**PRÁCTICA 2.4  
Programación en ensamblador MIPS. Procedimientos I**

**Objetivos:**

* Iniciarse en la programación estructurada en ensamblador del MIPS.
* Aprender a manejar los procedimientos: definición, llamada y retorno.
* Manejar las llamadas al sistema operativo desde un programa en ensamblador.

**Desarrollo / Comentario:**

**Paso de parámetros por valor.** Para el paso de parámetros por valor, antes de realizar la llamada a la subrutina, el programa principal carga el valor de los parámetros de entrada en los registros elegidos para ello.

**Paso de parámetros por referencia.** Para el paso de parámetros por referencia, antes de realizar la llamada a la subrutina, el programa principal carga en los registros elegidos para ello las direcciones de memoria en las que está almacenada la información.

* Escribir en ensamblador MIPS32 un programa equivalente al siguiente programa en C.

int main()

{ int a[8]={0x50,0x60,0x70,0,0,0,0,0x090};

int b[8]={1,3,5,7,9,11,13,15};

int c[8]={2,4,6,8,0x10,0x12,0x14,0x16};

int v1[8];

int v2[8];

int sumando;

sumando=Suma\_vectores (a,b,v1,8);

printf(“%d”,sumando);

sumando=Suma\_vectores (b,c,v2,3);

printf(“%d”,sumando);

return 0;

}

int Suma\_vectores (int \*a, int \*b, int \*c, int dim\_vec)

{

int i,sumando=0;

for (i=0; i<dim\_vec; i++)

{

c[i]=a[i]+b[i];

sumando=sumando+c[i];

}

return(sumando);

}

**Nota**:

* Los registros $a0 - $a3 se usarán para pasar los argumentos.
* Los registros $v0 - $v1 se usarán para el retorno de valores.
* Si hay que usar más registros (como argumentos o como retorno) se pasarán por la pila.
* Los registros $t0 - $t9 se usarán como registros temporales y no será necesario que el procedimiento preserve su valor.
* Los registros $s0 - $s7 deberán preservar su valor, es decir el procedimiento, en el caso que los haya modificado, los apilará al comienzo de su ejecución y los desapilará justo antes de retornar.

**Anexo: Llamadas al sistema**

El simulador usado en las prácticas, Simula3MS, ofrece un pequeño conjunto de servicios de llamada al sistema operativo a través de la instrucción de llamada al sistema (syscall).

Para realizar una llamada al sistema (Tabla 1) se cargará en el registro $v0 el código de la llamada y en los registros $a0 y $a1 (o $f12 para valores de punto flotante) los argumentos.

Las llamadas al sistema que devuelven valores ponen sus resultados en el registro $v0 (o $f0 para resultados de punto flotante).

Existen llamadas al sistema para salida de datos:

* *print\_int* muestra en el diálogo como un entero el valor entero que se le pasa
* *print\_float* muestra en el diálogo el valor en punto flotante que se le pasa.
* *print\_double* hace lo mismo que print\_float pero permite un rango más amplio de números en punto flotante.
* *print\_string* muestra en el diálogo la cadena almacenada a partir de la dirección que indica $a0 hasta que encuentra el carácter de finalización de cadena.

También hay llamadas al sistema para entrada de datos, estos datos se solicitan al usuario a través de diálogos:

* *read\_int* almacena el entero que introduce el usuario en $v0.
* *read\_float* almacena el número de punto flotante que le introduce el usuario en el registro $f0 con formato de simple precisión.
* *read\_double* almacena el número de punto flotante que le introduce el usuario en los registros $f0 y $f1, puesto que el formato de doble precisión necesita dos registros para almacenar un valor.
* *read\_string* almacena la dirección de comienzo de la cadena que introduce el usuario en el registro $a0 y la longitud de dicha cadena en $a1.

Otra llamada al sistema muy utilizada es *exit*, que indica al procesador que se ha terminado la ejecución del programa.

|  |  |  |
| --- | --- | --- |
| **Código** | **Nombre** | **Operación** |
| 1 | *print\_int* | Imprime como un entero aquello que se encuentra en $a0. |
| 2 | *print\_float* | Imprime como un número en punto flotante de  aquello que se encuentra en $f12. |
| 3 | *print\_double* | Imprime como un número en punto flotante  aquello que se encuentra en $f12 y $f13, considerando  que forman un sólo registro de doble precisión. |
| 4 | *print\_string* | Imprime como un string aquello que se encuentra  en la posición indicada por $a0. |
| 5 | *read\_int* | Solicita un entero que se almacenará en el registro $v0. |
| 6 | *read\_float* | Solicita un número en punto flotante que se almacenará en el registro $f0 con formato de simple precisión |
| 7 | *read\_double* | Solicita un número en punto flotante que se almacenará en el registro $f0 (y $f1) con formato de doble precisión. |
| 8 | *read\_string* | Solicita un string que se almacena en $a0 y cuya longitud se guarda en $a1. |
| 10 | *exit* | Finaliza la ejecución. |

Tabla 1: Códigos asociados a *syscall*

**Ejemplo sencillo:**

.data

str: .asciiz "Hola Mundo!!!"

.text

.globl main

main:

li **$v0**, **4**

la **$a0, str**

syscall # print string

li **$v0**, **10**

syscall # exit