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, potencia(int x, int y) (Asumimos que x puede ser positiva o negativa, pero y siempre es positivo). En esta función x se multiplica por si mismo y veces. Si y=0 la función debe devolver 1. No debe ser una función recursiva sino iterativa

```
int potencia(x,y){




}
```

2. Suponiendo que tienes implementada la función anterior potencia(x,y), el siguiente programa principal utilizará la función potencia para calcular ax^2+bx+c.

```
main() {
register int A, B, C,x;
int y;
A=10;  // estos valores los podréis cambiar
B=6; // estos valores los podréis cambiar
C=-1; // estos valores los podréis cambiar
x=3; // estos valores los podréis cambiar
      y=a*potencia(x,2)+b*x+c; //guardar en memoria el resultado

}
```

Traduce a ensamblador el programa main().

3. Supongamos el siguiente código en C de una función recursiva que también calcula la potencia (Asumimos que x puede ser positiva o negativa, pero y siempre es positivo):

```
int potencia_recursiva(int x,int y) {
int result;
   if(y==0) result=1;
   if(y==1) result=x;
   else {
   result = potencia_recursiva (x ,y/2) * potencia_recursiva(x ,y-
   y/2);
   }
return result;
}
```

Donde la llamamos con un main:

```
main(){
int result;
register int x=2;
register int y=5;
   result=potencia_recursiva(2,5);//guardar en memoria
}
```

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 rutina potencia(x,y) e impleméntala en el simulador junto con el programa main de la pregunta 2 del informe previo 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 potencia(x,y) en ensamblador:

    1.2. Prueba con diferentes valores de A,B,C y X: A=10, B=6, C=-1, X=3, A=-3, B=2, C=1, X=1, A=10, B=-3, C=2 y X=0; cuando funcione avisa al profesor.

2. A partir del ejercicio 3 del trabajo previo implementa el main y la subrutina en el simulador.

a) En el archivo ejercicio3.s debes completar la traducción. Simúlalo paso a paso.

b) Si el programa main llama a la subrutina potencia_recursica con unos valores iniciales de x=2 e y=5, 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.