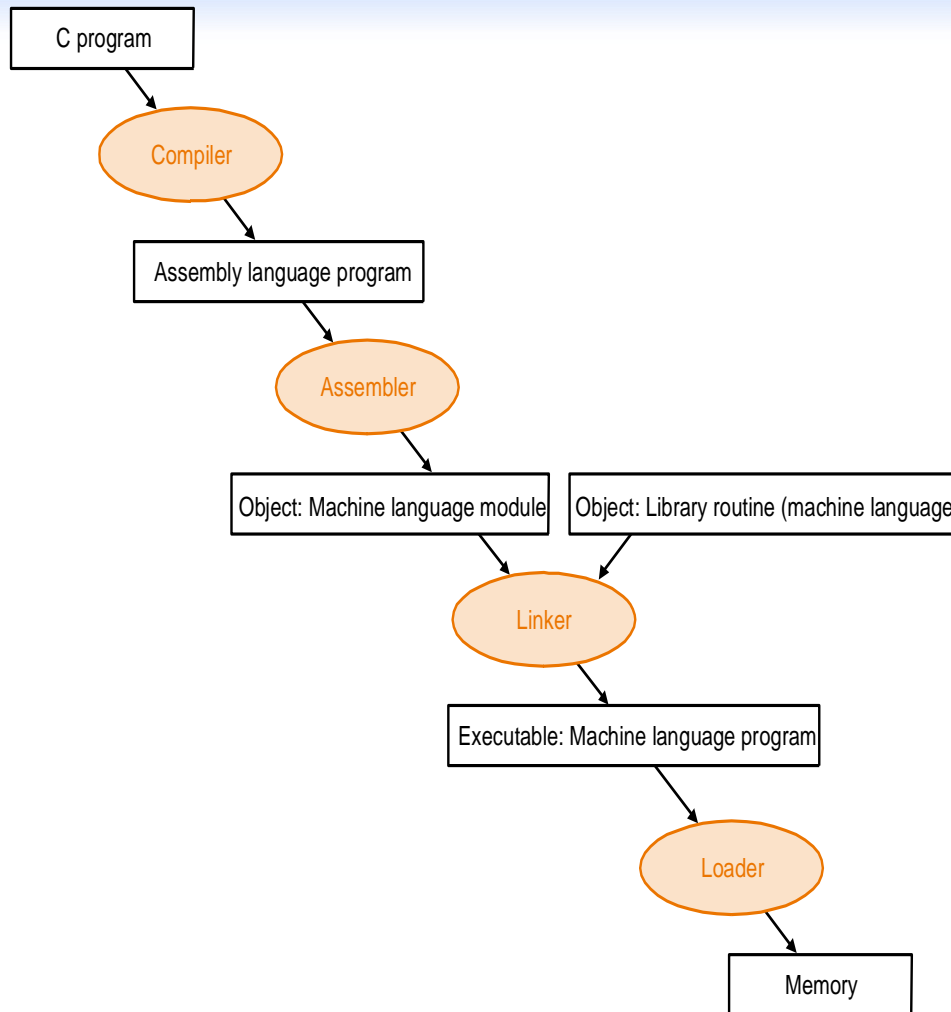
- ⊙ MIPS, acrónimo de Microprocessor without Interlocked Pipeline Stages.
- ⊙ Diseñado por John L. Hennessy en 1981 en la Universidad de Stanford y fundó la compañía MIPS Computer Systems Inc.
- ⊙ Es un diseño RISC
- ⊙ Varias versiones: 32 bits (**R2000** y R3000), 64 bits (R4000)
- ⊙ Aplicaciones
 - ⊙ Routers Cisco y Linksys
 - ⊙ Videoconsolas como la Nintendo 64 o las Sony PlayStation
- ⊙ Simuladores: **MARS**, QTSPIM, XSPIM, ...





Introducción MIPS

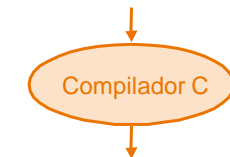
Jerarquía de traducción



Programa
en lenguaje
de alto
nivel
(en C)

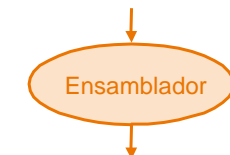
```
swap(int v[], int k)
{int temp;
 temp = v[k];
 v[k] = v[k+1];
 v[k+1] = temp;
}
```

Programa
En lenguaje
Ensamblador
(para MIPS)



```
swap:
    muli $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

Programa
En lenguaje
Máquina
(para MIPS)

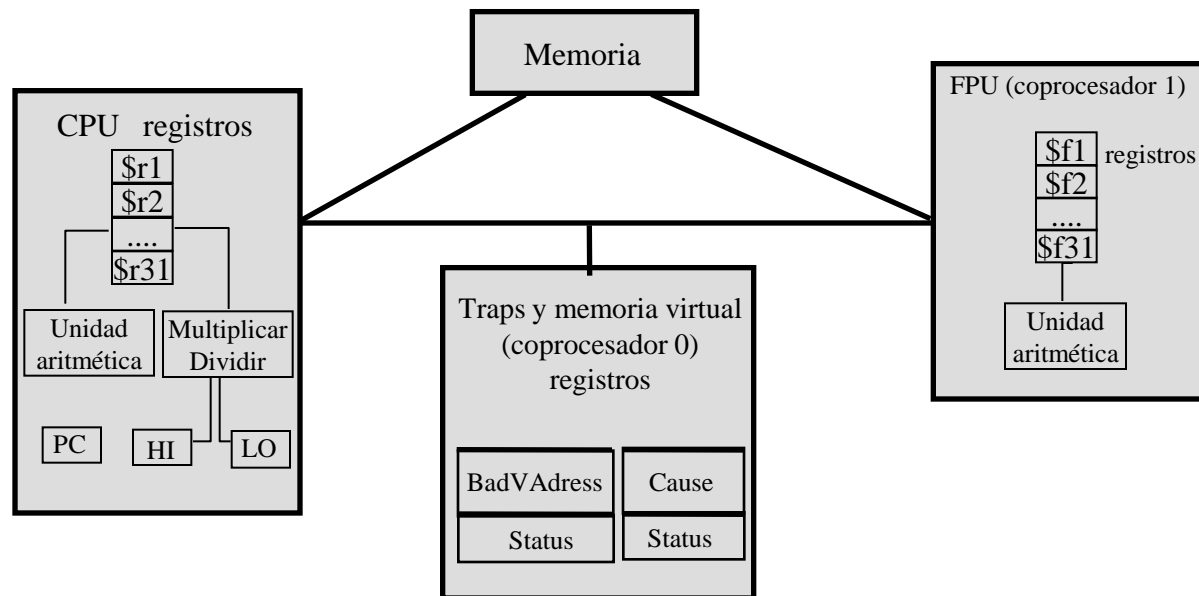


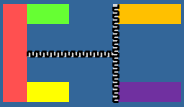
```
000000001010000100000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
10101100111100100000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

Arquitectura MIPS R2000

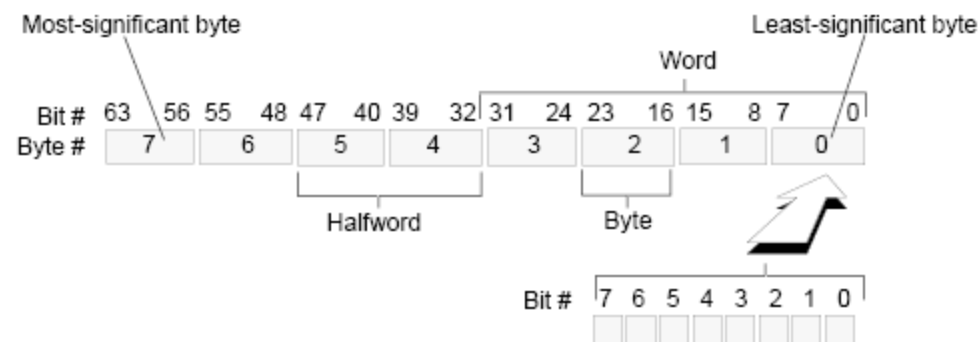
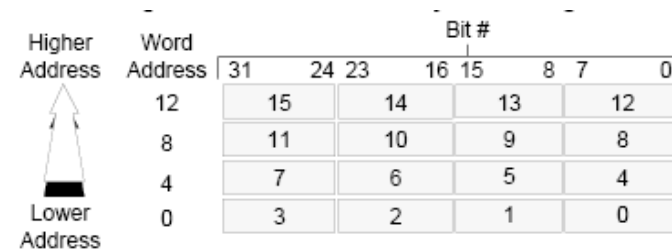
Arquitectura MIPS

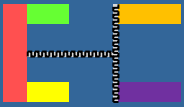
- Arquitectura del MIPS: máquina RISC de 32 bits.
- Procesador MIPS = CPU + coprocesadores auxiliares.
- Coprocesador 0 = excepciones, interrupciones y sistema de memoria virtual.
- Coprocesador 1 = FPU (Unidad de punto flotante).





- Se utiliza direccionamiento por byte (2^{32} bytes).
- La palabra es de 4 bytes (32 bits), por lo que direcciona 2^{30} palabras.
- Los objetos deben estar alineados en direcciones que sean múltiplo de su tamaño.

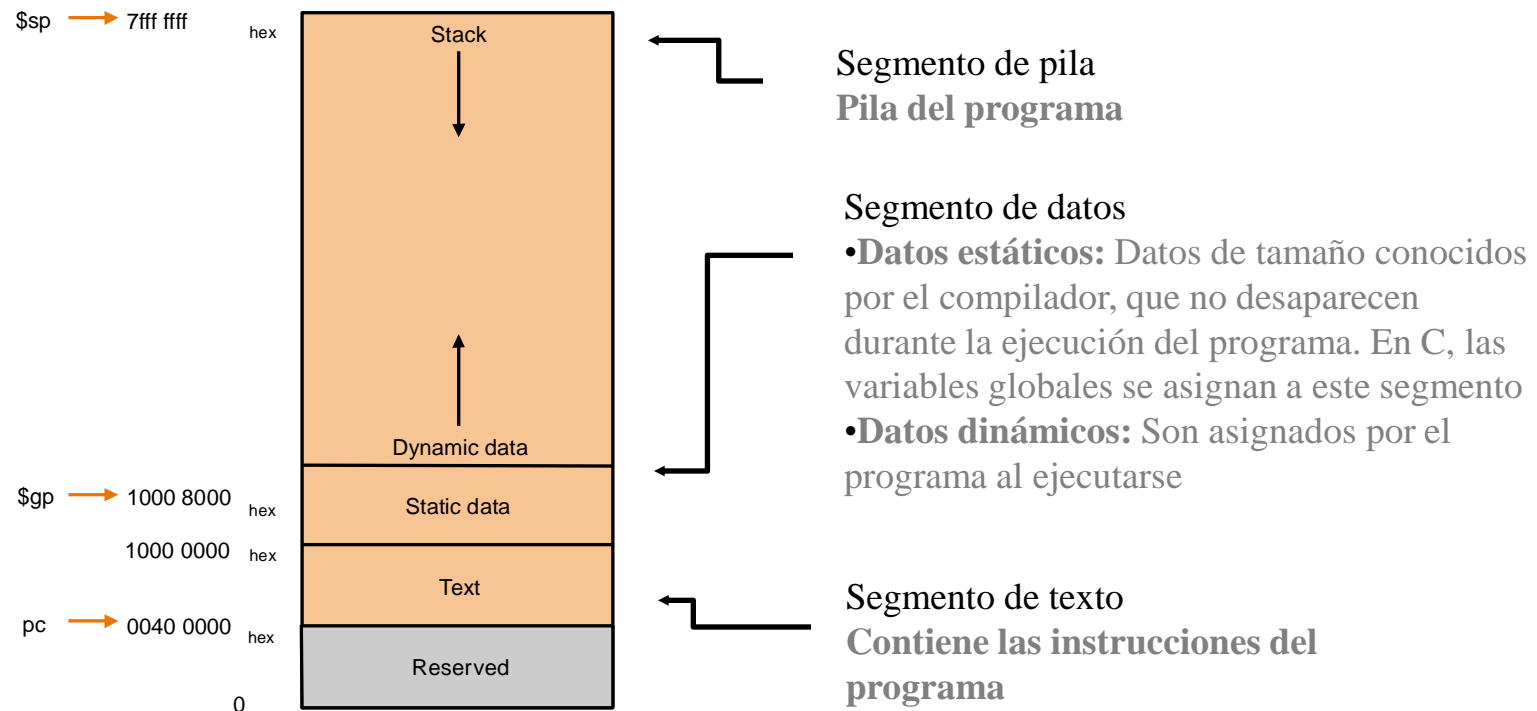


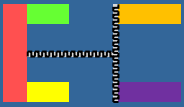


Mapa de memoria

Arquitectura MIPS

- Los segmentos de pila y datos son expandibles dinámicamente.
- Están tan distantes como sea posible y pueden crecer para utilizar el espacio completo de direcciones del programa.



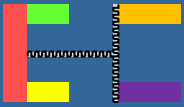


Uso de los registros

Arquitectura MIPS

- Hay 32 registros con el siguiente convenio de uso:

Nombre	Número del registro	Utilización
\$zero	0	El valor constante 0
\$at	1	reservado para el ensamblador
\$v0-\$v1	2-3	valores para resultados y evaluación de expresiones
\$a0-\$a3	4-7	argumentos a rutina (resto argumentos, a pila)
\$t0-\$t7	8-15	temporales (no preservados a través de llamada, guardar invocador)
\$s0-\$s7	16-23	guardado temporalmente (preservado a través de llamada, guardar invocador)
\$t8-\$t9	24-25	más temporales (no preservados a través de llamada, guardar invocador)
\$k0,\$k1	26,27	Reservados para el núcleo del sistema operativo
\$gp	28	puntero global, apunta a la mitad de un bloque de 64k en seg. Datos estáticos
\$sp	29	puntero de pila. Apunta a la primera posición libre de la pila
\$fp	30	puntero de encuadre
\$ra	31	Dirección de retorno usado en llamadas a procedimientos)



Formato de instrucciones

Ensamblador

- ⦿ Todas las instrucciones tienen 32 bits.
- ⦿ Los campos de una instrucción son los siguientes:

Campo	Bits	Descripción
opcode	6	Código de la operación
rd	5	Registro de destino
rs	5	Registro fuente
rt	5	Registro auxiliar
immediate	16	Para operaciones y desplazamientos
instr_index	26	Para saltos
sa	5	Desplazamiento
function	6	Especifica funciones junto con <i>opcode</i>



rd, rs, rt
Add \$t0, \$t1, \$t2

opcode	rs	rt	rd	sa	function
--------	----	----	----	----	----------

5 bits

5 bits

5 bits

5 bits

6 bits

Dirección
del primer
operando fuente

Dirección
del segundo
operando fuente

Dirección
del operando
destino

Cantidad de desplazamiento

Función,
selecciona la
variante de operación
a realizar

opcode	rs	rt	immediate
--------	----	----	-----------

6 bits

5 bits

5 bits

16 bits

rt, rs

```
lw $t0, dir($t1)
```

sw \$t0, dir(\$t1)

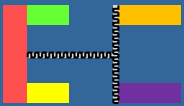
Dato inmediato
o desplazamiento

Dirección
de memoria

opcode	instr_index
--------	-------------

6 bits

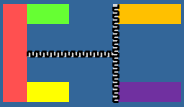
26 bits



Instrucciones aritméticas

Ensamblador

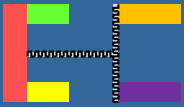
Instrucción	Ejemplo	Significado	Comentarios
suma	add \$1,\$2,\$3	$\$1 \leftarrow \$2 + \$3$	3 operandos; posible excepción
suma inmediata	addi \$1,\$2,100	$\$1 \leftarrow \$2 + 100$	+ constante; posible excepción
suma sin signo	addu \$1,\$2,\$3	$\$1 \leftarrow \$2 + \$3$	3 operandos; no excepción
suma inmediata sin signo	addiu \$1,\$2,100	$\$1 \leftarrow \$2 + 100$	+ constante; no excepción
resta	sub \$1,\$2,\$3	$\$1 \leftarrow \$2 - \$3$	3 operandos; posible excepción
resta sin signo	subu \$1,\$2,\$3	$\$1 \leftarrow \$2 - \$3$	3 operandos; no excepción
multiplicación	mult \$2,\$3	$Hi, Lo \leftarrow \$2 \times \3	producto con signo de 64-bit
multiplicación sin signo	multu \$2,\$3	$Hi, Lo \leftarrow \$2 \times \3	producto sin signo de 64-bit
división	div \$2,\$3	$Lo \leftarrow \$2 \div \$3,$ $Hi \leftarrow \$2 \bmod \3	Lo = cociente, Hi = resto
división sin signo	divu \$2,\$3	$Lo \leftarrow \$2 \div \$3,$ $Hi \leftarrow \$2 \bmod \3	Lo = cociente, Hi = resto



Instrucciones lógicas

Ensamblador

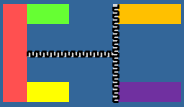
Instrucción	Ejemplo	Significado	Comentarios
and	and \$1,\$2,\$3	$\$1 \leftarrow \$2 \& \$3$	3 registros operandos; AND lógica
or	or \$1,\$2,\$3	$\$1 \leftarrow \$2 \mid \$3$	3 registros operandos; OR lógica
xor	xor \$1,\$2,\$3	$\$1 \leftarrow \$2 \oplus \$3$	3 registros operandos; XOR lógica
nor	nor \$1,\$2,\$3	$\$1 \leftarrow \sim(\$2 \mid \$3)$	3 registros operandos; NOR lógica
not	not \$1,\$2	$\$1 \leftarrow \sim \2	2 registros operandos; NOT lógica (pseudoinstrucción)
and inmediata	andi \$1,\$2,10	$\$1 \leftarrow \$2 \& 10$	AND lógica registro y constante
or inmediata	ori \$1,\$2,10	$\$1 \leftarrow \$2 \mid 10$	OR lógica registro y constante
xor inmediata	xori \$1, \$2,10	$\$1 \leftarrow \sim \$2 \& \sim 10$	XOR lógica registro constante



Instrucciones de desplazamiento

Ensamblador

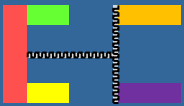
Instrucción	Ejemplo	Significado	Comentarios
despl. izq. lógico	sll \$1,\$2,10	$\$1 \leftarrow \$2 \ll 10$	desplazamiento lógico a la izquierda por constante
despl. izq. lógico	sllv \$1,\$2,\$3	$\$1 \leftarrow \$2 \ll \$3$	desplazamiento lógico a la izquierda por variable
despl. dcha. lógico	srl \$1,\$2,10	$\$1 \leftarrow \$2 \gg 10$	desplazamiento lógico a la derecha por constante
despl. dcha. lógico	srlv \$1,\$2, \$3	$\$1 \leftarrow \$2 \gg \$3$	desplazamiento lógico a la derecha por variable
despl. dcha. arit.	sra \$1,\$2,10	$\$1 \leftarrow \$2 \gg 10$	desplazamiento aritmético a la derecha por constante
despl. dcha. arit.	srav \$1,\$2, \$3	$\$1 \leftarrow \$2 \gg \$3$	desplazamiento aritmético a la derecha por variable



Instrucciones de transferencia

Ensamblador

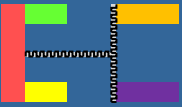
Instrucción	Ejemplo	Significado	Comentarios
load word	lw \$1, 30(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+30]$	cargar palabra
load half	lh \$1, 40(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+40]$	cargar media palabra
load half unsigned	lhu \$1, 50(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+50]$	cargar media palabra sin signo
load byte	lb \$1, 60(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+60]$	cargar byte
load byte unsigned	lbu \$1, 70(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+70]$	cargar byte sin signo
load upper immediate	lui \$1, 123	$\$1 \leftarrow 123\ 0000000000000000$	cargar inmediato superior (carga 16 bits + desplaza a la izquierda 16 bits)
store word	sw \$1, 35(\$2)	$\text{Memoria}[\$2+35] \leftarrow \1	almacenar palabra
store half	sh \$1, 45(\$2)	$\text{Memoria}[\$2+45] \leftarrow \1	almacenar media palabra
store byte	sb \$1, 55(\$2)	$\text{Memoria}[\$2+55] \leftarrow \1	almacenar byte
move from hi	mfhi \$1	$\$1 \leftarrow \text{HI}$	mueve HI a registro
move from lo	mflo \$1	$\$1 \leftarrow \text{LO}$	mueve LO a registro



Instrucciones de comparación y salto

Ensamblador

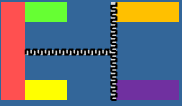
Instrucción	Ejemplo	Significado	Comentarios
menor que	slt \$1,\$2,\$3	if (\$2 < \$3) then \$1 ← 1; else \$1 ← 0	compara menor que; complemento a 2
menor que inmediato	slti \$1,\$2,100	if (\$2 < 100) then \$1 ← 1; else \$1 ← 0	compara < constante; complemento a 2
menor que sin signo	sltu \$1,\$2,\$3	if (\$2 < \$3) then \$1 ← 1; else \$1 ← 0	compara menor que; números enteros
menor que inmediato sin signo	sltiu \$1,\$2,100	if (\$2 < 100) then \$1 ← 1; else \$1 ← 0	compara < constante; números enteros
saltar si igual	beq \$1,\$2,100	if (\$1 == \$2) then PC ← PC + 4 + 100	test igual; salto relativo al PC
saltar si no igual	bne \$1,\$2,100	if (\$1 != \$2) then PC ← PC + 4 + 100	test no igual; salto relativo al PC
salto incondicional	j 10000	PC ← 10000	salta a la instrucción inmediata
salto registro	jr \$31	PC ← \$31	salta a instrucción del registro
salta y enlaza	jal 10000	\$31 ← PC + 4; PC ← 10000	para llamada a un procedimiento



Convenios de llamadas a procedimientos

Ensamblador

- ⦿ Por convenio se utilizan los registros de esta forma:
 - ⦿ \$a0...\$a3: argumentos a pasar a la subrutina.
 - ⦿ \$v0...\$v1: registro para la devolución de valores.
 - ⦿ \$ra: dirección de retorno al punto de origen.
 - ⦿ \$t0...\$t9: registros temporales no preservados por el procedimiento llamado.
 - ⦿ \$s0...\$s9: registros que han de preservarse en la llamada.
- ⦿ Instrucción de llamada a procedimiento: *jal dirección*
- ⦿ Instrucción de retorno del procedimiento: *jr \$ra*
- ⦿ Si hay más de 4 argumentos, se guardan en la pila.



Pasos de llamadas a procedimientos

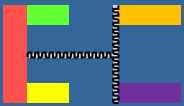
Ensamblador

⦿ Procedimiento que llama:

1. Salvar los registros no preservados en la llamada.
2. Pasar los argumentos con \$a0...\$a3, y el resto en la pila.
3. Ejecutar llamada con jal.

⦿ Procedimiento llamado:

1. Alojarse la pila (nuevo valor a \$sp).
2. Salvar registros \$s0..\$7, \$fp y \$ra.
3. Establecer \$fp.
4. Devolver valores en \$v0..\$v1 si es una función.
5. Restaurar registros preservados.
6. Actualizar \$sp.
7. Volver con jal \$ra.



Ejemplo de llamada a procedimiento

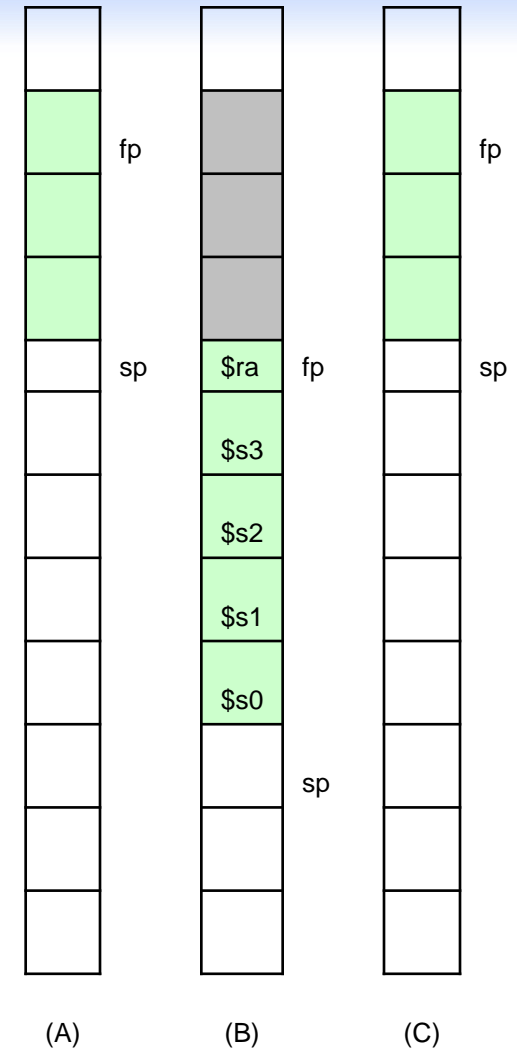
Ensamblador

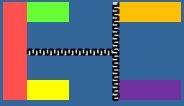
```
principal:                                # Pila en estado (A)
    subu $sp, $sp, 20                     # reservamos espacio en la pila para 5 registros
    sw $ra, 16($sp)                       # salvamos $ra en la pila
    sw $s3, 12($sp)                      # salvamos $s3 en la pila
    sw $s2, 8($sp)                       # salvamos $s2 en la pila
    sw $s1, 4($sp)                       # salvamos $s1 en la pila
    sw $s0, 0($sp)                       # salvamos $s0 en la pila
```

```
    .....
    move $a0, $s2                        # primer parámetro para "otra"
    move $a1, $s1                        # segundo parámetro para "otra"
    jal otra                             # llamada al procedimiento
    .....                               # Pila es estado (C)
```

otra:

```
    .....
Salida de otra:                          # Pila en estado (B)
    lw $s0, 0($sp)                       # restauramos $s0 de la pila
    lw $s1, 4($sp)                       # restauramos $s1 de la pila
    lw $s2, 8($sp)                       # restauramos $s2 de la pila
    lw $s3, 12($sp)                     # restauramos $s3 de la pila
    lw $ra, 16($sp)                     # restauramos $ra de la pila
    addi $sp, $sp, 20                   # restauramos el puntero de pila
    jr $ra                             # volvemos a la rutina que llamo, a "principal"
```

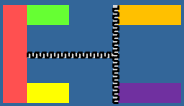




Sintaxis del ensamblador

Ensamblador

- ⦿ Cada línea puede contener como máximo una sentencia.
- ⦿ **Identificadores:** Caracteres alfanuméricos, guiones bajos (_) y puntos (.). No puede comenzar por un número. No permite palabras reservadas.
- ⦿ **Comentarios:** comienzan con el carácter “#” y van al final de la línea.
- ⦿ **Etiquetas:** identificadores seguidos del carácter “:”. Van al principio de la línea y referencian a la posición de memoria.
- ⦿ **Directivas:** indican al ensamblador cómo tiene que traducir un programa, pero no se traducen en instrucciones de máquina.
- ⦿ Los **números** se consideran en base 10. Si van precedidos de “0x”, se interpretan en hexadecimal.
- ⦿ Las **cadenas** de caracteres van entre comillas dobles (“”).
- ⦿ **Especiales:** Salto de línea (\n) , tabulador (\t) y comilla (\”).

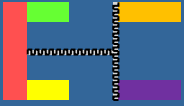


Directivas del ensamblador

Ensamblador

- Comienzan por un punto (.). Indican al ensamblador cómo traducir el programa, pero no se traducen en instrucciones máquina.

Directiva	Descripción
.align n	Alinea el siguiente dato sobre un límite de 2n byte
.ascii "cadena"	Almacena los caracteres en memoria, no termina con carácter nulo.
.asciiz "cadena"	Almacena los caracteres en memoria, termina con carácter nulo.
.byte b1,..bn	Almacena n cantidades de 8 bits en posiciones consecutivas de memoria.
.data <dirección>	Los elementos siguiente son almacenados en el segmento de datos. Si está presente el argumento <dirección>, los datos se almacenan a partir de esa posición.
.double d1,..dn	Almacena n números de doble precisión y coma flotante (64 bits) en posiciones consecutivas de la memoria.
.extern sym size	Declara que el dato almacenado en sym ocupa size bytes y es global. El ensamblador lo pone en parte del segmento de datos fácilmente accesible via \$gp.
.float f1,..fn	Almacena n números en simple precisión y coma flotante (32 bits) en posiciones consecutivas de la memoria.
.globl sym	Declara sym como global: se puede referenciar desde otros archivos.
.half h1,..hn	Almacena n cantidades de 16 bits en posiciones consecutivas de memoria.
.kdata <dirección>	Los elementos siguiente son almacenados en el segmento de datos. del núcleo. Si está presente el argumento <dirección>, los datos se almacenan a partir de esa posición.
.ktext <dirección>	Define el comienzo del segmento de código del kernel
.set	Asigna variables de ensamblador
.space n	Reserva n bytes de espacio en el segmento de datos.
.text <dirección>	Define el comienzo del segmento de código
.word w1,..wn	Almacena n cantidades de 32 bits en posiciones consecutivas de memoria.

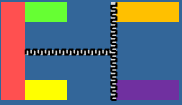


Resumen ensamblador MIPS

Ensamblador

- ⊙ Todas las instrucciones son de 32 bits.
- ⊙ Se trabaja con palabras de 32 bits, pero se direccionan bytes.
- ⊙ Las operaciones aritméticas trabajan con registros.
- ⊙ Se usa la pila para llamar a procedimientos.
- ⊙ Las directivas indican cómo traducir las instrucciones, pero no se traducen en instrucciones de código máquina.



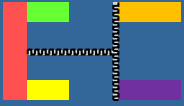


MARS- Características

MARS

- ⊙ MARS es un simulador software que permite ejecutar programas escritos en ensamblador MIPS R2000/R3000.
- ⊙ Es un entorno de desarrollo interactivo que permite la introducción del código en ensamblador.
- ⊙ Permite realizar el seguimiento de la ejecución de los programas y ver el contenido de los registros del procesador.
- ⊙ Se puede descargar desde la página siguiente:
<http://courses.missouristate.edu/kenvollmar/mars/index.htm>
- ⊙ Está implementado en java.





MARS - Pantalla Principal

MARS

- La pantalla principal de MARS se divide en varias partes:

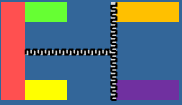
Código

Registros

Mensajes

The screenshot displays the MARS 4.2.2 application window. The main window is divided into three primary sections. The top-left section, labeled 'Código' (Code), contains an assembly file named 'ejemplo1.asm'. The code includes comments in Spanish and assembly instructions for data definition and memory allocation. The top-right section, labeled 'Registros' (Registers), shows a table of MIPS registers, including general-purpose registers (e.g., \$zero, \$at, \$v0) and floating-point registers (e.g., \$f0, \$f1), along with their current values. The bottom section, labeled 'Mensajes' (Messages), is currently empty and contains a 'Clear' button. The window title bar indicates the file path: 'C:\Users\Antonio\Documents\Documentos de Texto\Docencia\Coordinación 2012-2013\EC-GID\Prácticas\Prácticas MIPS\ejemplo1.asm - MARS 4.2'.

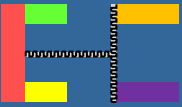
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffc00
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000



🎯 La pantalla principal:

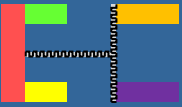
- 🎯 Barra de tareas y barra de acceso rápido. A través de la barra de tareas y los botones de acceso directo accedemos a las diferentes opciones y herramientas que nos ofrece el simulador.
- 🎯 Cuadro de edición (Edit) . Se puede ver y modificar el código fuente del programa. Códigos de colores:
 - 🎯 Registros Rojo
 - 🎯 Instrucciones Azul
 - 🎯 Directivas de ensamblador Rosa
 - 🎯 Cadenas y comentarios Verde
 - 🎯 Otros Negro (sin resaltado)





- ④ La pantalla principal:
 - ④ Cuadro de ejecución (Execute). En la pestaña Execute se puede apreciar un mapeado de la memoria empleada en el segmento de texto y de datos. Para que se cargue la información hay que ensamblar el código fuente.
 - ④ Mensajes (Mars Messages). Aquí se mostrarán los mensajes de estado del simulador Mars. Por ejemplo, se mostrarán mensajes cuando ensemblemos, ejecutemos o finalice nuestro código.
 - ④ Terminal (Run I/O). Es el terminal de la ejecución. Cuando el código se ejecuta e imprime un mensaje se muestra aquí.
 - ④ Registros y valores de los Coprocesadores. En este apartado se puede ver (y alterar) los valores de los distintos registros y valores de los coprocesadores 1 (para valores en coma flotante) y 0 (distintos flags y valores).






🎯 Implementación de un programa

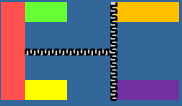
🎯 Escribir el programa ejemplo1.asm


Ejemplo 1: Uso memoria

```
.data # comienza zona de datos
palabra1: .word 15 # decimal
palabra2: .word 0x15 # hexadecimal
medpalabra1: .half 2
medpalabra2: .half 6
byte1: .byte 8
byte2: .byte 5
espacio: .space 4 # Reserva 4 bytes a 0
cadena1: .asciiz "Estructuras de los computadores"
```

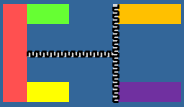
🎯 Grabar el programa:

- 🎯 Menu File->Save AS
- 🎯 Tecleando Ctrl+S
- 🎯 Icono 



- ① Carga de un programa
 - ① Los programas en ensamblador tienen la extensión “.asm”
 - ① Desde Menú File → Open
 - ① Tecleando Ctrl+O
 - ① Icono 

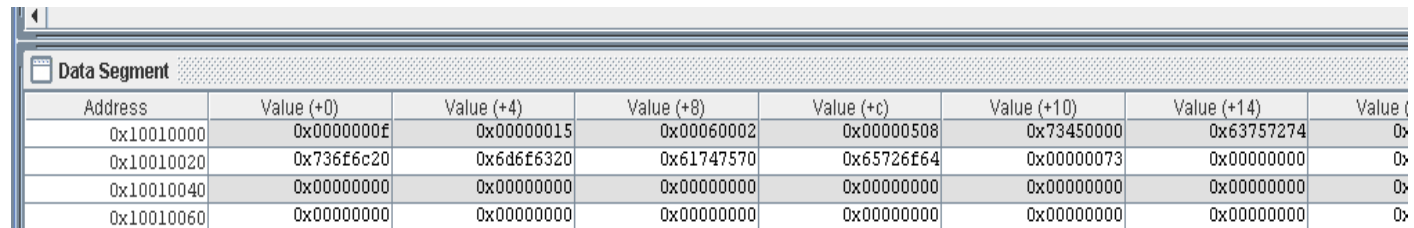




MARS - Ejecución de programas

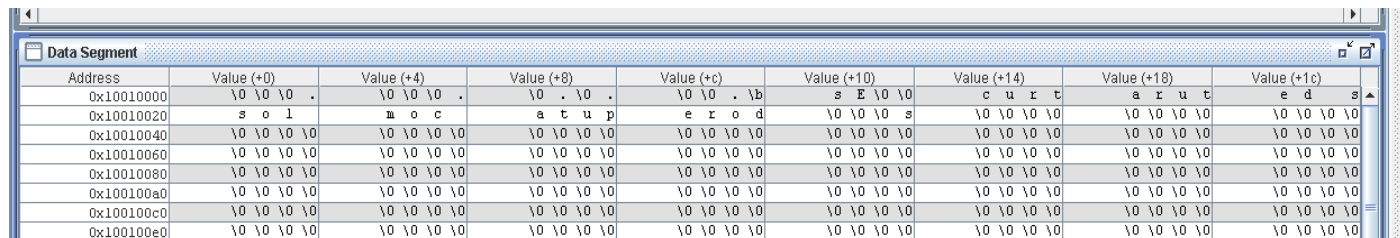
MARS

- 🎯 Ensamblar el programa Menú Run->Assemble
- 🎯 Ejecutar programa Menu Run->Go
- 🎯 Observar cómo se almacenan los datos

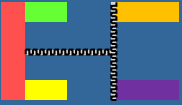


Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x10010000	0x0000000f	0x00000015	0x00060002	0x00000508	0x73450000	0x63757274	0x00000000
0x10010020	0x736f6c20	0x6d6f6320	0x61747570	0x65726f64	0x00000073	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

- 🎯 Ahora en formato ASCII



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	\0 \0 \0 .	\0 \0 \0 .	\0 . \0 .	\0 \0 . \b	s E \0 \0	c u r t	a r u t	e d s
0x10010020	s o l	m o c	a t u p	e r o d	\0 \0 \0 s	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0



Ejemplo 2

🎯 Escribir el siguiente programa (ejemplo2.asm)

```
# Ejemplo 2

        .data          # Segmento de datos
num_a:  .word 2
num_b:  .word 3
resul:  .word 0

        .text          # Segmento de instrucciones
main:   lw $t0,num_a    # La etiqueta main: indica el comienzo del programa
        lw $t1,num_b
        add $t2,$t0,$t1
        sw $t2,resul

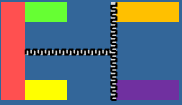
fin:    li $v0, 10      # Fin programa
        syscall
```

🎯 Ejecútalo

🎯 Comprueba el segmento de datos y de texto

🎯 ¿qué valor tiene \$t2? y ¿la dirección de memoria 0x100100008?





Ejemplo 3

🎯 Escribir el siguiente programa (ejemplo3.asm)

```
# Ejemplo 3: Suma componentes de un vector

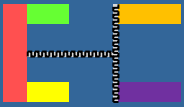
        .data
vector:  .word 6,7,8,9,10,1    # Suma sera 41 dec/29 hex
res:     .space 4

        .text
main:    la $t2,vector #$t2=dirección de vector
        and $t3,$0,$t3 #$t3=0
        li $t0,0 #$t0=0
        li $t1,6 #$t1=5
para:    bgt $t0,$t1,finpara #si $t0>$t1 saltar finpara
        lw $t4,0($t2) #carga elemento vector en $t4
        add $t3,$t4, $t3 #suma los elementos del vector
        addi $t2,$t2, 4 #$t2=$t2+4
        addi $t0,$t0, 1 #$t0=$t0+1
        j para #saltar a bucle
finpara: sw $t3, res($0) #almacenar $t3 en res
fin:     li $v0, 10          # Fin programa
        syscall
```

🎯 Comprueba cómo se hace un bucle

🎯 Ejecútalo y Comprueba el segmento de datos y de texto



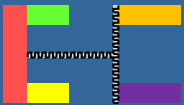


Entrada/Salida de MIPS

Syscall

- ⦿ Conjunto de servicios que permiten interacción del programa con la consola del simulador. Usa la instrucción “syscall”.
- ⦿ $\$v0$: se usa para poner el código de la llamada.
- ⦿ $\$a0 \dots \$a3 / \$f12$: se utilizan para intercambiar los parámetros.

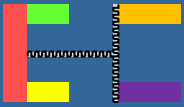
Servicio	Código de llamada	Argumentos	Resultado	Descripción
print_int	1	$\$a0$ = entero		Imprimir en consola
print_float	2	$\$f12$ = flotante		
print_double	3	$\$f12$ = doble		
print_string	4	$\$a0$ = comienzo de cadena		
read_int	5		Entero en $\$v0$	Leer de consola
read_float	6		Flotante en $\$f0$	
read_double	7		Doble en $\$f0$	
read_string	8		$\$a0$ = buffer, $\$a1$ = longitud	
sbrk	9	$\$a0$ = cantidad	Dirección en $\$v0$	Puntero a memoria



Entrada/Salida de MIPS

Syscall

Servicio	Código de llamada	Argumentos	Resultado	Descripción
exit	10			Finaliza ejecución
print_char	11	\$a0 = carácter		Escribe carácter
read_char	12		carácter en \$v0	Lee carácter
open	13	\$a0 = nombre fichero, \$a1= flags, \$a2=modo	descriptor de fichero en \$a0	Operaciones sobre fichero
read	14	\$a0 = descriptor fichero, \$a1= buffer, \$a2=tamaño	número de caracteres leídos en \$a0	
write	15	\$a0 = descriptor fichero, \$a1= buffer, \$a2=tamaño	número de caracteres escritos en \$a0	
close	16	\$a0 = descriptor fichero		
exit2	17	\$a0 = resultado		Finaliza ejecución



Ejemplo 4

🎯 Implementación de un programa

🎯 Escribir el programa ejemplo4.asm

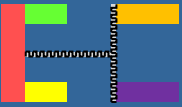
```
# Ejemplo 4: Leer un entero por teclado

.data
dir: .asciiz "Introduce el entero: "
.align 2
entero: .word 0

.text
main: li $v0, 4 # código de imprimir cadena
      la $a0, dir # dirección de la cadena
      syscall # Llamada al sistema
      li $v0, 5 # código de leer entero
      syscall # Llamada al sistema
      sw $v0, entero
fin: li $v0, 10      # Fin programa
     syscall
```

🎯 Ejecútalo

🎯 Comprobar cómo se lee e imprime en la consola



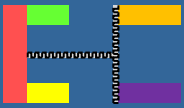
Ejemplo sencillo

Ejemplo sencillo

- Se muestra un ejemplo sencillo para ver el funcionamiento paso a paso:

```
.data 0x10000000
var1:  .word 0x88884444
      .word 0x11110000
var2:  .half 0x2222
      .half 0x4444
      .byte -8,1,2,3
fincad: .ascii "\n Fin del ejemplo."

.text
main:
      la $7,0x10000000
      lw $8,0($7)
      lw $9,4($7)
      lw $10,8($7)
      move $11,$8
      sw $11,4($7)
      add $12,$10,$9
      sub $13,$10,$9
      divu $8,$10
      lb $14,12($7)
      abs $15,$14
      mult $15,$10
      li $v0,4
      la $a0,fincad
      syscall
fin:   li $v0, 10    # Fin programa
      syscall
```



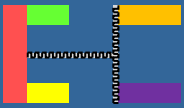
Ejemplo sencillo - Carga del programa

Ejemplo sencillo

Se comprueban el código y los datos cargados.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10000000	0x88884444	0x88884444	0x44442222	0x030201f8	0x6946200a	0x6564206e	0x6a65206c	0x6c706d65
0x10000020	0x00002e6f	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10000040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10000060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10000080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10000000	. . D D	. . D D	D D " "	i F \n	e d n	j e l	l p m e
0x10000020	\0 \0 . o	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10000040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10000060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10000080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100000a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100000c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0



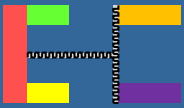
Ejemplo sencillo - Carga del programa

Ejemplo sencillo

- Se comprueban el código y los datos cargados.

Codificación de la instrucción

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011000	lui \$1,0x00001000	11: la \$7,0x10000000
	0x00400004	0x34270000	ori \$7,\$1,0x00000000	
	0x00400008	0x8ce80000	lw \$8,0x00000000(\$7)	12: lw \$8,0(\$7)
	0x0040000c	0x8ce90004	lw \$9,0x00000004(\$7)	13: lw \$9,4(\$7)
	0x00400010	0x8cea0008	lw \$10,0x00000008(\$7)	14: lw \$10,8(\$7)
	0x00400014	0x00085821	addu \$11,\$0,\$8	15: move \$11,\$8
	0x00400018	0xaceb0004	sw \$11,0x00000004(\$7)	16: sw \$11,4(\$7)
	0x0040001c	0x01496020	add \$12,\$10,\$9	17: add \$12,\$10,\$9
	0x00400020	0x01496822	sub \$13,\$10,\$9	18: sub \$13,\$10,\$9
	0x00400024	0x010a001b	divu \$8,\$10	19: divu \$8,\$10
	0x00400028	0x80ee000c	lb \$14,0x0000000c(\$7)	20: lb \$14,12(\$7)
	0x0040002c	0x000e0fc3	sra \$1,\$14,0x0000001f	21: abs \$15,\$14
	0x00400030	0x002e7826	xor \$15,\$1,\$14	
	0x00400034	0x01e17823	subu \$15,\$15,\$1	
	0x00400038	0x01ea0018	mult \$15,\$10	22: mult \$15,\$10
	0x0040003c	0x24020004	addiu \$2,\$0,0x00000004	23: li \$v0,4



Ejemplo sencillo - Ejecución paso a paso

Ejemplo sencillo



Se va ejecutando el programa paso a paso con el botón



C:\Users\Antonio\Documents\Documentos de Texto\Docencia\Coordinación 2012-2013\EC-GDI\Prácticas\Prácticas MIPS\ejemplosencillo.asm - MARS 4.2

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

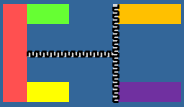
Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011000	lui \$1,0x00001000	11: la \$7,0x10000000
<input type="checkbox"/>	0x00400004	0x34270000	ori \$7,\$1,0x00000000	
<input type="checkbox"/>	0x00400008	0x8ce80000	lw \$8,0(\$7)	12: lw \$8,0(\$7)
<input type="checkbox"/>	0x0040000c	0x8ce90004	lw \$9,0x00000004(\$7)	13: lw \$9,4(\$7)
<input type="checkbox"/>	0x00400010	0x8cea0008	lw \$10,0x00000008(\$7)	14: lw \$10,8(\$7)
<input type="checkbox"/>	0x00400014	0x00085821	addu \$11,\$0,\$8	15: move \$11,\$8
<input type="checkbox"/>	0x00400018	0xaceb0004	sw \$11,0x00000004(\$7)	16: sw \$11,4(\$7)
<input type="checkbox"/>	0x0040001c	0x01496020	add \$12,\$10,\$9	17: add \$12,\$10,\$9
<input type="checkbox"/>	0x00400020	0x01496822	sub \$13,\$10,\$9	18: sub \$13,\$10,\$9
<input type="checkbox"/>	0x00400024	0x010a001b	divu \$8,\$10	19: divu \$8,\$10
<input type="checkbox"/>	0x00400028	0x80ee000c	lb \$14,0x0000000c(\$7)	20: lb \$14,12(\$7)
<input checked="" type="checkbox"/>	0x0040002c	0x000e0fc3	sra \$1,\$14,0x0000001f	21: abs \$15,\$14
<input type="checkbox"/>	0x00400030	0x002e7826	xor \$15,\$1,\$14	
<input type="checkbox"/>	0x00400034	0x01e17823	subu \$15,\$15,\$1	
<input type="checkbox"/>	0x00400038	0x01ea0018	mult \$15,\$10	22: mult \$15,\$10
<input type="checkbox"/>	0x0040003c	0x24020004	addiu \$2,\$0,0x00000004	23: li \$v0,4

Data Segment

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x10000000
\$t0	8	0x88884444
\$t1	9	0x11110000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000



Ejemplo sencillo - Fin de ejecución

Ejemplo sencillo

- Al finalizar puede verse el estado de los registros y la salida por consola.

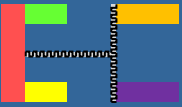
The screenshot displays the MARS MIPS simulator interface. The 'Registers' window is open, showing the state of 32 registers. The 'Mars Messages' window is also open, displaying the console output.

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10000000
\$v0	2	0x00000004
\$v1	3	0x00000000
\$a0	4	0x10000010
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x10000000
\$t0	8	0x88884444
\$t1	9	0x11110000
\$t2	10	0x44442222
\$t3	11	0x88884444
\$t4	12	0x55552222
\$t5	13	0x33332222
\$t6	14	0xffffffff
\$t7	15	0x00000008
\$s0	16	0x00000000
\$s1	17	
\$s2	18	
\$s3	19	
\$s4	20	
\$s5	21	
\$s6	22	
\$s7	23	
\$t8	24	
\$t9	25	
\$k0	26	
\$k1	27	
\$gp	28	
\$sp	29	
\$fp	30	
\$ra	31	
pc		
hi		
lo		0x22211110

The 'Mars Messages' window shows the following output:

```
Fin del ejemplo.  
-- program is finished running (dropped off bottom) --
```

A 'Clear' button is visible below the message window.



Ejercicio propuesto

Práctica

Dado el siguiente programa escrito en el lenguaje ensamblador del MIPS R2000:

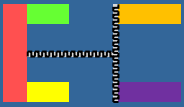
```
.data 0x10000000
num1:      .word 0x11112222
           .word 0x88884444
           .half 0x1234
           .half 0x4321
           .byte -8
           .byte 123

.text
main:
    la $7,0x10000000
    lw $8,0($7)
    lw $9,4($7)
    lw $10,8($7)
    move $11,$8
    lh $12,8($7)
    lb $13,8($7)
    sw $11,16($7)
    add $14,$11,$12
    add $15,$9,$11
    sub $16,$10,$11
    sub $17,$12,$11
    divu $8,$12
    mult $8,$13
    mfhi $18
    xor $19,$10,$11
    sll $19,$19,4

bucle:
    beqz $18,fin
    add $13,$13,$13
    addi $18,$18,-1
    j bucle

fin:
    li $v0, 10      # Fin programa
    syscall
```





Ejercicio propuesto

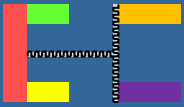
Práctica

Se solicita realizar las siguientes tareas:

1. Cargar el programa en el simulador MARS y comprobar su funcionamiento paso a paso.
2. Mostrar el contenido del segmento de datos de memoria antes y después de la ejecución del programa como se muestra en el ejemplo:

Dirección de memoria	Contenido INICIAL	Contenido FINAL
0x10000000	0x11112222	0x11112222





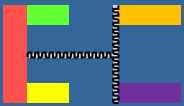
Ejercicio propuesto

Práctica

3. Indicar el contenido de los registros antes y después de la ejecución del programa:

Registro	Contenido INICIAL	Contenido FINAL
\$7	0x0	0x10000000
\$8		
\$9		
\$10		
\$11		
\$12		
\$13		
\$14		
\$15		
\$16		
\$17		
\$18		
\$19		





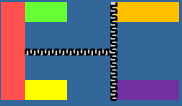
Ejercicio propuesto

Práctica

4. Codificar las siguientes instrucciones del programa en código máquina, indicando el tipo de instrucción (R, I, J) y separando cada uno de sus campos como se muestra en el ejemplo:

Instrucción ensamblador	Instrucción código máquina	Tipo	opcode (6)	rs (5)	rt (5)	rd (5)	immediate (16)	instr_index (26)	sa (5)	function (6)
lw \$10,8(\$7)	0x8CEA0008	I	100011	00111	01010	-	0000000000001000	-	-	-
add \$14,\$11,\$12										
divu \$8,\$12										
sll \$19,\$19,4										
addi \$18,\$18,-1										
j bucle										

5. Modificar el programa para que al llegar a la etiqueta “fin” muestre por consola el nombre del alumno mediante el uso de la instrucción “syscall”.



Ejercicio propuesto

Práctica

- ⊙ La realización de la práctica consiste en la contestación de cada uno de los 5 apartados del ejercicio propuesto.
- ⊙ Se debe entregar una memoria con la respuesta a cada una de las cuestiones planteadas.
- ⊙ Se debe entregar también el código en ensamblador del programa con las modificaciones solicitadas.

