Práctica 15. Subrutinas.

1. Introducción

El diseño de un programa puede simplificarse si se utilizan adecuadamente las subrutinas. Éstas permiten dividir un problema grande en otros más pequeños y evitan tener que repetir fragmentos de código.

En esta práctica se especificarán los mecanismos e instrucciones que dispone el lenguaje ensamblador del procesador MIPS para llevar a cabo las tareas relativas a la codificación, llamada y retorno de las subrutinas.

2. Instrucciones para la llamada y retorno de una subrutina

Hay dos instrucciones que intervienen en el proceso de llamada y retorno de la subrutina:

- Llamada a la subrutina

Se denomina “llamada a la subrutina” a la ejecución de una instrucción que indica al simulador que interrumpa el flujo de la ejecución del programa (de forma análoga a como lo hacen las instrucciones de salto incondicional) para continuar en otro punto, y además preserve la dirección de la instrucción siguiente al punto donde se llamó a la subrutina, de modo que cuando ésta termine, devolverá el control del programa al lugar desde donde fue llamada. De ello se encarga la instrucción `jal`:

Jump and Link: `jal etiqueta`
- La dirección de memoria de la siguiente instrucción a **`jal etiqueta`** se almacena automáticamente en el registro 31 (`$ra`, *return address*).
- El control de flujo pasa a la instrucción cuya dirección es `etiqueta` (comienza la ejecución de la subrutina cuyo nombre es `etiqueta`).

- Retorno de la subrutina

El retorno desde una subrutina consiste en, una vez finalizada ésta, indicarle al simulador que continúe la ejecución del código a partir de la instrucción siguiente a la de llamada a la subrutina (esta dirección ha sido convenientemente almacenada en el registro `$ra` por la instrucción `jal`). De esto se encarga la instrucción `jr`:

Jump Register: `jr $registro`
Salta incondicionalmente a la dirección contenida en `$registro`. Como se ha dicho, el registro que contendrá la dirección de retorno de la subrutina es `$ra`, de modo que el formato que se empleará cuando se utilice esta instrucción será **`jr $ra`**.

El hecho de que la dirección de retorno se almacene en el registro `$ra` implica que no se pueden construir subrutinas anidadas. Aparentemente, pues, resulta imposible la recursividad, pero con la estructura de memoria conocida como "pila" podemos saltarnos esta limitación.

2. Paso de parámetros y devolución de resultados

El ensamblador del MIPS establece el siguiente convenio para realizar el paso de parámetros.

1. Los 4 primeros parámetros de entrada se pasarán a través de los registros `$a0` - `$a3`. A partir del quinto parámetro, deberán pasarse a través de la pila.

2. Los 2 primeros parámetros de salida se devuelven a través de los registros `$v0` - `$v1`; el resto, a través de la pila.

Para pasarle hasta cuatro parámetros a una subrutina, simplemente se colocarán en los registros `$a0`, `$a1`, `$a2`, `$a3` antes de realizar la llamada; esto es similar al mecanismo que se emplea para realizar las llamadas al sistema.

Sólo queda decidir si se pasarán los parámetros por valor o por referencia, un concepto idéntico al visto para los lenguajes de alto nivel. El paso por valor implica que el argumento es un valor con el que se desea realizar alguna operación. El paso por referencia significa que el argumento es en realidad una dirección de memoria en la que se almacenan los datos que se necesitarán para realizar los cálculos.

A la hora de devolver resultados, la subrutina los colocará en los registros `$v0` y `$v1`. Cuando la subrutina finalice, habrá que tomar la precaución de salvar estos valores en otros registros o en la memoria, si fuera necesario.

3. Ejemplo

```
      .data
cad:  .asciiz "\nIntroduce un número: "
cadr: .asciiz "\nEl resultado es: "

      .globl main
      .text
main:

      li $v0, 4              #Imprimo cadena
      la $a0, cad
      syscall

      li $v0, 5              #Leo número
      syscall
      move $t0, $v0          #Lo guardo en $t0

      li $v0, 4              #Imprimo cadena
      la $a0, cad
      syscall

      li $v0, 5
      syscall
      move $t1, $v0          #Leo número y lo guardo en $t1


      move $a0, $t0          #Como hay 2 argumentos van en $a0 y $a1
      move $a1, $t1
      jal suma               #Llamo a la función
      move $t0, $v0          #Guardo el resultado en $v0

      li $v0, 4              #Imprimo cadena
      la $a0, cadr
      syscall

      move $a0, $t0          #Imprimo número
      li $v0, 1
```

```
      syscall

      li $v0, 10
      syscall

suma:                         #Función suma
      add $v0, $a0, $a1       #Le llegan los argumentos en $a0 y $a1
      jr $ra                  #Cuando acaba tengo el resultado en $v0
```

4. Ejercicios

1. Realice un programa que solicite al usuario un número entero ‘n’ y que calcule el factorial de dicho número, haciendo uso de una subrutina llamada “factorial”. El argumento que se debe pasar es el número indicado por el usuario.

2. Escriba un programa que solicite al usuario una cadena de texto y que posteriormente la copie en otra cadena imprimiendo el resultado por pantalla. Se le debe pasar a la subrutina “strcpy” dos argumentos: un puntero a la cadena original y un puntero a la cadena destino. NOTA: Recordad que las cadenas terminan con el carácter ‘\0’ que tiene el valor $zero.