Práctica 15. Subrutinas

Objetivos

- Comprender el uso de subrutinas en ensamblador.

1. Introducción

Además de las instrucciones de salto condicional o incondicional tratadas en la práctica anterior, hay un tercer tipo de instrucciones de salto. Son las instrucciones de **llamadas a subrutinas.**

En general, las instrucciones de salto harán referencia a otras instrucciones a las que se quiera "saltar". Para hacer referencia a instrucciones del segmento de texto se utilizarán únicamente **etiquetas** que precedan a las instrucciones de destino (las cuales estarán en el segmento de texto).

2. Subrutinas

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.

Como a una función se la puede llamar desde distintas partes del código, se tiene que almacenar la dirección de retorno (la siguiente a la que llamó a la función) para saber dónde volver cuando se termine de ejecutar la función.

Por tanto, 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 para continuar en otro punto, y además preserve la dirección de la instrucción siguiente a la instrucción 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 x1, funcion` → salta a `funcion` y guarda en `x1` la dirección de retorno de la función.

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 `x0` por la instrucción `jal`). De esto se encarga la instrucción `jalr x0, x1, 0` o `jalr x0, 0(x1)`o las pseudoinstrucciones `jr rs` o `ret`.

Por tanto, los pasos para llamar a una función son:

1. Poner los argumentos en los registros `x10-x17` o en la pila si hay más.
2. Saltar a la función (`jal x1, funcion` dejando la dirección de retorno en `x1`; *jump and link*) → La dirección de retorno será `PC + 4`
3. Reservar espacio de memoria requerido por la función, almacenando los registros que se vayan a utilizar.
4. Realizar la tarea de la función.
5. Poner el resultado en los registros `x10-x17` o en la pila si hay más, restaurar los registros y liberar memoria.
6. Retornar el control al punto de origen (`ret` o `jalr x0, 0(x1)` siendo `x1` la dirección de retorno y `x0` el valor 0).

2.1. Paso de parámetros y devolución de resultados

Los argumentos de las subrutina se pasan por los registros `a0`, `a1` hasta el `a7`. Si solo tiene un argumento, va en `a0`, si tiene dos, en `a0` y `a1`, si tiene tres, en `a0`, `a1` y `a2`, etc.

Los valores que devuelve la función se devuelven en `a0` y si son dos, en `a0` y en `a1`.

Si hay más argumentos o más resultados, se pasan por la pila; lo veremos en la próxima práctica.

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.

3. Ejemplos

El siguiente programa pide por la consola un número y a continuación lo muestra elevado al cuadrado. Para ello usa la función `cuadrado` que recibe como argumento el número leído y devuelve el número al cuadrado:

```
1   .data # comienzo del segmento de datos
2
3 cad1: .string "Introduzca un número: "
4 cad2: .string "El resultado es: "
5
6   .globl main
7
8   .text # comienzo del segmento de texto (instrucciones)
9
10 main: # Comienza la función main
11
12   li a7, 4 # código de la llamada print_string
13   la a0, cad1 #imprime la cadena que está en la dirección a0
14   ecall
15
16   li a7, 5 # código de la llamada read_int
17   ecall # ahora se introduce un número en la consola y queda en a0
18
19   jal x1, cuadrado # se llama a la función (el argumento está en
       a0)
20   #en x1 queda la dirección de retorno (PC+4)
21
22
23   mv t0, a0   # muevo el resultado (está en a0) a t0 para no
       perderlo
24
25   li a7, 4 # código de la llamada print_string
26   la a0, cad2 #imprime la cadena que está en la dirección $a0
27   ecall
28
29   li a7,1 # código de llamada print_int
30   mv a0, t0 # a0 = número que se quiere imprimir
31   ecall # se realiza la llamada
32
33   li a7, 10 # código de llamada exit
34   ecall # llamada al sistema
35
36 cuadrado:
```

```
37   mul a0,a0,a0 # se eleva al cuadrado y guarda el resultado en a0
38   #Para hacer el return podemos usar una de estas 4 instrucciones:
39   #jalr x1  # salta a la dirección que hay en x1 (la dirección de
       retorno)
40   #ret
41   #jalr x0, x1, 0
42   jalr x0, 0(x1)
```

Otro ejemplo de función que calcula la longitud de una cadena:

```
1    .data
2  cad1: .string "Introduzca una cadena: "
3  cad2: .zero 100
4  cad3: .string "La longitud de la cadena es: "
5    .text
6    .globl main
7  main:
8    li a7, 4
9    la a0, cad1
10   ecall
11
12   li a7, 8
13   la a0, cad2
14   li a1, 100
15   ecall
16
17   la a0, cad2     #Argumento en a0
18   jal x1, funcion   #Llamada a la función
19
20   mv t0, a0 #El resultado queda en a0
21
22   li a7, 4
23   la a0, cad3
24   ecall
25
26   li a7, 1
27   mv a0, t0
28   ecall
29
30   li a7, 10
31   ecall
32
33 funcion:
34   li t0, 0 #contador de caracteres
35 bucle:
36   lb t1, 0(a0) # leo una letra y la guardo en t1
37   beq t1, x0, fin #si es el \0 se ha terminado la cadena
38   addi t0, t0, 1  # sumo uno al contador de la longitud
39   addi a0, a0, 1  # muevo el puntero para que apunte a la letra
       siguiente
```

```
40   j bucle # salto a bucle
41 fin:
42   mv a0, t0  #muevo el contador a a0
43   ret #salgo de la función
```

4. Ejercicio

4.1. Ejercicio

Realice un programa que lea una cadena por teclado y llame a una función que copia esa cadena leída en otra cadena destino. Además calculará cuántas letras 'a' tiene la cadena. Nota: para la lectura de la cadena debe emplearse la llamada al sistema *read_string*. No se puede acceder al segmento de datos, lo que se necesite ha de pasarse como argumento (las dos cadenas y la letra 'a'). La función `main` imprimirá los resultados por pantalla (la cadena copiada y el número de letras 'a').

4.2. Ejercicio

Realice un programa que pida un número por teclado y llame a una función que cuente el número de ceros que tiene cuando se representa en binario. La función `main` imprimirá el resultado por pantalla.

4.3. Ejercicio

Realice un programa que lea dos números por teclado, comprueben que sean positivos y el primero menor que el segundo y si no lo son de un error y los vuelva a leer hasta que lo sean. Además, el programa leerá una cadena y llamará a una función que copia en una cadena destino la subcadena definida por el primer número (posición de partida) hasta el segundo número leído (posición de fin de la cadena). Esa función además imprimirá por pantalla los números que van del primer al segundo número. La función main imprimirá la cadena copiada. NOTA: No se puede acceder al segmento de datos, lo que se necesite ha de pasarse como argumento. Por ejemplo, si lee la cadena `Hola` y las posiciones 1 y 3, imprimirá 1, 2 y 3 y la cadena copiada será `ola`.