ESTRUCTURA DE COMPUTADORES

GRADO EN INGENIERÍA INFORMÁTICA

UNIVERSIDAD CARLOS III DE MADRID

Práctica 2
Ensamblador de MIPS e introducción al simulador QTSpim

Octubre de 2014

Contenido

Objetivos de la práctica

El objetivo de esta práctica es que el alumno se familiarice con la programación en ensamblador y la representación de distintos tipos de datos utilizando el ensamblador del MIPS32. Para el desarrollo de esta práctica se usará el emulador QTSPIM disponible en:
http://spimsimulator.sourceforge.net/

QTSPIM es un simulador auto-contenido que ejecutará programas escritos en el lenguaje ensamblador del MIPS32. QTSPIM proporciona además un depurador y un conjunto mínimo de servicios del sistema operativo.

NOTA: Las subrutinas pedidas deben seguir OBLIGATORIAMENTE el nombre indicado en las siguientes secciones. Se ha de tener en cuenta que un nombre con mayúsculas es diferente de otro con minúsculas. Por ejemplo, los siguientes nombres corresponden a subrutinas diferentes:

imprimir_entero, Imprimir_Entero, imprimir_ENTERO, etc.

Observe que los nombres de función que se piden en esta práctica tienen todas sus letras en MINÚSCULAS.

Descripción del simulador QTSPIM

La versión para Windows de QTSPIM posee una interfaz similar a la que aparece en la siguiente figura. Hay dos paneles horizontales. El panel de la izquierda contiene los valores de los registros, y el panel de la derecha contiene el código y los datos declarados.

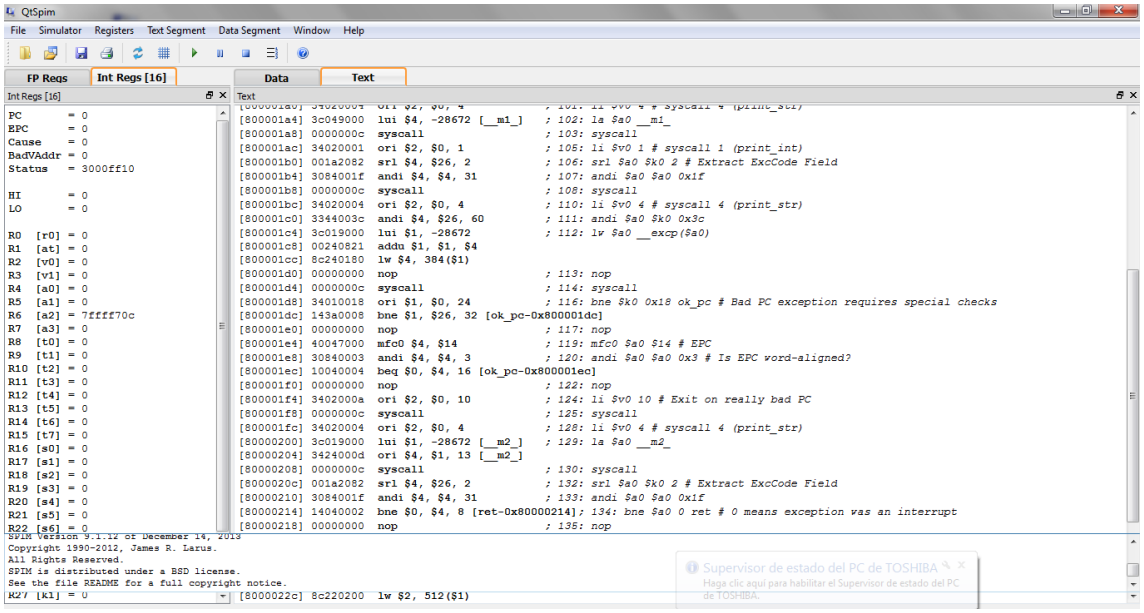
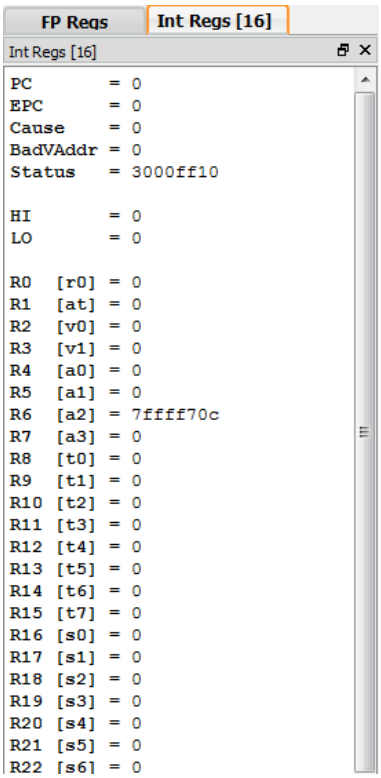
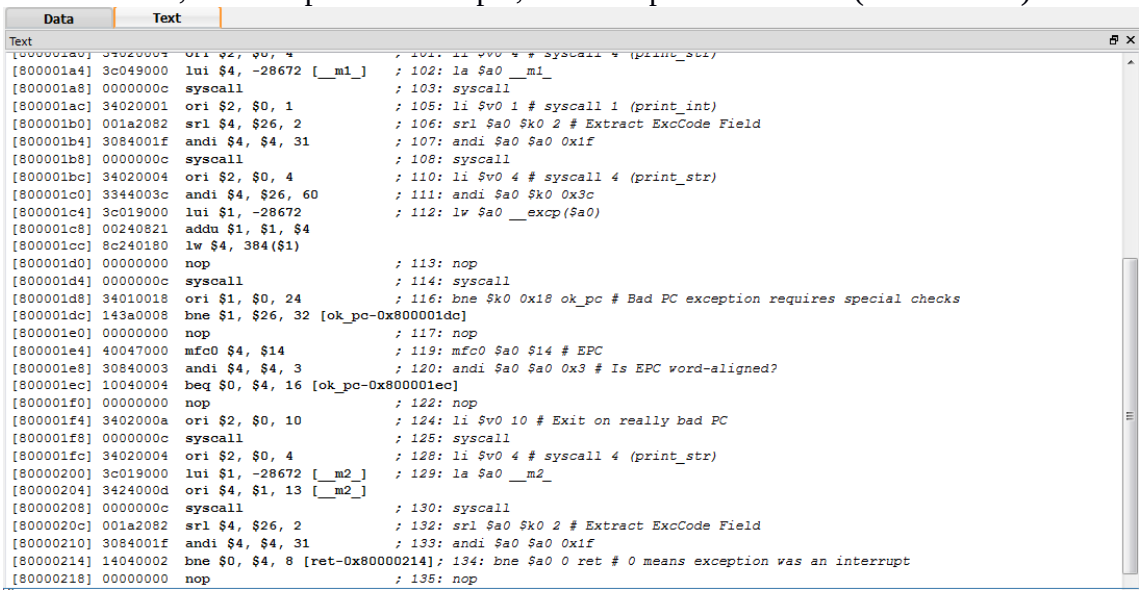


Figura 1. Interfaz del simulador QTSPIM



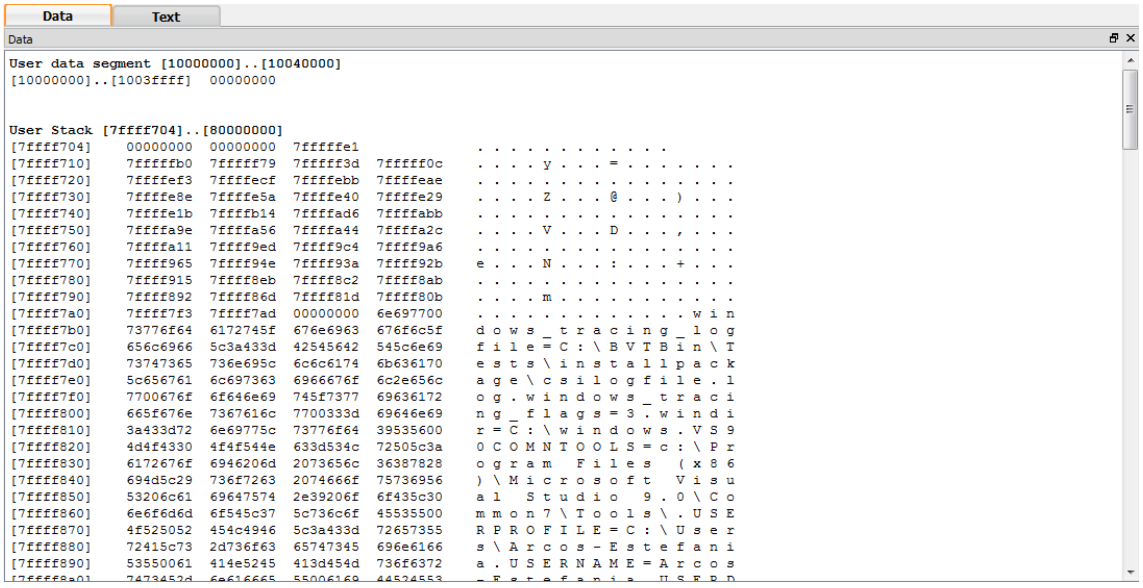
PANEL 1. Registros de la máquina MIPS.

Incluye el banco de registros de la máquina MIPS.
Primero aparece el contador de programa (**PC**) y otros registros especiales como **HI** (almacena parte alta de registro) y **LO** (almacena la parte baja).
En las siguientes líneas aparecerán los 32 registros enteros de propósito general (**R0** a **R31**).
En la pestaña FP Regs aparecen los 32 registros usados para representar números en coma flotante, tanto en precisión simple, como en precisión doble (**FP0** a **FP31**).



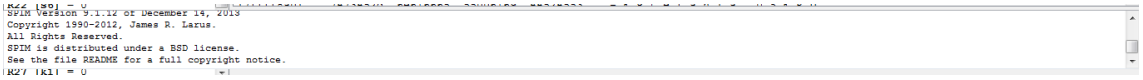
PANEL 2. Segmento de Texto.

Incluye el código del programa ensamblador a ejecutar, identificado mediante la directiva *.text*.
Cada *pseudoinstrucción* ocupa una única línea, identificada por una dirección de memoria que aparece en la parte izquierda de la línea.
La primera dirección de memoria utilizada para almacenar el texto de un programa es 0x00400000.



PANEL 3. Segmento de Datos y pila.

Incluye el conjunto de datos definidos en el segmento de datos *.data* del programa a ejecutar. Estos datos se almacenan de manera lineal en memoria, a partir de la dirección **0x10000000.** A continuación se muestra el contenido de la pila (*stack*) que comienza en la dirección **0x7fffefff**.



PANEL 4. Pantalla para depuración de programas

El último panel es el depurador, destinado a visualizar mensajes de control o error del simulador QTSPIM.

Consideraciones generales sobre invocación a subrutinas

Convenio para paso de parámetros y retorno

El paso de parámetros a subrutina se hace mediante registros. La pila se usa solo cuando es necesario. El paso de parámetros se realiza de la siguiente forma:

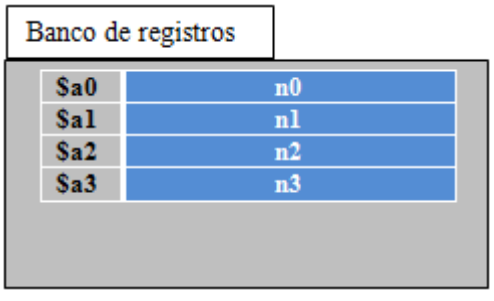
- Los parámetros que no sean números en coma flotante se pasan usando los registros $a0, $a1, $a2 y $a3 del banco general de registros. Si hay más de cuatro parámetros de este tipo, los parámetros adicionales se pasan en la pila.
- Los parámetros de valores en coma flotante en simple precisión se pasan usando los registros $f12, $f13, $f14 y $f15 del coprocesador de coma flotante. Si hay más de cuatro parámetros de este tipo, los parámetros adicionales se pasan en la pila.
- Los parámetros de valores en coma flotante de doble precisión se pasan usando los registros $f12 y $f14. Si hay más de dos parámetros de este tipo, los parámetros adicionales se pasan en la pila.
- Los resultados que no sean valores de coma flotante se devuelven en los registros $v0 y $v1. Si se devuelven más valores, éstos se han de devolver en la pila.
- Los resultados de valores en coma flotante se devuelven en el registro $f0. Si se devuelven más valores, éstos se han de devolver en la pila.
- Cuando una subrutina es invocada, debe dejar la pila en el mismo estado que la recibió. Es decir, la función llamante debe apilar el registro de retorno ($ra) y los parámetros necesarios en la pila y llamar a la subrutina. Cuando la subrutina ha retornado debe restaurar el registro de retorno ($ra) y restaurar la pila.

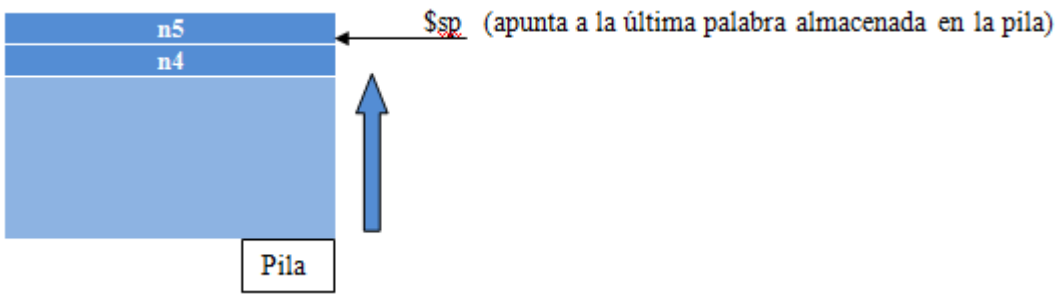
Ejemplo 1: para una función que tenga el siguiente prototipo en Java:

```
int  funcion (int n0, int n1, int n2, int n3, int n4, int n5)
```

- El parámetro n0 se pasará en el registro $a0.
- El parámetro n1 se pasará en el registro $a1.
- El parámetro n2 se pasará en el registro $a2.
- El parámetro n3 se pasará en el registro $a3.
- El parámetro n4 se coloca en la cima de la pila.
- El parámetro n5 se coloca encima del anterior (y así sucesivamente con el resto de parámetros que reciba la función).
- *El entero resultado se retorna en el registro $v0.*

En la siguiente figura se puede ver gráficamente el ejemplo 1 descrito.





El código de la función debe asumir que el valor del parámetro n1 se encuentra en el registro $a1 y que el valor del parámetro n4 se encuentra en la posición de la pila correspondiente.

Ejemplo 2: para una función que tenga el siguiente prototipo en Java:

```
float  funcion (float n0,float n1,float n2,float n3,float n4,float n5)
```

- El parámetro n0 se pasará en el registro $f12.
- El parámetro n1 se pasará en el registro $f13.
- El parámetro n2 se pasará en el registro $f14.
- El parámetro n3 se pasará en el registro $f15.
- El parámetro n4 se coloca en la cima de la pila.
- El parámetro n5 se coloca encima del anterior (y así sucesivamente con el resto de parámetros que reciba la función).
- *El float resultado se retorna en el registro $f0.*

Convenio para paso de arrays por parámetro

Para pasar como parámetro un array de valores (independientemente del tipo de los valores que haya en el array), se han de pasar dos parámetros:

- En el primer parámetro ($a0) se pasa la dirección de memoria de comienzo del array.
- En el segundo parámetro ($a1) se pasa el número de elementos del array.

Ejemplo 3: para una función que tenga el siguiente prototipo en Java:

```
int funcion (float[] vector,  int n)
```

- El parámetro vector (dirección de comienzo de vector) se pasará en el registro $a0.
- El parámetro n se pasará en el registro $a1.
- *El int resultado se retorna en el registro $v0.*

Ejemplo 4: para una función que tenga el siguiente prototipo en Java:

```
int funcion (char[] cadena)
```

- El parámetro cadena (dirección de comienzo de cadena) se pasará en el registro $a0.
- *El int resultado se retorna en el registro $v0.*

En el caso de cadenas de caracteres, no es necesario pasar el segundo argumento que indica la longitud, ya que se puede conocer el fin de la cadena, puesto que todas las cadenas finalizan con el byte 0.

Convenio para variables locales

Las variables locales deben mantener su valor durante todo el tiempo de vida de la función. Tenga en cuenta las siguientes recomendaciones:

- Las variables locales de una función se asignarán a registros siempre que sea posible. Para estas variables se pueden utilizar los registros $s o $t y para el coprocesador en coma flotante algunos $f.
- Para las variables que no se puedan almacenar en un registro, ya sea por su tamaño o porque no queden registros disponibles, se guardará en la pila.

En el MIPS existe un convenio respecto al uso de los registros. Se pueden considerar dos tipos de registros:

- Registros que deben preservar (registros preservados) su valor entre llamadas a funciones.
  Son los registros [$s, $f20 en adelante].
  El significado de registros preservados es que una función debe conservar el valor que tienen antes de poder modificarlos.

- Registros para los que no se garantiza el mantenimiento de su valor entre llamadas a funciones (registros temporales o no preservados).
  Son los registros [$t, $f4, $f6, $f8, $f10, $f16 y $f18].
  El significado de registros no preservados es que una función puede modificarlos sin preocuparse de conservar el valor que tenían anteriormente.

Ejercicio 1.

En el material de apoyo de la práctica 2, que se puede descargar de aula global, se dispone del fichero *ejercicio1.s*.
Analícelo, cárguelo en el simulador QTSpim, ejecútelo y responda a las siguientes preguntas en la memoria:
a) Indique la dirección de memoria donde comienza el main del *ejercicio1.s*.

b) Indique la dirección de memoria que contiene el registro `$ra` después de ejecutar el programa. Indique además la instrucción que contiene esa dirección de memoria.

c) La dirección **0x10010043** contiene un carácter de la cadena almacenada en el segmento de datos.

Indique el valor que contiene esa dirección de memoria, y el carácter ASCII correspondiente.
d) Indique en qué dirección está la primera **'a'** de la cadena almacenada en el segmento de datos.

e) Indique la dirección que corresponde a utilizar: **hueco+2**

f) Indique la dirección que corresponde a utilizar: **cadena+8**

g) Escriba las instrucciones necesarias para imprimir por pantalla el tercer byte de la cadena.

Ejercicio 2.

En un fichero llamado *indexOf.s* desarrollar un programa escrito en código ensamblador MIPS32 que dado un carácter y una cadena devuelva la posición en la que se encuentra ese carácter dentro de la cadena.

El alumno debe desarrollar la función **indexOf,** que busca en la cadena de caracteres la primera ocurrencia del carácter seleccionado, retorna su posición dentro de la cadena y se imprime por pantalla la posición. Una vez implementada la función debe invocar desde el main que aparece en el código proporcionado a la función desarrollada con los datos que aparecen en la sección de datos de dicho código.

La cadena de caracteres debe declararse en la región de datos de la siguiente forma:

```
.data
string:   .asciiz "nice work..."
char:     .byte 'w'
```

Dónde:

- **string :** es la cadena de caracteres donde se va a buscar.
- **char** : es el carácter a buscar.

La salida por consola con estos datos sería la siguiente:

```
5
```

Si el carácter no se encuentra dentro de la cadena se retornará un -1.

NOTAS:

- **El programa debe funcionar independientemente de la cadena que se asigne a string o el valor que se asigne a char.**

Ejercicio 3.

En un archivo con el nombre *srtLength.s*, desarrollar un programa escrito en código ensamblador MIPS32 que, dada una cadena, cuente el número de caracteres de la cadena definida en la zona de datos.

El alumno debe desarrollar la función **strlength** que cuenta el número de caracteres de una cadena. Para este cálculo no se tendrá en cuenta el carácter nulo. La función devuelve el número de caracteres.

Una vez implementada la función se invocará a la misma desde la función main pasando como parámetro la cadena definida en el segmento de datos. A continuación se imprimirá el valor retornado por la función strlength.

La cadena de caracteres de prueba deberá declararse en el segmento de datos del fichero strLength.s como .asciiz de la siguiente manera:

```
.data
string:   .asciiz "nice work..."
```

Dónde:

- **string** es la cadena de carateres a contar**.**

El resultado mostrado por consola para el ejemplo anterior sería:

```
11
```

NOTAS:

- **El programa debe funcionar independientemente de la cadena que se asigne a string.**

-

Ejercicio 4.

En un archivo con el nombre *average.s,* desarrollar un programa escrito en código ensamblador MIPS32 que, dado un vector de enteros calcula media de los valores del vector y la imprime por pantalla.

El alumno debe desarrollar el código de la función **average** que calcula la media de los valores guardados en un vector. La función devuelve el valor medio.

A continuación debe llamarse a dicha función desde la función main pasando como parámetro el vector definido en el segmento de datos. El valor retornado ha de imprimirse por pantalla a continuación desde la función main.

El vector de enteros debe declararse en la zona de datos del fichero *average.s,* de la siguiente manera:

```
.data
     vector:   .word 4 5 6 7 8 2 1
     size:     .word 7
```

Dónde:

- **vector** es el vector que contiene los valores a partir de los cuales se realizará el cálculo.
- **size** es el número de elementos que contiene el vector

El resultado mostrado por consola para el ejemplo anterior sería:

```
4.71
```

NOTAS:

- **El programa debe funcionar independientemente de los valores asignados a las posiciones del vector y de su tamaño.**

Ejercicio 5.

En un archivo con el nombre *minMax.s*, desarrollar un programa escrito en MIPS32 que, dado un vector de enteros, calcule el número máximo y mínimo de ese vector y lo imprima por pantalla.

El alumno debe desarrollar la función **minMax** que calcula el máximo y el mínimo de todos los números guardados en un vector. La función devuelve dos valores: el máximo y el mínimo.

A continuación debe llamarse a dicha función desde la función main pasando como parámetro el vector definido en el segmento de datos e imprimir posteriormente el valor máximo y mínimo.

El vector de enteros debe declararse en la zona de datos del fichero *minMax.s*, de la siguiente manera**:**

```
.data
     vector:   .word 4 5 6 7 8 2 1
     size:     .word 7
```

Dónde:

- **vector** es el vector que contiene los valores a partir de los cuales se realizará el cálculo.
- **size** es el número de elementos que contiene el vector

El resultado mostrado por consola para el ejemplo anterior sería:

```
1
8
```

NOTAS:

- **El programa debe funcionar independientemente de los valores asignados a las posiciones del vector y de su tamaño.**
- **Lo valores de retorno deben estar en el orden requerido: primero el mínimo del vector y en la siguiente línea el máximo del vector.**

Ejercicio 6.

En un archivo con el nombre *concat.s*, desarrollar un programa escrito en MIPS32 que dadas dos cadenas de caracteres, las concatene en una sola y la imprima por pantalla.

Se debe desarrollar la función **concat** que une dos cadenas de caracteres declaradas en la zona de datos en una sola. Finalmente, imprime la cadena resultante por pantalla.

Las dos cadenas deben declararse en la zona de datos del fichero *concat.s* de la siguiente manera:

```
.data
string1:  .asciiz "nice work..."
string2:  .asciiz "Yeah, good work."
```

Dónde:

- **string1** es la primera cadena de caracteres**.**
- **string2** es la segunda cadena de caracteres a concatenar.

La función main debe invocar a la función concat e imprimir el resultado después. El resultado mostrado por consola para el ejemplo anterior sería:

```
nice work...Yeah, good work.
```

NOTAS:

- **El programa debe funcionar independientemente de la cadena que se asigne a string1 y a string2.**
- **No se puede declarar ninguna otro elemento en la zona de datos del fichero.**
- **Para la reserva de memoria debe emplearse la llamada al sistema sbrk.**

Ejercicio 7.

En un archivo con el nombre *order.s*, desarrollar un programa escrito en MIPS32 que dado un vector de enteros, los ordene de menor a mayor, y posteriormente imprima el resultado por pantalla.

Se debe desarrollar el código de la función **order** que rdena un vector de enteros de menor a mayor.

A continuación se debe invocar desde la función main a la función anterior pasando como parámetro el vector declarado en la sección de datos. Una vez ordenado, la función main debe imprimir todos los elementos del nuevo vector ordenado.

El vector debe declararse en la zona de datos del fichero *order.s* de la siguiente manera:

```
.data
     vector:   .word 4 5 6 7 8 2 1
     size:     .word 7
```

Dónde:

- **vector** es el vector que contiene los valores a partir de los cuales se realizará el cálculo.
- **size** es el número de elementos que contiene el vector

El resultado mostrado por consola para el ejemplo anterior sería:

```
1 2 4 5 6 7 8
```

NOTAS:

- **El programa debe funcionar independientemente de los valores y el tamaño del vector.**
- **No se puede declarar ningún otro elemento en la zona de datos del fichero y se debe ordenar sin el uso de vectores auxiliares. La ordenación debe realizarse intercambiando posiciones en el vector original.**

Procedimiento de evaluación de la práctica

La evaluación de la práctica se va a dividir en dos partes.

- **Código (8 puntos)**
  - Ejercicio 1 (*1 punto*)
  - Ejercicio 2 (*1 punto*)
  - Ejercicio 3 (*1 punto*)
  - Ejercicio 4 (*1 punto*)
  - Ejercicio 5 (*1 punto*)
  - Ejercicio 6 (*1.5 puntos*)
  - Ejercicio 7 (*1.5 puntos*)
- **Memoria (2 puntos)**

Los ejercicios 1, 2, 3, 4 y 5, y la memoria son obligatorios. Deben entregarse para poder seguir la evaluación continua.

Aspectos importantes a tener en cuenta

Normas generales

1) Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.
2) Un programa no comentado, obtendrá una calificación de 0.
3) La entrega de la práctica se realizará a través de los entregadores habilitados. No se permite la entrega a través de correo electrónico sin autorización previa.
4) Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas, ambas obtendrán una calificación de 0.

Memoria de la práctica

La memoria tendrá que contener al menos los siguientes apartados:

- Portada donde figuren los autores (incluyendo nombre completo, NIA y dirección de correo electrónico).
- Índice de contenidos.
- Descripción de todos los programas en MIPS32 solicitados en este cuaderno de prácticas detallando las principales funciones implementadas. La memoria debe describir el comportamiento de los programas, así como las principales decisiones de diseño (adicionalmente se pueden incluir diagramas de flujo, algoritmos, etc.).
- Las respuestas a las preguntas planteadas en los ejercicios debidamente justificadas.
- Batería de pruebas utilizadas y resultados obtenidos. Se dará mayor puntuación a pruebas avanzadas, casos extremos, y en general a aquellas pruebas que garanticen el correcto funcionamiento de la práctica en todos los casos. Hay que tener en cuenta:
    - Evite pruebas duplicadas que evalúan los mismos flujos de programa. La puntuación de este apartado no se mide en función del número de pruebas, sino del grado de cobertura de las mismas. Es mejor pocas pruebas que evalúan diferentes casos a muchas pruebas que evalúan siempre el mismo caso.
- Conclusiones y problemas encontrados.

<u>NOTA:</u> NO DESCUIDE LA CALIDAD DE LA MEMORIA DE SU PRÁCTICA.

Aprobar la memoria es tan imprescindible para aprobar la práctica, como el correcto funcionamiento de la misma. Si al evaluarse la memoria de su práctica, se considera que no alcanza el mínimo admisible, su práctica estará suspensa.

La longitud de la memoria no deberá superar las 10 páginas (portada e índice incluidos)

Procedimiento de entrega de la práctica

La entrega de la práctica 1 se realizará de la siguiente manera: se habilitarán dos entregadores distintos, uno para los **ejercicios** de este cuaderno y otro para la **memoria** completa.
La fecha límite de entrega para ambos es el día **11 de Noviembre de 2014 a las 23:55 horas.**

- La entrega de las prácticas ha de realizarse de forma electrónica. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales podrá realizar la entrega de las prácticas.

- La única versión registrada de su práctica es la última entregada. La valoración de esta es la única válida y definitiva.

Entregador 1: Se deberá entregar un único archivo comprimido en formato zip con el nombre ec_p2_AAAAAAAAA_BBBBBBBBB.zip donde A…A y B…B son los NIA de los integrantes del grupo.

El archivo zip debe contener solo los siguientes archivos:

- **indexOf.s**
- **strLength.s**
- **average.s**
- **minMax.s**
- **concat.s**
- **order.s**

Entregador 2: Se deberá entregar la memoria en un único archivo en formato pdf con el nombre ec_p2_AAAAAAAAA_BBBBBBBBB.pdf donde A…A y B…B son los NIA de los integrantes del grupo.