# Normas de estilo para el lenguaje ensamblador de MIPS:

Código:

- Usar nombres nemónicos para etiquetas y variables:
  voltaje1 en lugar de v1
- Usar nemónicos para registros:
  $a0 en lugar de $4
- Usar indentación y espaciado apropiados para facilitar la lectura del programa.
- Usar convenios de llamadas a procedimientos (ver transparencias 20 y 21 del tutorial). Una violación razonable de estos convenios deberá documentarse claramente en los comentarios. Por ejemplo, todos los registros guardados por el invocador (caller-save) deberían ser guardados en la pila por el invocador (caller). Pero si alguno de esos registros no se usan en el invocado (callee) no hay porque guardarlo. SIN EMBARGO, hay que incluir un comentario diciendo que esos registros normalmente se guardan.

Comentarios:

- Usar comentarios significativos:

```
add  $t0,$t2,$t4    # temp ← voltaje1 + voltaje2
```

en lugar de:

```
add  $t0,$t2,$t4    # sumar los registros $t2 y $t4 y poner resultado en $t0
```

# Práctica 1. Introducción al simulador SPIM.

## Objetivos:

- Instalación y manejo del entorno de SPIM.
- Operaciones básicas con la memoria, para comprender la forma de localización de datos y direcciones.

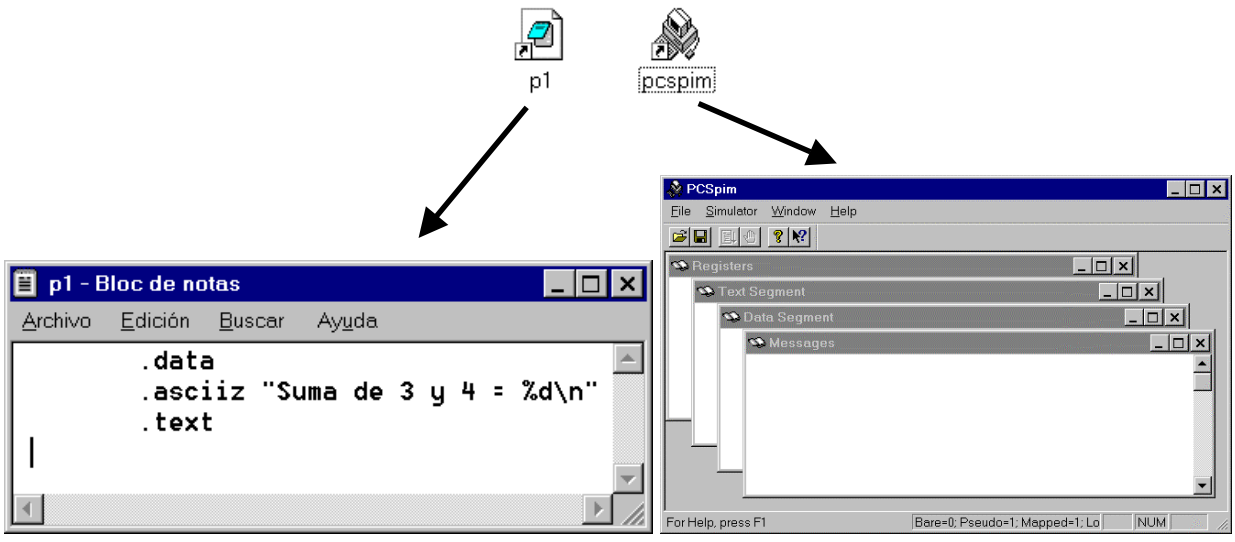
## Fundamentos teóricos.

- **Leer las transparencias 1, 2, 4 y 5 del tutorial.**
- El contenido de las transparencias 3, 6 y 7 resume algunos aspectos del formato del ensamblador MIPS, aspectos que también se ven en la parte teórica de la asignatura.
- **Conviene leer y tener presentes las transparencias 22 a la 28**, en las que se muestran distintas características del simulador SPIM, y que iremos viendo a lo largo de este guión.

## Desarrollo.

- Instalación de SPIM. Si vamos a utilizar un PC que no tiene SPIM instalado, ejecutamos el fichero de instalación del programa. Una vez instalado, ejecutamos el programa. Lo primero que observamos es un entorno de ventanas con menús y sub-menús. Es conveniente tener todas las ventanas abiertas (*messages, registers, text segment* y *data segment*). Observar que las ventanas no contienen nada aún.

- Conviene tener abierto un editor de texto ASCII en Windows. Se recomienda el "block de notas", o "notepad". En este editor se escribirá el código ensamblador MIPS, y lo almacenaremos con un fichero con extensión **.s**, por ejemplo **p1.s**, en una camino de datos temporal (como c:\temp ó a:\). Se recomienda encarecidamente el uso de diskettes. Escribimos el código ensamblador indicado en la figura.

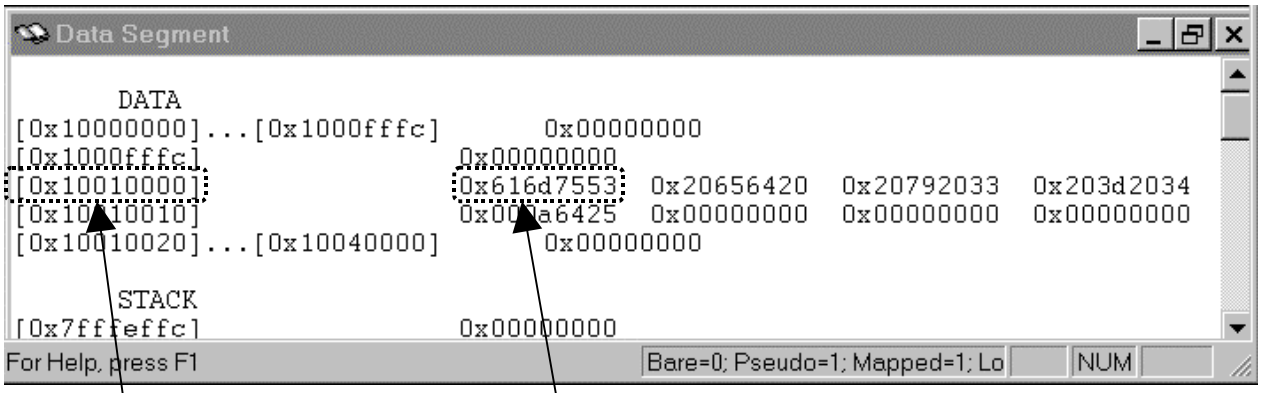


- Algunas versiones de SPIM tienen un “bug”, que se evita haciendo que los códigos ensamblador que almacenamos en los ficheros **.s** terminen siempre con una línea en blanco (vacía).

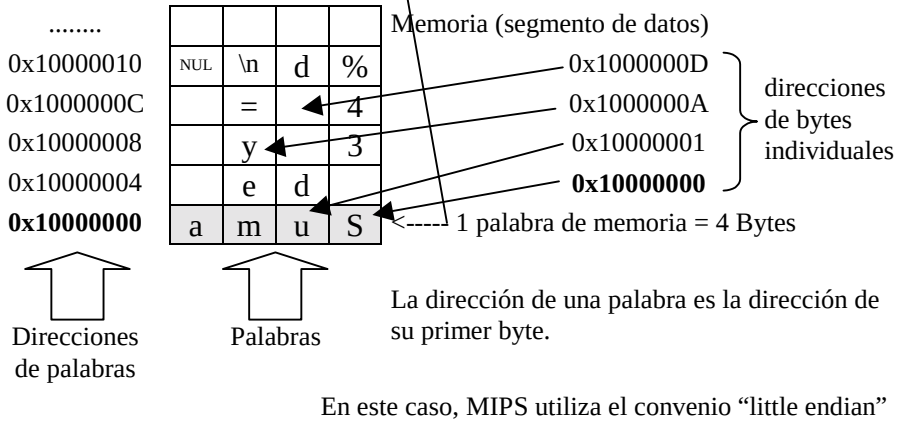
- El código **p0.s** de esta práctica no contiene instrucciones, luego no es necesario ejecutarlo. Consta de la declaración de un segmento de datos estático, es decir, mediante la directiva **.data** indicamos al ensamblador que vamos a utilizar una porción de memoria en la que vamos a declarar unos contenidos para unas direcciones dadas (ver transparencia 16). También, es necesario declarar el segmento de texto (mediante la directiva **.text**), que en este caso estará vacío, es decir, que no tendrá insturcciones.

- Para cargar el programa, vamos al menú FILE -> OPEN, y seleccionamos p0.s en el directorio adecuado. Si no hay error en el código, en la ventana *messages* debe aparecer un texto que concluya con la frase *".....\p1.s has been successfully loaded*".

- Al cargarlo en SPIM, se cargarán en el segmento de datos los bytes que se han definido mediante la directiva **.asciiz**. Esto no es una ejecución del programa: la ejecución del programa sería la ejecución de las instrucciones presentes en el segmento de texto. Abrir la ventana del segmento de datos, e interpretar las direcciones de memoria y el contenido de las mismas. Las siguientes figuras ayudan a comprender cómo es la estructuración de la memoria.




| Ascii | S | u | m | a | | d | e | | 3 | | y | | 4 | | = | | % | d | \n | NULL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dec | 83 | 117 | 109 | 97 | 32 | 100 | 101 | 32 | 51 | 32 | 121 | 32 | 52 | 32 | 61 | 32 | 37 | 100 | 10 | 0 |
| Hex | 53 | 75 | 6D | 61 | 20 | 64 | 65 | 20 | 33 | 20 | 79 | 20 | 34 | 20 | 3D | 20 | 25 | 64 | 0A | 00 |

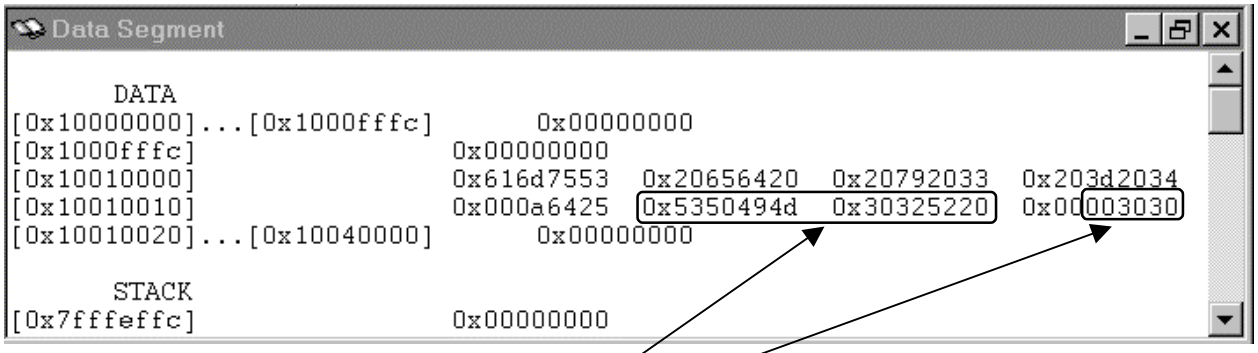



En este caso, MIPS utiliza el convenio “little endian”

- Vamos a colocar, a continuación de la directiva **.asciiz**, otra directiva, llamada **.byte** (ver transparencia nº 8) que carga datos en memoria en forma de bytes, de forma que éstos se colocarán inmediatamente después de los últimos cargados en memoria:

```
        .data
        .asciiz "Suma de 3 y 4 = %d\n"
        .byte 77,73,80,83,32,82,50,48,48,48,0   #Cadena “MIPS R2000”
        .text
```

- Grabar el fichero p1.s modificado, y en el menú Simulator, hacer sucesivamente *Clear Registers*, *Reinizialite* y *Reload*.... El contenido de la memoria ha de ser el siguiente:




Nuevos datos colocados en la memoria:

# Práctica 2: Introducción al ensamblador MIPS.

## Objetivos:

- Carga y ejecución de un código ensamblador MIPS en el simulador SPIM.
- Localización del segmento de datos y el segmento de código.
- Arrays y constantes en memoria.
- Operaciones de carga de memoria a registros.

## Desarrollo:

- SPIM no lleva editor de texto asociado, por lo que el código ha de manipularse con algún procesador de textos (ASCII), como ya se indicó en la anterior práctica. Escribir el código de abajo en un fichero con extensión .S (no olvidar de incluir una línea en blanco al final).

```
          .data                    # *** SEGMENTO DE DATOS ***
valor:    .word  8,9,10,11         # Defino array 4 palabras (decimal).
                                   # valores[2] es 10 = 0xa
          .byte  0x1a,0x0b,10      # Defino los 3 primeros bytes de la
                                   # siguiente palabra (hex. y dec.).
          .align 2                 # Para que el siguiente dato en mem.
                                   # esté alineado en palabra
          .ascii "Hola"            # Cadena de caracteres
          .asciiz",MIPS"           # Cadena de caracteres terminada
                                   # con el caracter nulo.
          .align 2                 # Las dos cadenas anteriores, junto
                                   # con el NULL final, ocupan 10 bytes,
                                   # por lo que alineo el siguiente dato
                                   # para que empieze en una palabra
id:       .word  1                 # Usado como índice del array "valor"
#-----------------------------------------------------------------------------
          .text                    # *** SEGMENTO DE TEXTO ***
          .globl main              # Etiqueta main visible a otros ficheros
main:                              # Rutina principal
          lw     $s0,valor($zero) # Carga en $s0 valor[0]
                                   # También vale: lw $s0,valor
          lw     $s4,id            # En $s4 ponemos el índice del array
          mul    $s5,$s4,4         # En $s5 ponemos id*4
          lw     $s1,valor($s5)    # Carga en $s1 valor[1]
          add    $s4,$s4,1         # Incrementamos el índice del array
          mul    $s5,$s4,4
          lw     $s2,valor($s5)    # Carga en $s2 valor[2]
          add    $s4,$s4,1         # Incrementamos el índice del array
          mul    $s5,$s4,4
          lw     $s3,valor($s5)    # Carga en $s3 valor[3]
```

- Ejecutar el simulador SPIM. Cargar el código ensamblador. Comprobar en la ventana de mensajes que la carga ha sido correcta.

⇒ Ver la ventana de **registros**. Observar los que hay, y que todos están inicializados a cero (excepto los que el S.O. ha dejado inicializados a ciertos valores). Podemos ver un listado de los registros y sus significados en la transparencia nº 9 del tutorial.

⇒ Ver la ventana del **segmento de datos**. Comprobar que todos los bytes que aparecen en las palabras de memoria corresponden con las definiciones de datos que aparecen en las directivas del código ensamblador.

⇒ Ver la ventana del **segmento de texto**. Observar que hay tres columnas. En la tercera aparece un código algo diferente al que hemos escrito. El código que hemos cargado en un fichero es un ensamblador con rótulos, escrito por el programador. El ensamblador lo convierte en un código ensamblador sin rótulos, apto para ser directamente convertido en código máquina (binario), que está descrito en la segunda columna. La primera columna informa de las direcciones de memoria donde se encuentra cada una de las instrucciones. En la versión SPIMWIN se puede observar cómo las primeras líneas de código no han sido escritas por el programador: son las instrucciones que el ensamblador introduce para una correcta carga y manipulación del programa en memoria. Las ignoraremos por ahora, ya que no son las que realizan las operaciones indicadas en nuestro código.

- Ejecutar el programa cargado en SPIM. La dirección que aparece en una ventana de confirmación corresponde a la dirección de inicio del programa que se va a ejecutar (inicialización del registro contador del programa, PC), y que corresponde con la promera de las instrucciones que el ensamblador introdujo para la adecuada ejecución de nuestro programa.

- Abrir la ventana de registros, y observar los valores que han tomado algunos de ellos. Para explicar esos valores, observar detenidamente las operaciones descritas por las instrucciones de nuestro código.

- Estudiar cómo se cargan en registros las distintas posiciones de un array de datos en memoria, y qué operaciones aritméticas es necesario realizar (y porqué).

- Estudiar las directivas e instrucciones que aparecen en el código (ver transparencia nº 8 del tutorial).