Lab 6 – Bytecode EVM y consumo de gas

1. Bytecode EVM, consumo de gas

Dado el siguiente contrato:

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.4.26;
contract lab6 {
  uint[] arr;
  uint sum;
  function generate(uint n) external {
    for (uint i = 0; i < n; i++) {
        arr.push(i*i);
    }
  }

  function computeSum() external {
    sum = 0;
    for (uint i = 0; i < arr.length; i++) {
      sum = sum + arr[i];
    }
  }
```

1. Genera el código ensamblador del contrato y guárdalo en un fichero de texto.

2. Considera solamente el código de ejecución (runtime code). **Dibuja <u>a mano</u>** el grafo **CFG** de `computeSum()`. Numera los nodos del grafo con las etiquetas generadas por el compilador (`tag 1`, `tag 2`, etc.). Si el compilador no genera etiqueta para ese bloque, utiliza la etiqueta del nodo inmediatamente anterior con un sufijo 'b', 'c', etc. Marca aquellos nodos que terminan con una instrucción `REVERT`, `RETURN`, `STOP` o `INVALID`.

3. Localiza las instrucciones que acceden a *storage*. Indica cuál es el objetivo de cada una de estas instrucciones en el programa. Indica en el CFG en qué nodos aparecen estas instrucciones.

   NOTA: Los comentarios que añade el compilador a la derecha del código en ensamblador son orientativos pero ten en cuenta que pueden ser imprecisos o incluso inducir a errores de interpretación.

4. Ejecuta `computeSum()` con un array de 100 elementos para obtener su **coste de ejecución** en gas (**execution cost**).

¿Se podría modificar el código Solidity de `computeSum()` para reducir el número de accesos a *storage*? Utiliza el CFG para proponer una versión mejorada de `computeSum()` **sin utilizar bloques assembly en Yul** con un consumo mínimo de gas. Determina su coste de ejecución en gas para un array de 100 elementos, compáralo con el coste de la versión original y explica el motivo por el que el coste varía. Solo se puede modificar el código de `computeSum()`, y se debe suponer que el array puede contener cualquier contenido (podría contener datos diferentes a los producidos por `generate()`).

IMPORTANTE: El coste de ejecución puede variar si se aumenta el número de funciones del contrato y si se ejecutan varias transacciones en el mismo contrato desplegado. Para obtener el coste de ejecución de forma precisa, crea otro contrato solo con las funciones `generate(n)` y `computeSum()` y despliégalo de nuevo cada vez que vayas a evaluar el coste de una función, y ejecuta en cada test solamente `generate(n)` y la función a evaluar.