Trabajo 5

Introducción al simulador SPIM.

Facultad De Ingeniería, Universidad De Cuenca
ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS
Freddy L. Abad L.
{freddy.abadl}@ucuenca.edu.ec

PRACTICA 1

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.

Ya que la Práctica fue escrita hace algunos años, el programa que se utilizó fue QtSPIM en lugar de MIPS, por distintas ventajas que brindan las recientes versiones.

INSTALACIÓN:

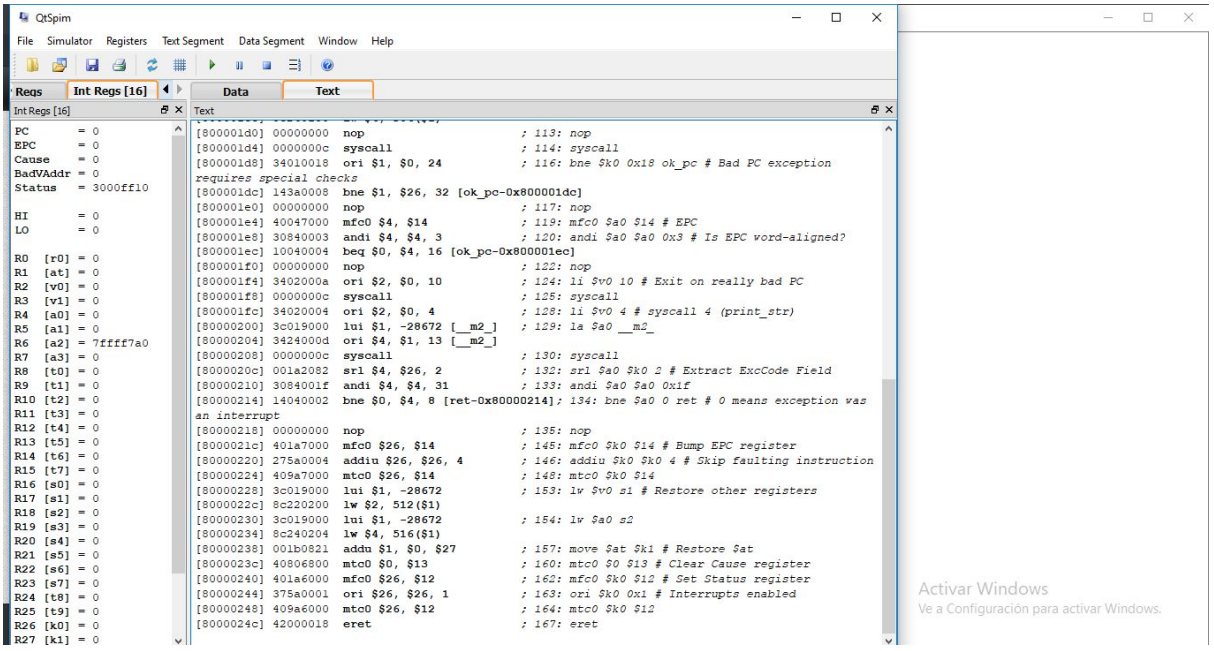
Older Versions of SPIM

es the older versions of SPIM. Please use *QtSpim* instead.

Most Recent Versions of SPIM			
Platform	**Program**	**Form**	**File**
Unix or Linux system Mac OS X	*spim xspim*	Source code	http://spimsimulator.svn.sourceforge.net/viewvc/spimsimulator/
Microsoft Windows (spim 7.0 and later versions no longer run on Windows 95/98. Use version 6.5 or earlier.)	*PCSpim*	Executable	http://sourceforge.net/projects/spimsimulator/files/PCSpim_9.0.zip/download
		Source code	http://spimsimulator.svn.sourceforge.net/viewvc/spimsimulator/

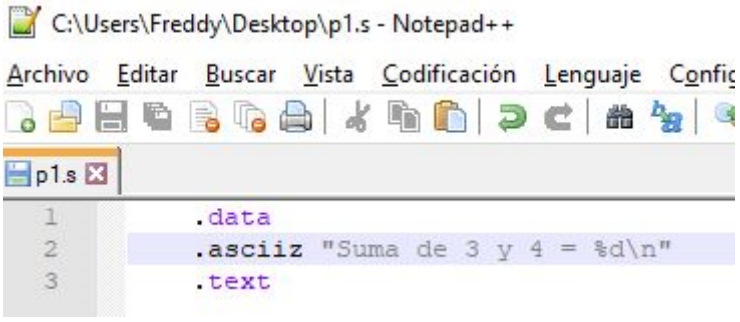
Panel de descarga en la página oficial de SPIM

Icono QtSpim creado en un SO - Windows 10



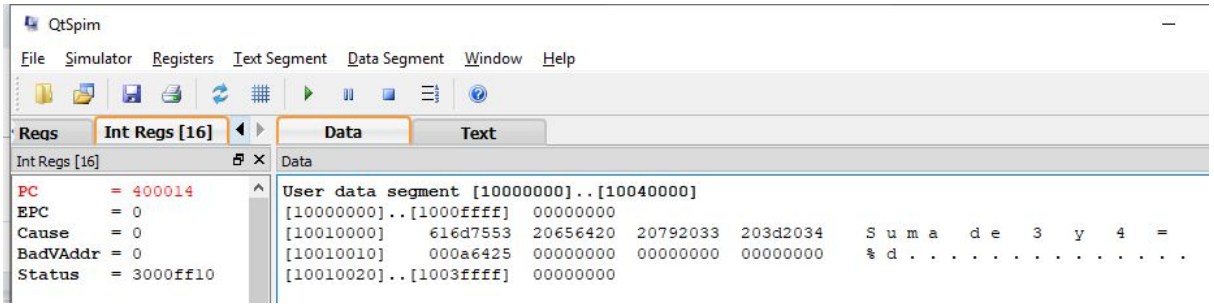
Entorno Principal - Ventanas principales de QtSPIM

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.

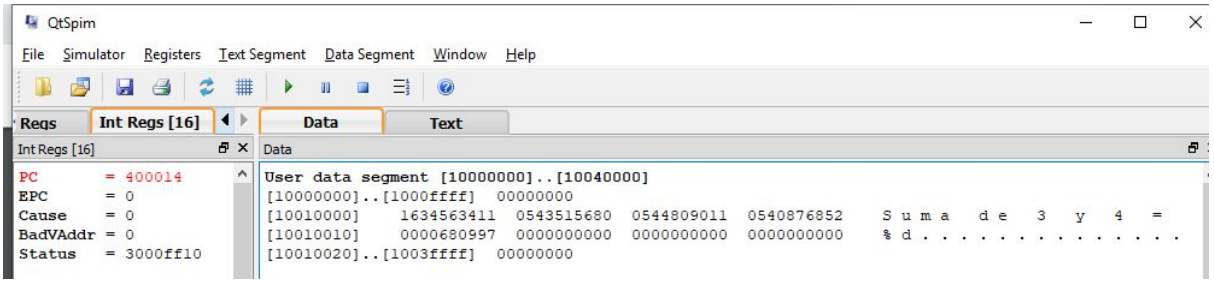


Sentencias MIPS

- La directiva .data perteneciente a MIPS señala al ensamblador que se utilizara una porción de memoria donde se declara unos contenidos para unas direcciones dadas.
- La directiva .text declara un segmento de texto.
- La directiva .asciiz define los segmentos de datos y el número de bytes a reservar.

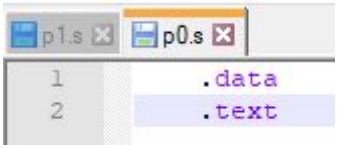


Segmento de datos del archivo p1.s en formato Hexadecimal

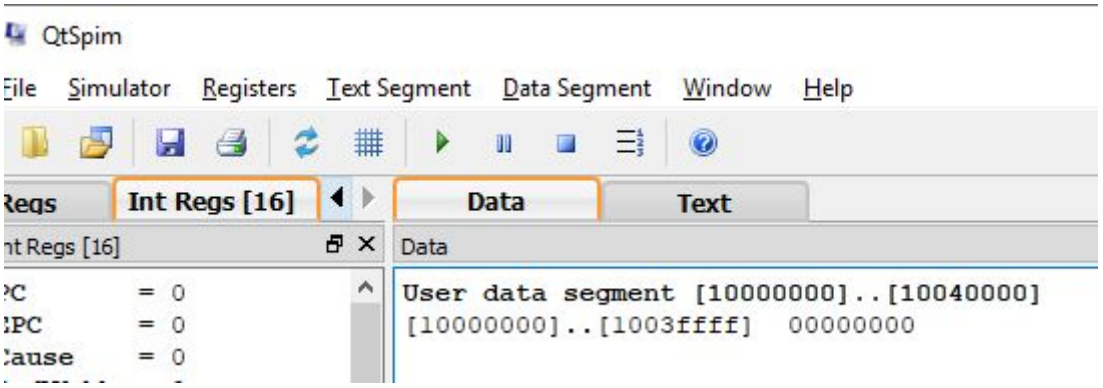


Segmento de datos del archivo p1.s en formato Decimal

- 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 instrucciones.



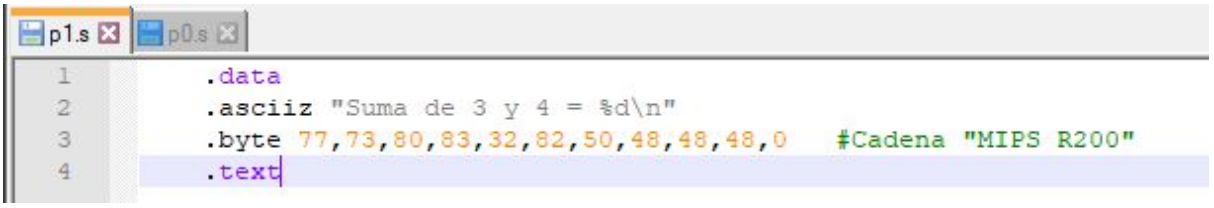
Archivo p0.s



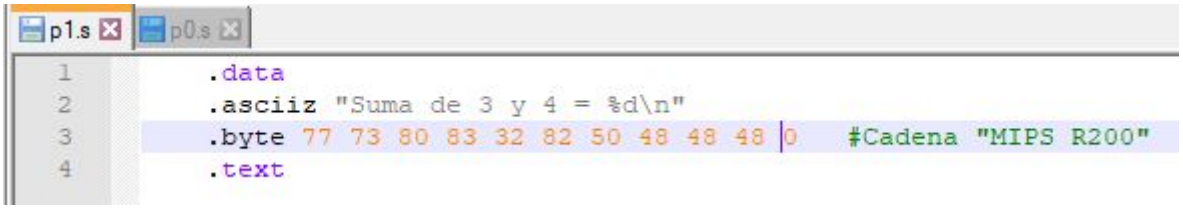
Salida del programa referente al p0.s

- 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.

- 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:

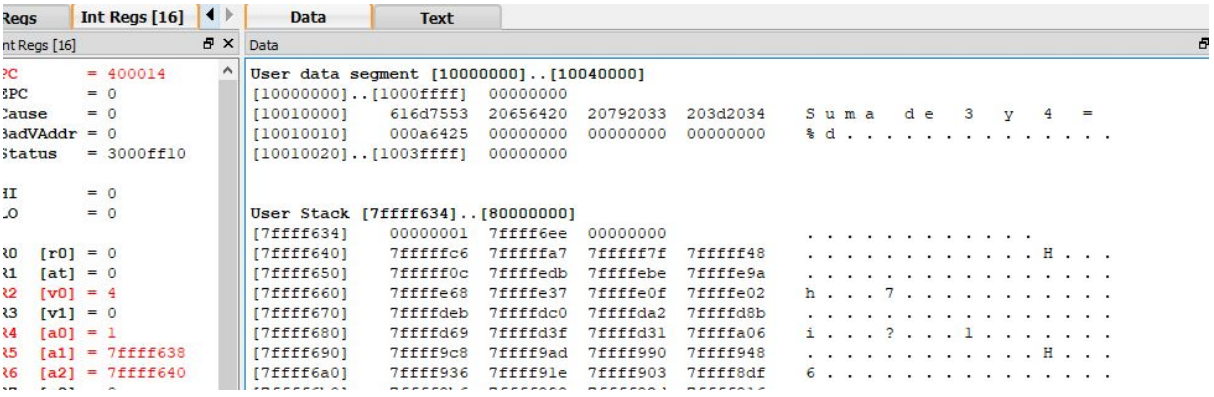


Archivo .s dado el formato de la práctica, sin embargo , presenta errores al cargar



Archivo .s corregido

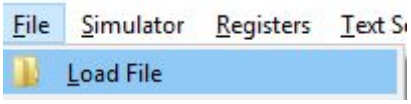
- Grabar el fichero p1.s modificado, y en el menú Simulator, hacer sucesivamente Clear Registers, Reinizialite y Reload.



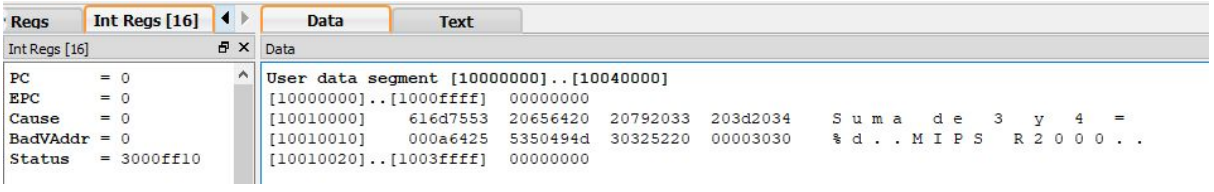
Salida de datos previo a limpiar los registros y reiniciar el proceso.

Clear Register

Reinitialized Simulator



Reload file



Archivo cargado

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).

Código del archivo p1.s

p1.s | p0.s

```
1         .data                     # *** SEGMENTO DE DATOS ***
2   valor:  .word 8 9 10 11           # Defino array 4 palabras (decimal).
3                                     # valores[2] es 10 = 0xa
4         .byte 0xla,0x0b,10        # Defino los 3 primeros bytes de la
5                                     # siguiente palabra (hex. y dec.).
6         .align 2                  # Para que el siguiente dato en mem.
7                                     # esté alineado en palabra
8         .ascii "Hola"             # Cadena de caracteres
9         .asciiz",MIPS"            # Cadena de caracteres terminada
10                                    # con el caracter nulo.
11        .align 2                  # Las dos cadenas anteriores, junto
12                                    # con el NULL final, ocupan 10 bytes,
13                                    # por lo que alineo el siguiente dato
14                                    # para que empieze en una palabra
15  id:     .word 1                   # Usado como índice del array "valor"
16  #-------------------------------------------------------------------------------
17      .text                         # *** SEGMENTO DE TEXTO ***
18      .globl main                   # Etiqueta main visible a otros ficheros
19  main:                             # Rutina principal
20      lw $s0,valor($zero)           # Carga en $s0 valor[0]
21                                    # También vale: lw $s0,valor
22      lw $s4,id                     # En $s4 ponemos el índice del array
23      mul $s5,$s4,4                 # En $s5 ponemos id*4
24      lw $s1,valor($s5)             # Carga en $s1 valor[1]
25      add $s4,$s4,1                 # Incrementamos el índice del array
26      mul $s5,$s4,4
27      lw $s2,valor($s5)             # Carga en $s2 valor[2]
28      add $s4,$s4,1                 # Incrementamos el índice del array
29      mul $s5,$s4,4
30      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.

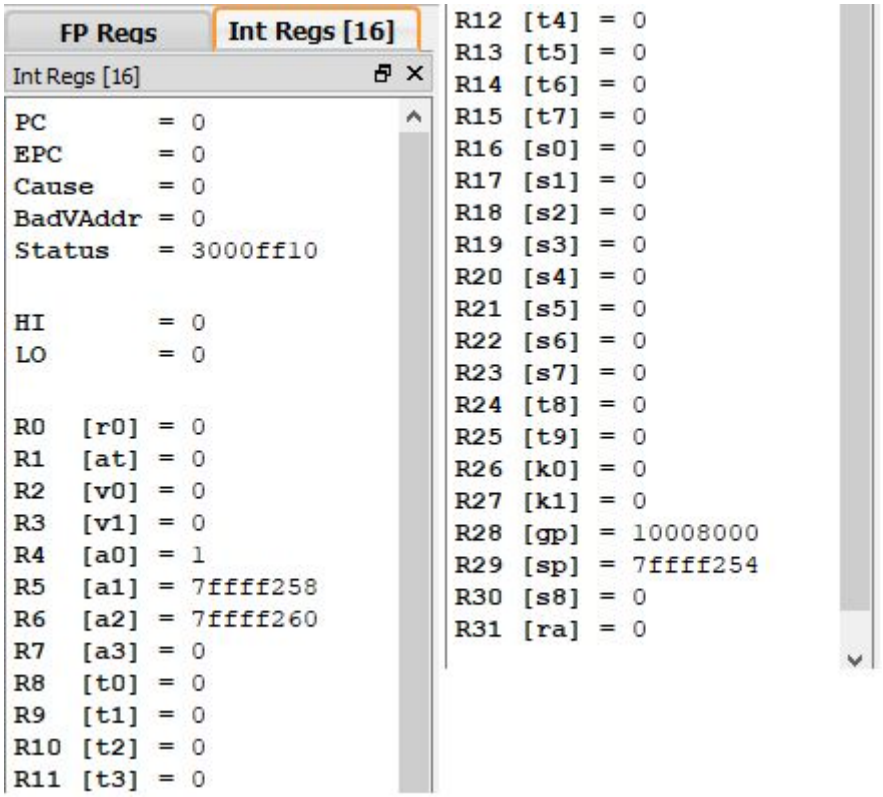
Salida del fichero p1.s

Data | Text

Data

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff]  00000000
[10010000]    00000008  00000009  0000000a  0000000b    . . . . . .
[10010010]    000a0b1a  616c6f48  50494d2c  00000053    . . . . H o l a , M I P S . . .
[10010020]    00000001  00000000  00000000  00000000    . . . . . .
[10010030]..[1003ffff]  00000000
```

- 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.
  La siguiente ventana es previa a la carga del archivo .s, así que se tiene encerado encerados.

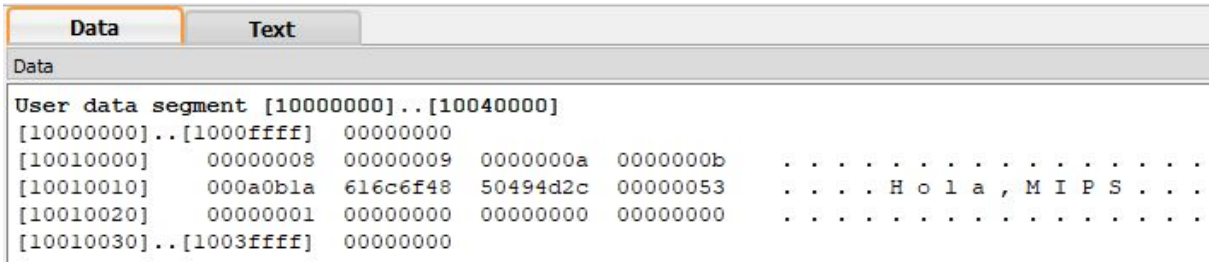


- Ver la ventana del segmento de datos. Comprobar que todos los bytes que aparecen en las palabras de memoria corresponden con las definiciones de datos que aparecen en las directivas del código ensamblador.

p1.s p0.s

```
1           .data                   # *** SEGMENTO DE DATOS ***
2   valor:  .word 8 9 10 11         # Defino array 4 palabras (decimal).
3                                   # valores[2] es 10 = 0xa
4           .byte 0xla,0x0b,10      # Defino los 3 primeros bytes de la
5                                   # siguiente palabra (hex. y dec.).
6           .align 2                # Para que el siguiente dato en mem.
7                                   # esté alineado en palabra
8           .ascii "Hola"           # Cadena de caracteres
9           .asciiz",MIPS"          # Cadena de caracteres terminada
10                                  # con el caracter nulo.
11          .align 2                # Las dos cadenas anteriores, junto
12                                  # con el NULL final, ocupan 10 bytes,
13                                  # por lo que alineo el siguiente dato
14                                  # para que empieze en una palabra
15  id:     .word 1                 # Usado como índice del array "valor"
```

Código del Fichero .s



Salida del fichero p1.s

- 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.

```
        .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]
        syscall
```

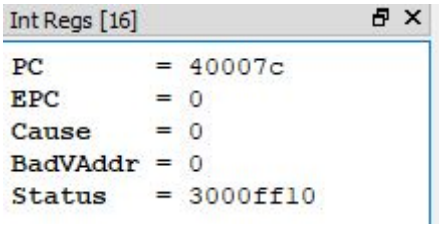
Código del Fichero .s

```
[00400024] 3c011001  lui $1, 4097            ; 17: lw $s0,valor($zero)
[00400028] 8c300000  lw $16, 0($1)
[0040002c] 3c010000  lui $1, 0 [id]          ; 18: lw $s4,id
[00400030] 8c340000  lw $20, 0($1) [id]
[00400034] 34010004  ori $1, $0, 4           ; 19: mul $s5,$s4,4
[00400038] 7281a802  mul $21, $20, $1
[0040003c] 3c011001  lui $1, 4097            ; 20: lw $s1,valor($s5)
[00400040] 00350821  addu $1, $1, $21
[00400044] 8c310000  lw $17, 0($1)
[00400048] 22940001  addi $20, $20, 1        ; 21: add $s4,$s4,1
[0040004c] 34010004  ori $1, $0, 4           ; 22: mul $s5,$s4,4
[00400050] 7281a802  mul $21, $20, $1
[00400054] 3c011001  lui $1, 4097            ; 23: lw $s2,valor($s5)
[00400058] 00350821  addu $1, $1, $21
[0040005c] 8c320000  lw $18, 0($1)
[00400060] 22940001  addi $20, $20, 1        ; 24: add $s4,$s4,1
[00400064] 34010004  ori $1, $0, 4           ; 25: mul $s5,$s4,4
[00400068] 7281a802  mul $21, $20, $1
[0040006c] 3c011001  lui $1, 4097            ; 26: lw $s3,valor($s5)
[00400070] 00350821  addu $1, $1, $21
[00400074] 8c330000  lw $19, 0($1)
```

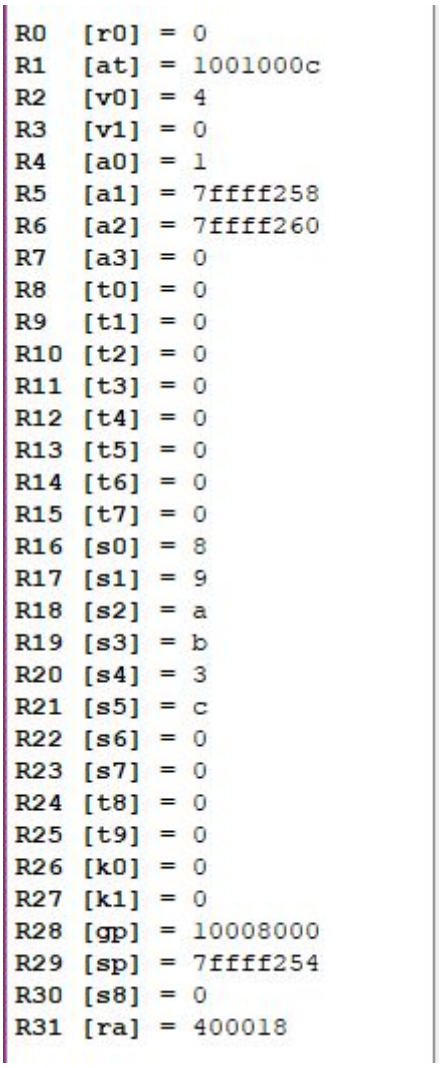
Segmento del texto

- 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 primera de las instrucciones que el ensamblador introdujo para la adecuada ejecución de nuestro programa.

Contador del programa posterior a la ejecución



- 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).

- Cargar valores.
- Guardar el índice del array en una variable.
- Una operación aritmética es el incremento del índice el cual nos permite acceder a cada datos del array.

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

Directiva	Descripcion
`.align n`	Alinea el siguiente dato sobre un límite de 2^(n) byte (.align 2, .align 0, ...)
`.ascii str`	Almacena la cadena str en memoria pero no la termina con el carácter nulo
`.asciiz str`	Almacena la cadena str en memoria y la termina con el carácter nulo

`.byte/half/word` `b1, … , bn`	Almacena los n valores de 8/16/32 bits en bytes/medias palabra / palabras consecutivas de memoria
`.data`	Los elementos siguientes son almacenados en el segmento de datos.
`.globl sym`	Declara sym como global: se puede referenciar desde otros archivos
`.text`	Los elementos siguientes son almacenados en el segmento de texto. Estos elementos solo pueden ser instrucciones o palabras