EJERCICIOS DE PRACTICAS DE ENSAMBLADOR MIPS
ESTRUCTURA / ORGANIZACIÓN DE COMPUTADORES
BOLETÍN Nº 2

Ejercicio 8: (Bucles y condiciones)

Se pretende realizar un programa en lenguaje ensamblador equivalente al siguiente código en C:

```
#include <iostream.h>
int ip(int *x, int *y, int n)
      {
      int i, sum;
      for (i=0, sum=0; i<n; i++, x++, y++)
            sum += (*x) * (*y);
      return sum;
      }

main()
      {
      int x[] = {1, 3, 5, 7};
      int y[] = {2, 4, 6, 8};
      int n = 4;
      cout << ip(x, y, n);
      }
```

*Este programa hace una llamada a la rutina **ip**, pasándole tres argumentos: las direcciones base de los arrays **x** e **y** (que son arrays de enteros en memoria) y el número entero 4.*
*La rutina **ip** calcula el producto interior de estos arrays, es decir, la suma de los productos de los elementos de los dos arrays.*

Estudiar el código ensamblador equivalente (que sería, por ejemplo, el resultado de compilar el código C), comparándolo con el código C y observando cómo trabajan en ensamblador las rutinas condicionales para implementar bucles:

```
# Prácticas ensamblador MIPS
# Estructura de Computadores. 2º ITIS CUM-UEX 2004/2005
# Ejercicio 8

      .text
      .globl main
main:
      la    $a0, x       # $a0 <-- dirección base del array x
      la    $a1, y       # $a1 <-- dirección base del array y
      lw    $a2, size    # $a2 <-- tamaño del array = n
      jal   ip           # invoca rutina ip, guarda en $ra la dir. de vuelta
      move  $a0, $v0     # escribe en consola el resultado (que ip dejó en $v0)
      li    $v0, 1
      syscall
      #----------------------#Fin de la ejecución del programa
      li    $v0, 10
      syscall

ip:   li    $v0, 0       # inicializa sum a 0. $v0 guardará el resultado
                         # que devuelve la rutina
      li    $t3, 0       # $t3: índice de los elementos del array (i)
ip1:  bge   $t3, $a2, ipx# sale cuando i >= n
      lw    $t0, 0($a0)  # $t0 <-- a0[i]
      lw    $t1, 0($a1)  # $t1 <-- a1[i]
      mul   $t2, $t0, $t1
      add   $v0, $v0, $t2# sum <-- sum + a0[i] * a1[i]
      addi  $a0, $a0, 4  # incrementa los punteros
      addi  $a1, $a1, 4
      addi  $t3, $t3, 1  # i++
      b     ip1          # cierra bucle (salto incond. a ip1)
```

```
ipx:   jr     $ra          # retorna al invocador
       #------------------------------#

       .data
size:  .word  4
x:     .word  1, 3, 5, 7
y:     .word  2, 4, 6, 8
```

Realizar las siguientes cuestiones:

a) Comprobar en el simulador, ejecutando el programa paso a paso y observando el contenido de $ra y de PC en la ventana de registros y las direcciones de las instrucciones en la ventana de texto, si se cumple la siguiente afirmación:

> *"Cuando se invoca mediante **jal** la rutina **ip**, la dirección de retorno, que es la de la instrucción siguiente a la instrucción en ejecución (jal), es decir PC+4, es guardada en $ra".*

b) Comprobar en el simulador, observando el cambio en el contador de programa, si se cumple la siguiente afirmación:

> *"Mediante la instrucción etiquetada por **ipx**, se está retornando a la instrucción siguiente a la que hizo la llamada a la rutina **ip**".*

c) Dibujar un diagrama de flujo que describa las operaciones realizadas en el código ensamblador.

Ejercicio 9: (Bucles y condiciones)

Se pretende realizar un programa en lenguaje ensamblador equivalente al siguiente código en C:

```
#include <iostream.h>
main()
       {
       int c = 2;
       char *s;            // s : puntero a una variable tipo char

       switch(c)
              {
              case 0: s = "Badajoz"; break;
              case 1: s = "Caceres"; break;
              case 2: s = "Merida"; break;
              case 3: s = "Plasencia"; break;
              default: s = "Otras"; break;
              }
       cout << s << end1;
       }
```

Este programa incluye la instrucción C switch, que dependiendo del valor del entero c, asigna a la cadena s el nombre de una ciudad. En concreto, como c=2, se tendrá s="Merida".

Estudiar el código ensamblador equivalente comparándolo con el código C y observando cómo se puede implementar en ensamblador la estructura switch usando instrucciones de salto:

```
# Prácticas ensamblador MIPS
# Estructura de Computadores. 2º ITIS CUM-UEX 2004/2005
# Ejercicio 9
              .text
              .globl main
main:
              li     $s0, 2       # Seleccion de ciudad

              bne    $s0, 0, c1
              la     $a0, Badajoz
              b      cx
c1:           bne    $s0, 1, c2
              la     $a0, Caceres
              b      cx
c2:           bne    $s0, 2, c3
              la     $a0, Merida
```

```
            b     cx
c3:         bne   $s0, 3, c4
            la    $a0, Plasencia
            b     cx
c4:         la    $a0, Otras

cx:         li    $v0, 4       # Escribe la ciudad
            syscall

            li    $v0, 10
            syscall

            .data
Badajoz:    .asciiz "Badajoz\n"
Caceres:    .asciiz "Caceres\n"
Merida:     .asciiz "Merida\n"
Plasencia:  .asciiz "Plasencia\n"
Otras:      .asciiz "Otras\n"
```

Comprobar la ejecución del programa en el simulador.

Cuestión: Dibujar un diagrama de flujo que describa las operaciones realizadas en el código ensamblador.

Ejercicio 10: (Bucles y condiciones)

Realizar un programa en ensamblador del MIPS que calcule la suma o resta de dos vectores especificados como datos. La dimensión de los vectores también debe especificarse como dato. El algoritmo especificado en pseudocódigo es:

```
inicio
      leer (elección)
      llamar_a cálculo
      escribir (r)
fin

procedimiento cálculo
      según_sea elección hacer
            0: r <-- x + y
            1: r <-- x - y
            si_no: r <-- (0,0, ... ,0)
      fin_según
fin_procedimiento
```

*Aquí la estructura **según_sea** es equivalente a la instrucción C **switch**, **leer (x)** indica introducir el valor de **x** por teclado y **escribir (r)** indica escribir todas las componentes del vector **r** por consola. Las operaciones "**x+y**" y "**x-y**" corresponden a suma y resta de vectores. La rutina **cálculo** debe usar un procedimiento iterativo.*

Ejecutar el programa en el simulador SPIM y comprobar su funcionamiento correcto en los distintos casos posibles, tanto para vectores de dimensión 4 como de dimensión 5.