PRÁCTICA 1: SUBRUTINAS

Resumen de teoría de subruitnas:

- Llamada a subrutina: jal ProcedureAddress #jump and link

  Guarda PC+4 en el registro $ra para tener un enlace a la siguiente instrucción para poder retornar de la subrutina

- Retorno de subrutina: jr $ra #return
- El programa que llama a la subrutina pone los argumentos en 4 regstros $a0 - $a3
- La subrutina pone los resultados en los registros de valores $v0 - $v1 para que el programa que la ha llamado pueda leer los resultados
- Qué pasa si el programa que llama a la subrutina necesita usar más registros que los existentes para argumentos y valores retornados? → Utilizaremos la pila
- Para llamadas anidadas, el programa que llama a la subrutina (*caller*) necesita salvar en la pila:
  - Su dirección de retorno
  - Argumentos y reg. temporales que necesite utilizar
- Restaurar desde la pila antes del retorno

TRABAJO PREVIO:

1. Implementa en C una función, int compara(A,B) que dados dos números naturales A y B calcule si A es mayor que B. Si A es mayor que B ***valor*** devuelve A-B. En cambio si A es igual a B devuelve en ***valor*** 0 y si A es menor que B en ***valor*** devolverá A. El return de la función devolverá ***valor***.

int compara(A,B){

}

2. Suponiendo que tienes implementada la función anterior compara(A,B), el siguiente programa principal utilizará la función compara para calcular A ***mod*** B, teniendo en cuenta que A mod B es el resto de dividir A por B. En este ejemplo no utilizaremos la división sino la función compara(A,B).

```
main() {
int A, B, modulo, mayor;
A=10;  // estos valores los podréis cambiar
B=6; // estos valores los podréis cambiar
modulo=A;
do{
modulo=compara(modulo,B);
}while (B≤modulo)
}
```

Traduce a ensamblador el programa main().

3. Supongamos el siguiente código en C, igual que el problema 4 de la colección:

```
main(){
leaf_function(1);
}
int leaf_function(int f);{
int result;
result=f+1;
if(f>5)
        return result;
leaf_function(result);
}
```

¿Es una función recursiva?

Recuerda que se ha de guardar también en la pila la @ de retorno, $ra; así como los argumentos $a0 y $a1 si es necesario cada vez que llamamos a la función.

3.1. Tradúcelo a ensamblador:

TRABAJO EN LABORATORIO:

1. Traduce a ensamblador la subrutina compara(A,B) e impleméntala en el simulador junto con el programa main también en ensamblador. Simúlalo paso a paso. NOTA: Aunque no sea una rutina que llame a otra ni tampoco recursiva seguiremos la metodología de al entrar en una subrutina guardar en la pila todos los registros $s0-s7 que vayamos a utilizar. Además si fuese una subrutina que llama a otra o a sí misma ha de guardar también en la pila la @ de retorno, $ra; así como los argumentos $a0 y $a1 si es necesario, por ejemplo en una recursiva.

    1.1. Escribe aquí la rutina compara(A,B) en ensamblador:

    1.2. Prueba con diferentes valores de A y B: A=10, B=6, A=10, B=2, A=10, B=3 y A=2 y B=5 y cuando funcione avisa al profesor.

2. A partir del ejercicio 3 del trabajo previo implementa el main y la subrutina en el simulador. *En el archivo ejercicio3.s* debes completar la traducción. Simúlalo paso a paso.

    2.1. Si el programa main llama a la subrutina con un valor inicial de 1, como está en el enunciado en C, describe la ejecución dinámica del programa, ayúdate simulando paso a paso en el simulador. Avisa al profesor cuando funcione.