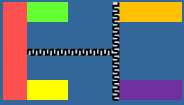


# PRÁCTICA 3.1. INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR - MARS

dtic



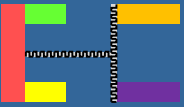


# PRÁCTICA 3.1. INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR- MARS

## Índice

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



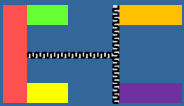


# Introducción MIPS

## Introducción MIPS

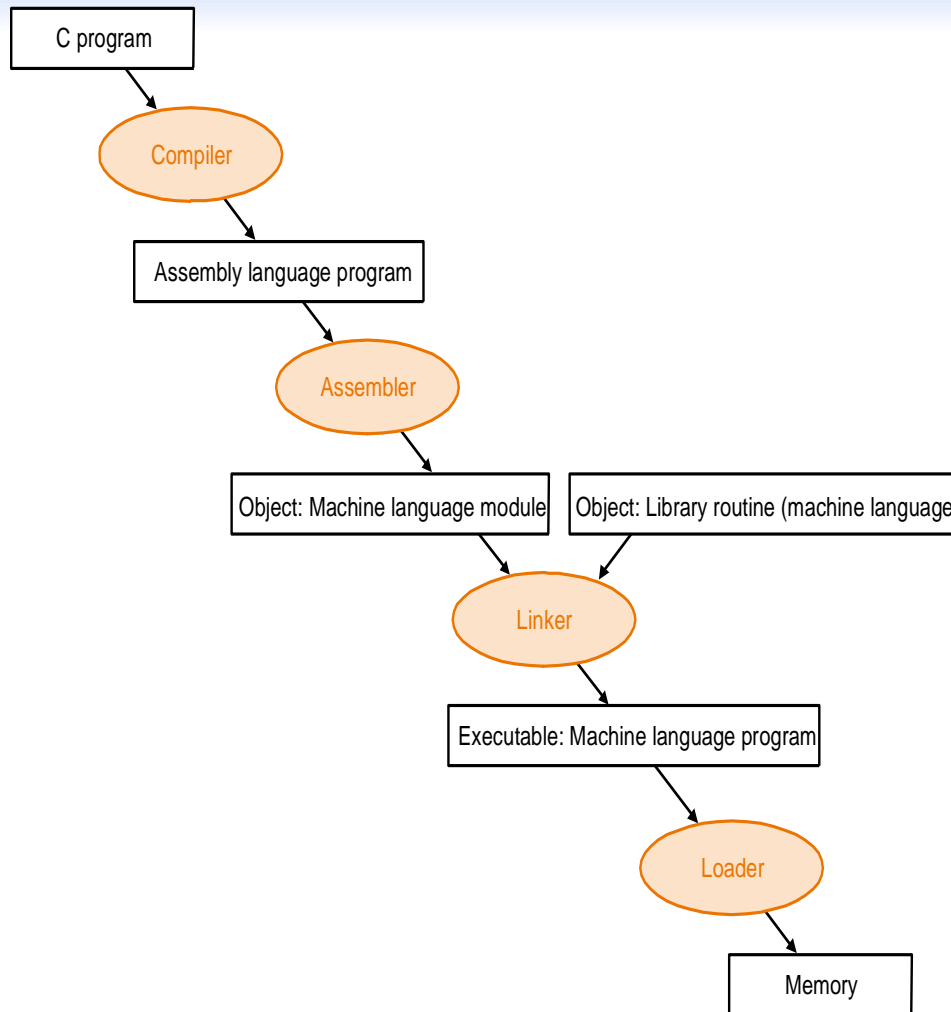
- ⊙ MIPS, acrónimo de Microprocessor without Interlocked Pipeline Stages.
- ⊙ Diseñado por John L. Hennessy en 1981 en la Universidad de Stanford y fundó la compañía MIPS Computer Systems Inc.
- ⊙ Es un diseño RISC
- ⊙ Varias versiones: 32 bits (**R2000** y R3000), 64 bits (R4000)
- ⊙ Aplicaciones
 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





# Introducción MIPS

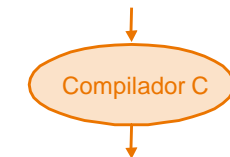
## Jerarquía de traducción



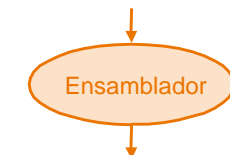
Programa  
en lenguaje  
de alto  
nivel  
(en C)

```
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
```



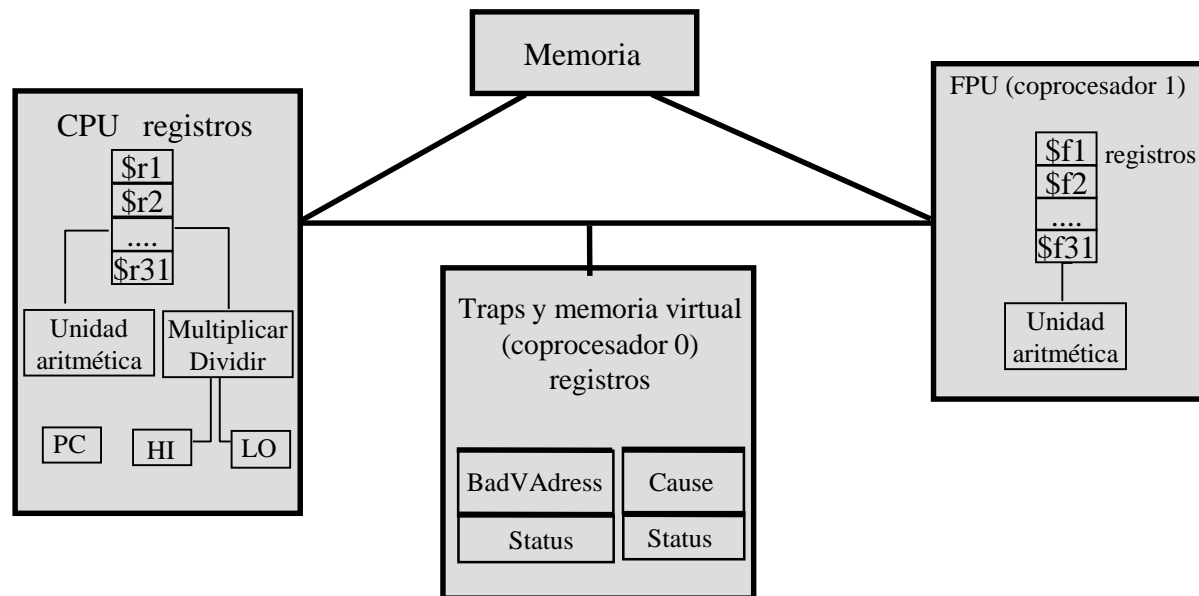
Programa  
En lenguaje  
Máquina  
(para MIPS)

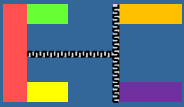
```
000000001010000100000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
10101100111100100000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

# Arquitectura MIPS R2000

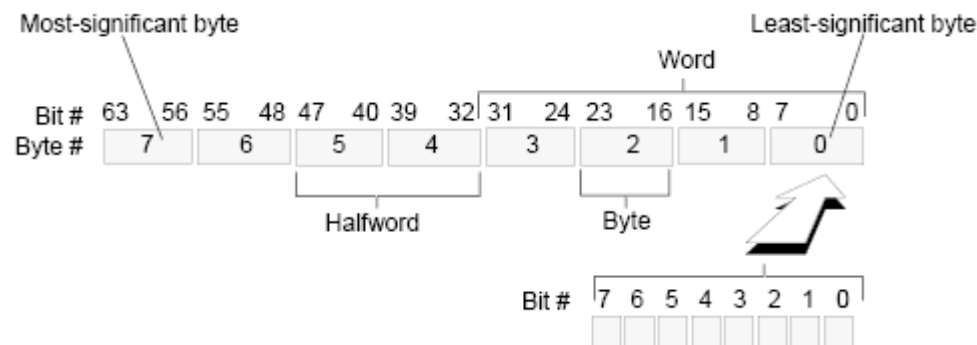
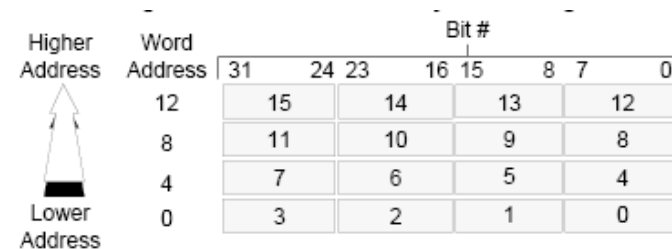
## Arquitectura MIPS

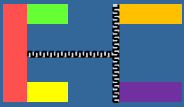
- Arquitectura del MIPS: máquina RISC de 32 bits.
- Procesador MIPS = CPU + coprocesadores auxiliares.
- Coprocesador 0 = excepciones, interrupciones y sistema de memoria virtual.
- Coprocesador 1 = FPU (Unidad de punto flotante).





- 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.

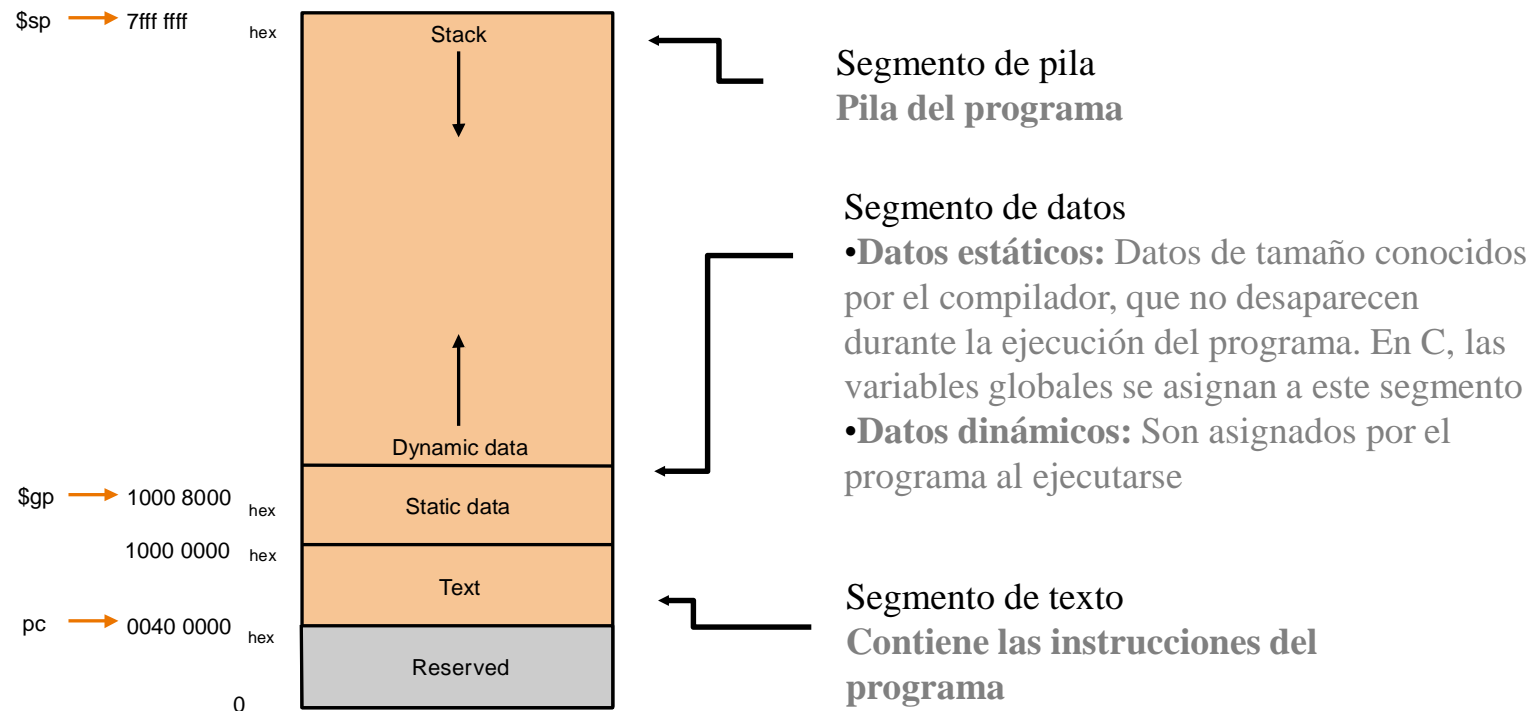


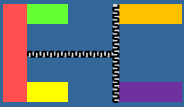


# Mapa de memoria

## 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.





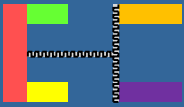
# Uso de los registros

## 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)



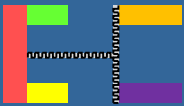


# Formato de instrucciones

## 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>



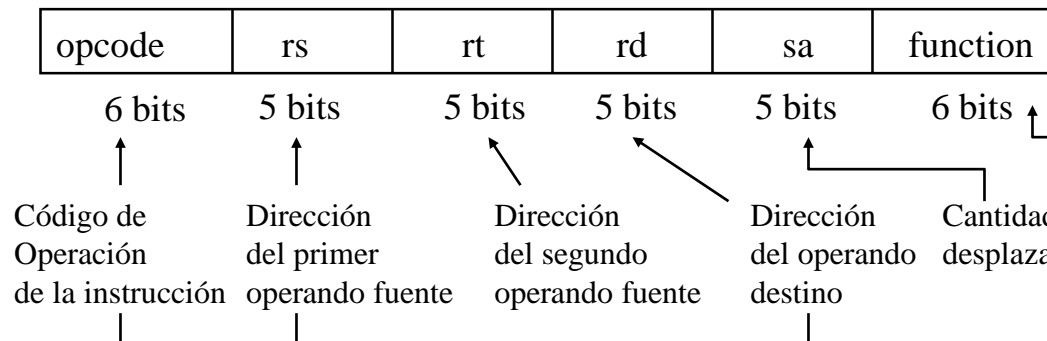
# Formato de instrucciones

Ensamblador

Existen 3 tipos de formatos de instrucción diferentes:

## Tipo R

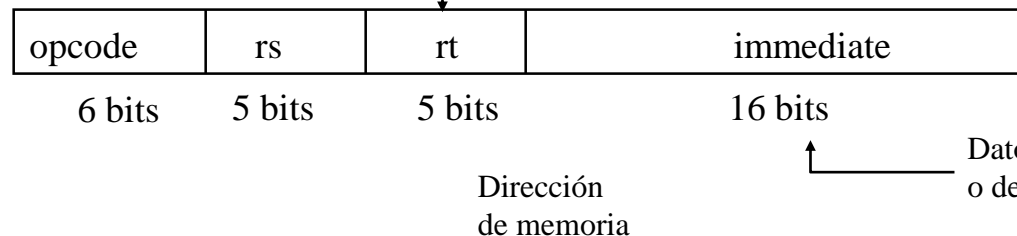
( Register )



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

## Tipo I

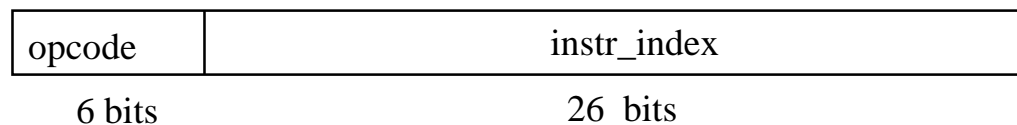
( Immediate )



rt, rs  
**lw \$t0, dir(\$t1)**  
**sw \$t0, dir(\$t1)**

## Tipo J

( Jump )

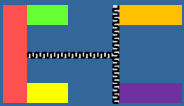




# Instrucciones aritméticas

## Ensamblador

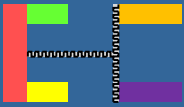
Instrucción	Ejemplo	Significado	Comentarios
suma	add \$1,\$2,\$3	$\$1 \leftarrow \$2 + \$3$	3 operandos; posible excepción
suma inmediata	addi \$1,\$2,100	$\$1 \leftarrow \$2 + 100$	+ constante; posible excepción
suma sin signo	addu \$1,\$2,\$3	$\$1 \leftarrow \$2 + \$3$	3 operandos; no excepción
suma 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

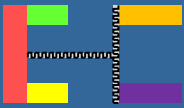
Instrucción	Ejemplo	Significado	Comentarios
and	and \$1,\$2,\$3	$\$1 \leftarrow \$2 \& \$3$	3 registros operandos; AND lógica
or	or \$1,\$2,\$3	$\$1 \leftarrow \$2 \mid \$3$	3 registros operandos; OR lógica
xor	xor \$1,\$2,\$3	$\$1 \leftarrow \$2 \oplus \$3$	3 registros operandos; XOR lógica
nor	nor \$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

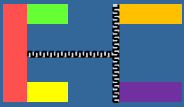
Instrucción	Ejemplo	Significado	Comentarios
despl. izq. lógico	sll \$1,\$2,10	$\$1 \leftarrow \$2 \ll 10$	desplazamiento lógico a la izquierda por constante
despl. izq. lógico	sllv \$1,\$2,\$3	$\$1 \leftarrow \$2 \ll \$3$	desplazamiento lógico a la izquierda por variable
despl. dcha. lógico	srl \$1,\$2,10	$\$1 \leftarrow \$2 \gg 10$	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, carga y almacenamiento

## Ensamblador

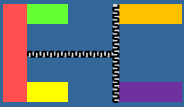
Instrucción	Ejemplo	Significado	Comentarios
cargar palabra	lw \$1, 30(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+30]$	cargar palabra
cargar media palabra	lh \$1, 40(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+40]$	cargar media palabra
cargar media palabra sin signo	lhu \$1, 50(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+50]$	cargar media palabra sin signo
carga byte	lb \$1, 60(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+60]$	cargar byte
larga byte sin signounsigned	lbu \$1, 70(\$2)	$\$1 \leftarrow \text{Memoria}[\$2+70]$	cargar byte sin signo
Carga inmediato superior	lui \$1, 123	$\$1 \leftarrow 123\ 0000000000000000$	cargar inmediato superior (carga 16 bits + desplaza a la izquierda 16 bits)
Carga inmediato	li \$1, 10	$\$1 \leftarrow 10$	Carga el dato inmediato en el registro
Carga dirección	la \$s0,etiqueta	$\$s0 \leftarrow \text{Dirección etiq.}$	Carga en registro la dirección asociada a la etiqueta.
Almacena palabra	sw \$1, 35(\$2)	$\text{Memoria}[\$2+35] \leftarrow \$1$	almacenar palabra
Almacena media palabra	sh \$1, 45(\$2)	$\text{Memoria}[\$2+45] \leftarrow \$1$	almacenar media palabra
Almacena byte	sb \$1, 55(\$2)	$\text{Memoria}[\$2+55] \leftarrow \$1$	almacenar byte
Carga hi en registro	mfhi \$1	$\$1 \leftarrow \text{HI}$	mueve HI a registro
Carga lo en registro	mflo \$1	$\$1 \leftarrow \text{LO}$	mueve LO a registro



# Instrucciones de salto incondicionales

Ensamblador

Instrucción	Ejemplo	Significado	Comentarios
salto incondicional	j 10000	$PC \leftarrow 10000$	salta a la instrucción inmediata
salto registro	jr \$31	$PC \leftarrow \$31$	salta a instrucción del registro
salta y enlaza	jal 10000	$\$31 \leftarrow PC + 4$ ; $PC \leftarrow 10000$	para llamada a un procedimiento

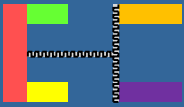


# Instrucciones de saltos condicionales

## Ensamblador

Instrucción	Ejemplo	Significado	Comentarios
Salto si igual	Beq rs, rt, etiqueta	if (\$2 = \$3) then    \$pc ←etiqueta	Salta a etiqueta si rs es igual a rt.
Salto si mayor o igual que cero	bgez rs, etiqueta	if (\$2 >=0) then    \$pc ←etiqueta	Salta a etiqueta si rs es mayor o igual que 0.
Salto si mayor que 0	bgtz rs, etiqueta	if (\$2 >0) then    \$pc ←etiqueta	Salta a etiqueta si rs es mayor que 0.
Salto si menor o igual que cero	blez rs,etiqueta	if (\$2 <=0) then    \$pc ←etiqueta	Salta a etiqueta si rs es menor o igual que 0.
Salto si menor que cero	bltz rs, etiqueta	if (\$2 <0) then    \$pc ←etiqueta	Salto si menor que 0.
Salto si distinto	bne rs, rt, etiqueta	if (\$1!= \$2) then    \$pc ←etiqueta	test no igual; salto relativo al PC
Sato mayor o igual	bge reg1,reg2,eti	if (\$1>= \$2) then    \$pc ←etiqueta	Salta a etiqueta si reg1 es mayor o igual que reg2.
Salto mayor	bgt reg1,reg2,eti	if (\$1> \$2) then    \$pc ←etiqueta	Salta a etiqueta si reg1 es mayor que reg2.
Salto menor o igual	ble reg1,reg2,eti	f (\$1<= \$2) then    \$pc ←etiqueta	Salta a etiqueta si reg1 es mayor o igual que reg2.
Salto menor	blt reg1,reg2,eti	if (\$1< \$2) then    \$pc ←etiqueta	Salta a etiqueta si reg1 es menor que reg2.

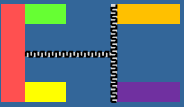




# Instrucciones de comparaciones

## Ensamblador

Instrucción	Ejemplo	Significado	Comentarios
Activa si menor	slt rd, rs, rt	if (\$2 = \$3) then    \$pc ←etiqueta	Pone el registro re a 1 si rs es menor que rt y a 0 en caso contrario.
Activa si menor con inmediato	slti rt, rs, inm	if (\$2 >=0) then    \$pc ←etiqueta	Pone el registro rt a 1 si rs es menor el dato inmediato inm y a 0 en caso contrario.
Activa si igual	seq rdest, rsrc1, rsrc2	if (\$2 >0) then    \$pc ←etiqueta	Pone el registro rdest a 1 si rsrc1 es igual de rsrc2 y a 0 en caso contrario.
Activa si mayor o igual	sge rdest, rsrc1, rsrc2	if (\$2 <=0) then    \$pc ←etiqueta	Pone el registro rdest a 1 si rsrc1 es mayor o igual que rsrc2 y a 0 en caso contrario.
Activa si mayor	sgt rdest, rsrc1, rsrc2	if (\$2 <0) then    \$pc ←etiqueta	Pone el registro rdest a 1 si rsrc1 es mayor que rsrc2 y a 0 en caso contrario.
Activa si menor o igual	sle rdest, rsrc1, rsrc2	if (\$1!= \$2) then    \$pc ←etiqueta	Pone el registro rdest a 1 si rsrc1 es menor o igual que rsrc2 y a 0 en caso contrario.
Activa si no igual	sne rdest, rsrc1, rsrc2	if (\$1>= \$2) then \$1←1 else \$1←0	Pone el registro rdest a 1 si rsrc1 es diferente de rsrc2 y a 0 en caso contrario.

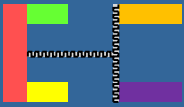


# Convenios de llamadas a procedimientos

## 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

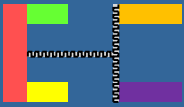
## Ensamblador

### ⦿ 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

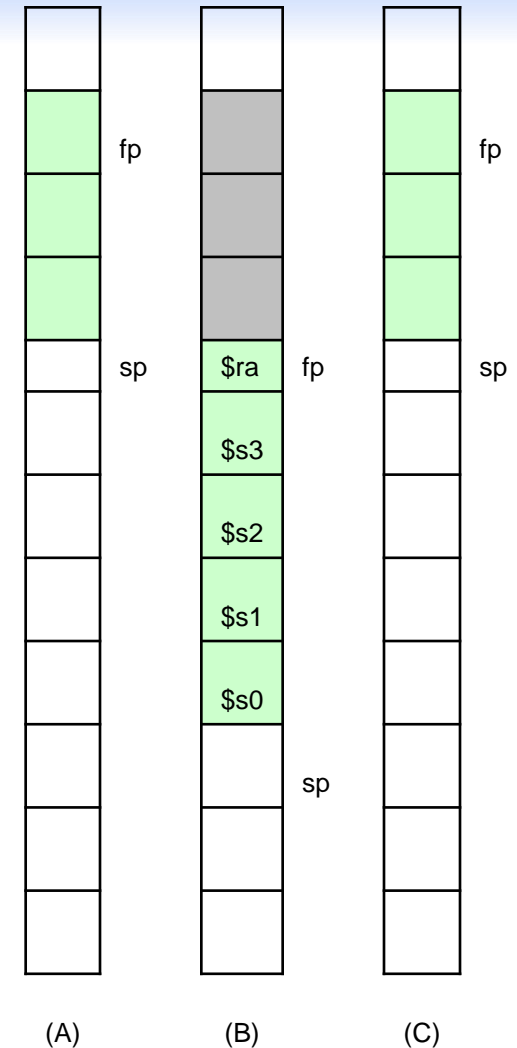
## Ensamblador

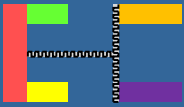
principal:                   # Pila en estado (A)  
    subu \$sp, \$sp, 20       # reservamos espacio en la pila para 5 registros  
    sw \$ra, 16(\$sp)        # salvamos \$ra en la pila  
    sw \$s3, 12(\$sp)        # salvamos \$s3 en la pila  
    sw \$s2, 8(\$sp)         # salvamos \$s2 en la pila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

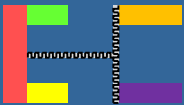




# Sintaxis del ensamblador

## 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 (\”).

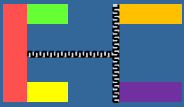


# Directivas del ensamblador

## 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.

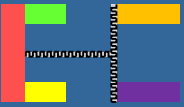


# Resumen ensamblador MIPS

## 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.





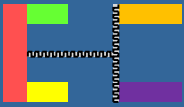
# MARS- Características

## 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:

The screenshot displays the MARS 4.2.2 application window. The main window is divided into several sections:

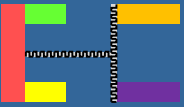
- Code Editor:** Contains assembly code for 'ejemplo1.asm'. It includes comments and instructions like `.data`, `.word`, `.half`, `.byte`, `.space`, and `.ascii`. An arrow labeled 'Código' points to this section.
- Registers Panel:** Located on the right, it shows a table of registers (e.g., \$zero, \$at, \$v0, \$v1, \$a0, \$a1, \$a2, \$a3, \$t0, \$t1, \$t2, \$t3, \$t4, \$t5, \$t6, \$t7, \$s0, \$s1, \$s2, \$s3, \$s4, \$s5, \$s6, \$s7, \$s8, \$s9, \$k0, \$k1, \$gp, \$sp, \$fp, \$ra, \$pc, \$hi, \$lo) and their current values. An arrow labeled 'Registros' points to this panel.
- Mars Messages:** A panel at the bottom left for displaying messages and I/O. An arrow labeled 'Mensajes' points to this panel.

The interface also includes a menu bar (File, Edit, Run, Settings, Tools, Help), a toolbar with various icons, and a status bar at the bottom showing 'Line: 1 Column: 1' and 'Show Line Numbers'.



### 🎯 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)



### 🎯 La pantalla principal:

- 🎯 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.
- 🎯 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.
- 🎯 Terminal (Run I/O). Es el terminal de la ejecución. Cuando el código se ejecuta e imprime un mensaje se muestra aquí.
- 🎯 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:

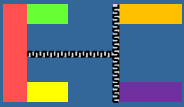
- 🎯 Menu File->Save AS
- 🎯 Tecleando Ctrl+S
- 🎯 Icono 





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

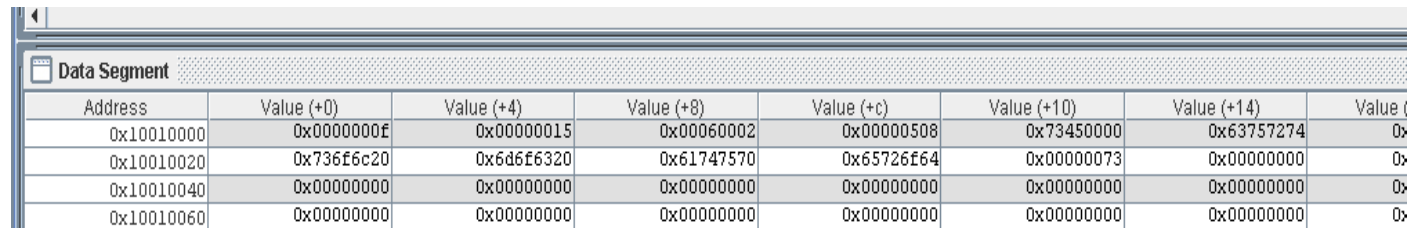




# MARS - Ejecución de programas

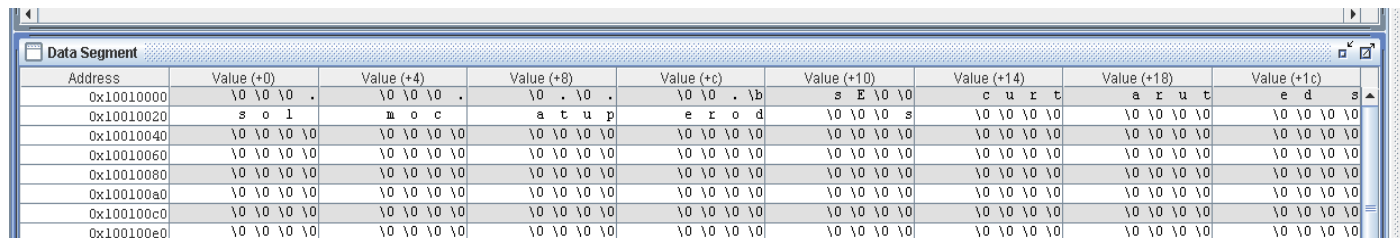
## MARS

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

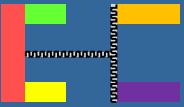


Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)
0x10010000	0x0000000f	0x00000015	0x00060002	0x00000508	0x73450000	0x63757274	0x00000000
0x10010020	0x736f6c20	0x6d6f6320	0x61747570	0x65726f64	0x00000073	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000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

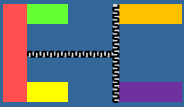
fin:    li $v0, 10       # Fin programa
        syscall
```

### 🎯 Ejecútalo

### 🎯 Comprueba el segmento de datos y de texto

### 🎯 ¿qué valor tiene \$t2? y ¿la dirección de memoria 0x10010008?





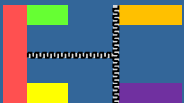
# Entrada/Salida de MIPS

## 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 ... \$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

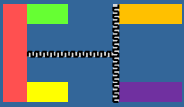




# Entrada/Salida de MIPS

## 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



# Ejercicios propuestos

## Práctica

Se solicita realizar las siguientes ejercicios:

1º.- Diseña un programa en ensamblador que calcule la suma de los valores de un vector compuesto por tres enteros  $V=(10, 7, 9)$  almacenado en la memoria de datos a partir de la posición  $0x10010000$  y almacene el resultado en la dirección  $0x10010100$ .

Ejemplo de vector:

Vector: `.word 6,7,8,9,10,1` # Suma será 41 dec/29 hex

2º.- El mismo programa multiplicará por 8 (sin utilizar la instrucción de multiplicación) el número almacenado en la dirección  $0x10010100$  y lo almacenará en la posición  $0x10010120$ .

3º.- Copia el contenido de memoria de la posición  $0x10010120$  en la posición  $0x1010140$  y modifica el programa para que modifique el valor de los bits 5, 7 y 8. ¿Cuál es el resultado obtenido?



# Ejercicios propuestos

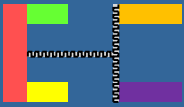
## Práctica

### 4.- Realiza el siguiente programa

```
#EJEMPLO DE RESTA
.data
numero1: .word 0x80000000
numero2: .word 1
.text
main:
lw $t1,numero1($0)
lw $t2,numero2($0)
sub $t1,$t1,$t2
sw $t1, numero1($0)
```

4.1.- Compíllalo y ejecútalo. ¿Qué sucede? Explica por que sucede.

4.2.- Sustituye las instrucciones sub por subu y ejecuta el programa. Comenta los resultados.



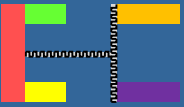
# Ejercicios propuestos

## Práctica

### 5.- Ejecuta el siguiente programa:

```
# Práctica 3.1 Ejer.5  Introducción a MIPS    (Uso de las llamadas al sistema)
.data
string1:      .asciiz "Introduce un número:"
String2:      .asciiz "El número introducido es :\n"
.globl main
.text
main: la $a0,string1      # Mostramos por consola una cadena
    li $v0,4
    syscall
    li $v0,5              # leemos un entero introducido por teclado
    syscall
    move $t0,$v0          # guardamos el valor leído por teclado
    la $a0,string2        # Mostramos por consola el numero introducido
    li $v0,4
    syscall
    move $a0,$t0
    li $v0,1
    syscall
    li $v0,10             # Fin del programa #
    syscall
```

Añade el código necesario al programa anterior para lea 2 números y calcule su valor medio.



# Ejemplo de bucle

## Práctica

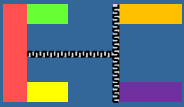
En el programa anterior hemos utilizado solo dos números, pero si hubiese sido preciso trabajar con más números es conveniente el uso de bucles. A continuación se muestra un ejemplo de cómo realizar la suma de un vector formado por seis números. Verificar su correcto funcionamiento.

```
# Ejemplo : Suma componentes de un vector

        .data
vector: .word 6,7,8,9,10,1    # Suma será 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=6
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
```





# Consideraciones a tener en cuenta

## Práctica

- ④ La realización de la práctica consiste en la ejecución y contestación a las cuestiones de cada uno de los 5 ejercicios propuestos.
- ④ Se debe entregar una memoria con la respuesta a cada una de las cuestiones planteadas. Cuando proceda, añadir volcados de pantalla de la ejecución para verificar su correcto funcionamiento. El documento de la memoria tendrá el siguiente formato:
  - Página 1: Nombre de la asignatura, título de la práctica, número de la práctica, nombre alumno, email alumno, D.N.I. del alumno, grupo teoría, fecha.
  - Página 2: índice de la práctica.
  - Página 3: listado de los archivos que entrega.
  - Página 4 y sucesivas el resto de la práctica (descripción de la práctica, qué hace, cómo lo hace, problemas surgidos, volcados de pantalla de la simulación, ...).
  - Las páginas deben tener número de página.
- ④ Se debe entregar también el código en ensamblador del programa con las modificaciones solicitadas.

