PRÁCTICA 5
PROGRAMACION EN ENSAMBLADOR RISC -V. SUBRUTINAS

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.
- Los convenios de paso de argumentos a subrutinas, así como de recogida de resultados.
- Instrucciones: **jal, jalr**
- Pseudoinstrucciones: **call, ret**

Subrutinas y llamadas:

Se conoce como subrutina a un fragmento de código que podemos invocar desde cualquier punto del programa (incluyendo otra o la propia subrutina), retomándose la ejecución del programa, en la instrucción siguiente a la invocación, cuando la subrutina haya terminado.

Las subrutinas se usan para simplificar el código y poder reusarlo, y son el mecanismo con el que los lenguajes de alto nivel implementan procedimientos, funciones y métodos.

Una subrutina puede aceptar parámetros (argumentos) que modifiquen su comportamiento, exactamente igual que una función de alto nivel. Por convenio en RISC-V, estos se colocan en los registros de argumento a0-a7. El valor de retorno de una subrutina se debe colocar en a0, permitiéndose además devolver un segundo valor en a1.

Ejemplo:

```
Programa principal      Subrutina 1          Subrutina 2
main :                  recorre:             mira:
   [... ]                  [... ]               [... ]
   call recorre            call mira            ret
  [... ]                   [... ]
                           ret
```

Como se puede observar, para el control de llamadas a funciones en RISC-V existen dos pseudoinstrucciones clave:

- **call etiqueta:** Calcula la dirección de destino de la etiqueta, y salta a la misma para iniciar la función correspondiente. Además, guarda la dirección de retorno en ra.
  ra ←PC+4
  PC ← etiqueta

.

- **ret**: Recupera el retorno de ra y continua en la instrucción siguiente a la de llamada de la función: PC ←ra

Hay un detalle con el que tener especial cuidado. Para realizar operaciones es posible que la subrutina modifique algunos registros y que en alguno tengamos guardado algún dato que necesitemos a la vuelta de la subrutina. Es por eso por lo que hay un convenio de uso de registros que es importante seguir:

- Los registros se dividen, a rasgos generales, en temporales t0-t6, argumento a0-a7 y salvados s0-s11.
- Cuando una subrutina tiene argumentos, se colocan en orden en los registros de argumento a0-a7, para posteriormente llamarla.
- Cuando una subrutina devuelve un valor, éste volverá en los registros a0,a1. Debemos recogerlo de ahí y guardarlo.
- Si la subrutina define más argumentos o resultados que los registros disponibles para ello entonces usará la pila para pasar los argumentos o devolver el resultado.
- Cualquier subrutina debe preservar, a su retorno, todos los registros salvados. Es libre de modificar los temporales y los de argumento
  Si modifica algún registro salvado, debe guardarlo en la pila al comenzar, y recuperarlo de la pila al terminar.
- Si además la subrutina llama a otra, también deberá guardar el registro ra en la pila.

Escribir en ensamblador RISC-V un programa equivalente al siguiente programa en C. Usaremos el ejercicio realizado en el apartado 3 de la práctica anterior.

```
int main()

{            int V1[8]= { 1, 2, 3, 0x400, 51, 6, 7, 8 };
             int V2[6] V2 = { -1, 3,5,-4,5,0x500};

             int mayor1= nummayor(V1,8);
             printf("%x",mayor1);

             int mayor2 = nummayor(V2, 6);
             printf("%x",mayor2);

}
int nummayor(int V[], int tama)
{
              int i, mayor=V[0];
              for (i = 1; i < tama; i++)
                 {
                     if (V[i]>mayor) mayor=V[i];

                 }
              return (mayor);
  }
```

```
            .data

V1:         .word 1,2,3,0x400,51,6,7
V2:         .word -1,3,5,-4, 5,0x500
mayor1:     .word 0
mayor2:     .word 0
            .text
            .globl main

main:       la a0, V1                   #Cargamos la dir. Memoria de V1 en a0
            la a1, 8                    #Cargamos el número de elementos
            call rutmayor               #Llamada a subrutina
            la t0, mayor1               #Cargamos 0 en t0
            j bucle                     #Vamos al bucle
            sw a0, 0(t0)                #Guardamos en a0 el valor de t0
            addi a7,zero,34             #Imprimimos
            ecall

finmain:    addi a7,zero,10
            ecall

rutmayor:   lw t0, 0(a0)                #Cargamos en t0 el valor de a0
            addi a1, a1, -1             #Restamos uno a a1, que es el contador

bucle:      addi a0, a0, 4              #Tomamos el siguiente valor de la cadena
            beq a1, zero, finrutmayor   #Si el contador se queda a 0, salto fin
            lw t1, 0(a0)                #Cargo el valor actual de la cadena
            blt t1, t0, salto           #Si el valor de t1 es menor que t0,
                                        #va al salto
            add t0, zero, t1            #Si es mayor, se queda ese valor

salto:      addi a1, a1, -1             #En el salto avanza al siguiente
            j bucle                     #Vuelve al bucle

finrutmayor:add a0, zero, t0            #Acaba la subrutina
            ret
```