TRABAJO DEL TEMA 2: ENSAMBLADOR MIPS

Plazo: una semana. **Límite de entrega**: viernes 22 de abril 2016, 23:55 horas.
Se permite entrega tardía durante el fin de semana, con penalización de 2 puntos a partir de las 23:55 horas del viernes y de 4 puntos a partir de las 00:00 horas del domingo. Pasadas las 23:55 horas del domingo, no se podrá entregar.

Funcionamiento:

Se ofrece aquí una serie de problemas y cuestiones con **diferentes valores de puntuación** según su dificultad. Cada alumno puede **escoger libremente** qué problemas quiere resolver; de esta manera cada uno puede graduar cuánto esfuerzo decide dedicar a este trabajo en función de la nota que quiera obtener. **La nota máxima es 10**, aunque el valor total de los problemas es mayor que 10.

Colaboración Vs. Plagio

Se permite consultar a profesores y otros alumnos sobre **técnicas y estrategias** para resolver los problemas, pero no se permite bajo ningún concepto emplear directamente soluciones ajenas. Si se detecta algún caso de plagio o similar, se aplicará un **suspenso directo** en la asignatura a todos los implicados, sin distinción sobre quién proporciona la información y quién la recibe. Además, en tal caso, se tomarán las medidas disciplinarias oportunas.

Formato de entrega:

(**Ficheros sueltos, no comprimidos**)

- Para todos los programas realizados, código **debidamente comentado** en fichero .asm listo para abrir y ejecutar con MARS.
- Documento Word (preferible) o bien .odt con la resolución de los problemas realizados:
  - Al principio del documento, la siguiente información:
    - Título del trabajo ("Trabajo del tema 2: Ensamblador MIPS").
    - Nombre y apellidos del autor.
    - Si se ha colaborado o se ha recibido ayuda, indicar "En colaboración con: ...".
  - Para cada problema resuelto:
    - Número o Título del problema
    - Enunciado del problema (salvo cuando el enunciado esté ya incluido en el código ASM).
    - Para los programas, cuadro de texto con el código correspondiente, igual que el entregado en fichero `.asm`.
    - Para los programas, explicación adicional de cómo funciona el programa y/o cómo se ha hecho para desarrollarlo. Puede ser una explicación breve.

Normas de programación:

- Seguir a rajatabla los convenios de uso de los registros MIPS (véase la referencia; para más detalles, véase el libro de Patterson y Hennesy).
- El flujo del programa debe seguir siempre las estructuras básicas (ejecución condicional, bucles y llamadas).
- Cada funcionalidad debe estar implementada en una rutina/función/módulo de manera que se pueda reutilizar en otro programa.
- Cada rutina/función solamente podrá usar los datos que se le pasen por valor o por referencia: queda prohibido utilizar etiquetas y registros que pertenezcan al programa principal.
- **Si se pide programar una función, su punto de entrada debe ser una etiqueta declarada global (directiva .globl) con el nombre exacto que el enunciado indique**. Además, el punto de entrada del programa principal será una etiqueta llamada *ppal* (no *main*), que coincidirá con la primera instrucción ejecutable de todo el fichero.
    - Esto último es necesario para que sea posible probar los de manera automatizada. Si al no cumplir estas normas se provoca que falle la corrección automática, el alumno en cuestión deberá ir a tutoría presencial para una corrección manual y tendrá además una penalización en nota.
- El nombre de cada fichero será una letra p minúscula seguida del número del problema (sin ceros a la izquierda) y con la extensión .asm. Ejemplo: el problema 3 se entregará como "p3.asm" y el problema 10 como "p10.asm". Los ficheros se entregarán por separado, no comprimidos.

PROBLEMAS DE COMPLEJIDAD BAJA

Problema 1.- (**0.5 puntos**) Programar una función llamada *abs* que tome como parámetro un número entero con signo y devuelva el valor absoluto de ese mismo entero. Emplear esta función en un programa principal sencillo para demostrar su funcionamiento.

Problema 2.- (**0.5 puntos**) Programar una función llamada *swap* que tome como parámetros dos punteros a enteros de tamaño palabra (Word). La función intercambiará los **contenidos** de las dos posiciones de memoria apuntadas por dichos punteros. Emplear esta función en un programa principal sencillo para comprobar su funcionamiento.

Problema 3.- (**1 punto**) Programar una función llamada *strupr* que tome como parámetros dos punteros a cadena: *orig* y *dest*. La función hará en *dest* una copia de la cadena apuntada por *orig*, pero sustituyendo las letras minúsculas que aparezcan por letras mayúsculas. La función no devuelve resultados.

Problema 4.- (**1 punto**) Programar una función llamada *strcat* que tome como parámetros dos punteros a cadena: *orig* y *dest*. La función hará una copia de la cadena apuntada por *orig* a partir de donde termina la cadena apuntada por *dest*, efectuando así la concatenación de ambas cadenas. La función no devuelve resultados.

Problema 5.- (**0.5 puntos**) Programar una función llamada *isdigit* que tome como parámetro un carácter ASCII. Si el carácter dado es numérico (un dígito del 0 al 9), la función debe devolver un valor entero 1. En caso contrario devolverá un valor 0. La comprobación debe ser compacta y eficiente.

Problema 6.- (**0.5 puntos**) Programar una función llamada *dtoi* que tome como parámetro un carácter ASCII numérico (un dígito del 0 al 9) y devuelva un entero con el valor entero correspondiente a ese dígito. Ejemplo: si se le da como parámetro el carácter ASCII '3' (valor numérico 51_{hex}) debe devolver un entero de valor 3.

Se debe emplear la función *isdigit* del ejercicio anterior para comprobar que el carácter dado es un dígito válido. En caso de error se debe devolver un valor numérico -1.

Problema 7.- (**1.5 puntos**) Programar una función llamada *strrev* que reciba un puntero a cadena e invierta el orden de los caracteres de dicha cadena. No devuelve resultados. Emplear esta función en un programa principal sencillo para comprobar su funcionamiento. Existen varias estrategias válidas para conseguir este comportamiento:

a) La función almacena uno a uno los caracteres en la pila, y seguidamente los recupera uno a uno y los va almacenando en memoria en el orden inverso al que tenían originalmente.
b) La función encuentra dónde está el último carácter (no nulo) de la cadena y obtiene así un puntero al último carácter. A partir de ahí lleva dos punteros que apuntan a los caracteres que hay que intercambiar entre sí. El intercambio de caracteres se hace usando la función *swap* del ejercicio anterior (convenientemente modificada para que maneje datos de tamaño byte).

PROBLEMAS DE COMPLEJIDAD MEDIANA

Problema 8.- **(2 puntos)** Programar una función llamada *atoi* que reciba como parámetro un puntero a cadena. Esa cadena se supone formada por caracteres ASCII numéricos. La función debe calcular el valor del número representado por dicha cadena y devolver ese valor como entero. Si se produce un error, se debe devolver el valor -1. Se debe utilizar la anterior función *isdigit*.

Problema 9.- **(3 puntos)** Programar una función llamada bsort que implemente el algoritmo de ordenación de la burbuja. Recibe como parámetros:

- Un puntero *p* a un vector de enteros de tamaño Word.
- Un entero *N*.

La función debe emplear dos bucles anidados para implementar el algoritmo de la burbuja (*bubble sort*). No devuelve resultados. Véase un ejemplo:
https://www.youtube.com/watch?v=MtcrEhrt_K0

PROBLEMA DE COMPLEJIDAD ALTA

Problema 10.- "Calculadora". Programar un programa que pida al usuario una operación aritmética. Esta operación se recibirá en forma de cadena de caracteres. El programa evaluará la expresión introducida y presentará por pantalla el resultado del cálculo. Se recomienda hacer un diseño incremental, partiendo de un funcionamiento básico y evolucionando en versiones subsiguientes hacia funcionamientos más complejos. Puntuación:

- 3 puntos si responde correctamente a las cuatro operaciones básicas. En esta primera versión se permite que los números introducidos tengan que ser de un solo dígito y que solamente se introduzca una operación a la vez. Ej:

```
Introduzca una operación:
> 2*7
= 14
Introduzca una operación:
```

- +1 punto si admite varias operaciones encadenadas. En esta versión todas las operaciones se ejecutarán de izquierda a derecha sin tener en cuenta orden de precedencia. Ej:

```
Introduzca una operación:
```

```
> 2+1*5
= 15
Introduzca una operación:
```

Pista: Una vez introducida la cadena, recorrerla con un puntero+índice. Mediante bloques de ejecución condicional, actuar de manera diferente según si se encuentra un dígito, un operador o un espacio.

- +1 punto si muestra un mensaje de error cuando la cadena introducida no está bien formada (por ej. contiene algún signo no soportado por el programa, o tiene sintaxis inadecuada). Ej:

```
Introduzca una operación:
> 2+1 3*5
Error. Introduzca una operación:
> 3*3ñ
Error. Introduzca una operación:
> 34/2
= 17
Introduzca una operación:
```

- +1 punto si además admite números de varias cifras. Se sugiere haber realizado antes el problema 17 y utilizar aquella función en este programa.
- +2 puntos si admite paréntesis y los procesa correctamente (un solo nivel de paréntesis, no anidados).
- +1 punto si admite paréntesis anidados y los procesa correctamente.
  Pista: tomar la cadena que queda entre los paréntesis y procesarla en una llamada recursiva.
- +2 puntos si tiene en cuenta orden de precedencia de operaciones: multiplicación/división se deben operar antes que suma/resta.
- Retos adicionales fuera de puntuación:
  a) Introducir operaciones nuevas, p.ej. potenciación o raíz cuadrada.
  b) Añadir funciones trigonométricas u otras funciones científicas.
  c) Añadir funcionalidad de memoria (como la de calculadoras tradicionales: memorizar el resultado de la operación anterior y posteriormente recuperarlo a petición del usuario).
  d) …