Práctica 1. Introdución al lenguaje ensamblador de MIPS

Primera parte

1. Descargue el simulador *MARS* de la dirección
   https://courses.missouristate.edu/KenVollmar/MARS/MARS_4_5_Aug2014/Mars4_5.jar

2. Arranque el simulador *MARS* y explore brevemente las posibilidades de los menús.

3. En la opción *Settings* active las opciones *Addresses displayed in hexadecimal*, *Values displayed in hexadecimal* y *Permit extended (pseudo) instructions and formats* y desactive las demás.

4. Arranque el simulador *MARS* y teclee en la pantalla de edición (no se recomienda copiar y pegar) el siguiente programa:

```
.txt
Vector: .wor 1 8,9,10,14,16; 23 25 31, 32
.data
main:   addi $s0, $cero, 10
        add  $s2, $zero, $zero
        addi $s3, $zero $zero
        la   $s1, vector
Bucle:  sll  $t1, $s2, 2
        addi $t1, $t1, s1
        lw   $t0, 0(tl)
        add  $s3, $s3 $t0
        add  $s2, $s2,1
        bne  $s2, $s0,bucle
        li   $V0 10            # exit
        syscal
```

5. Con la opción *Save as...* del menú *File* grabe el programa con el nombre `Ejemplo1.asm`

6. Compile el programa con la opción *Assemble* del menú *Run*, pulsando la tecla de función `F3` o pinchando en el icono que tiene dos herramientas cruzadas.

7. Observará algunos mensajes de error que son debidos a que el programa tiene errores. Trate de corregirlos editando de nuevo el programa. Consulte para ello el documento *Resumen ensamblador MIPS32*. Una vez corregidos, compílelo de nuevo. Trate de descifrar el código de colores de la ventana de edición.

8. Cuando haya corregido todos los errores, observe la ventana *Text Segment* donde se representa la zona de memoria donde está el código, de izquierda a derecha: las direcciones de memoria, su contenido, que es la codificación de las instrucciones en lenguaje máquina, las instrucciones y el programa original.

9. Ejecute el programa paso a paso mediante la tecla `F7` o la opción *Step* del menú *Run*. Observe la evolución de los registros, en la parte derecha de la pantalla (pestaña *Registers*), y de los datos de memoria en la ventana *Data Segment*. Puede ejecutar todo el programa de forma continua mediante la tecla `F5` o la opción *Go* del menú *Run*.

10. A la vista de lo observado en el paso anterior: ¿Puede deducir el propósito del programa? ¿Cuantas iteraciones realiza?

11. Seleccione la opción *Reset* del menú *Run* y ejecute ahora de nuevo el programa de forma continua, pulsando `F5` o la opción *Go* del menú *Run* . Cuando se observa la consola (en la parte inferior, pestaña *Run I/O*) ¿Por qué parece que el programa no hace nada?

Segunda parte

Escriba pequeños programas en ensamblador de MIPS para los siguientes casos:

1. Suponiendo que las variables `f`, `g` y `h` están asignadas a los registros `$s3` a `$s5` respectivamente, escriba en MIPS el código correspondiente a la compilación de la instrucción de alto nivel `f = g-(f+h-3);` Antes de ese código añada las instrucciones necesarias para dar a las variables los valores siguientes: `f` = 8, `g` = 2, `h` = 1. Compruebe el contenido del registro resultado tanto en hexadecimal como en decimal (para ello puede activar y desactivar la opción *Values displayed in hexadecimal*). Cambie los valores iniciales de las variables para comprobar el funcionamiento del programa, incluya algunos valores negativos.

2. Supuesto que los nombres anteriores corresponden a variables almacenadas en la sección de datos de la memoria, escriba el programa en MIPS que acceda a los datos de memoria y deje el resultado en la posición de memoria correspondiente a la variable `f`.

3. Repita el ejercicio 2 con la instrucción de alto nivel `f = 8*(g+h)+4*(f+h)-2*(h-f);` Para multiplicar, en general, utilice la instrucción sintética `mul`, sin embargo, para multiplicar por potencias de 2 utilice `sll`.

4. Declare e inicialice en la sección de datos dos variables enteras de 32 bits, `a` y `b`, y un vector `c` del miso tipo. Construya un programa en MIPS que calcule la componente 8 del vector de la forma siguiente: `c[8] = a+b+1;`

5. **(Entregable)** Reserve espacio en memoria para un vector de enteros de 32 bits con 100 componentes y escriba un programa que dé valores a dichas componentes según el siguiente algoritmo:

   `c[0] = -5; c[1] = 10; c[2] = -50 ... c[i] = c[i-3]+c[i-2]-c[i-1]`