PRACTICA 6 BLOCKCHAIN

Alejandro Ramírez y David Seijas

March 2022

1 Código

```
// SPDX-License-Identifier: GPL-3.0
//ALEJANDRO RAM RES Y DAVID SEIJAS
pragma solidity ^0.4.0;
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];
        }
    }
}
```

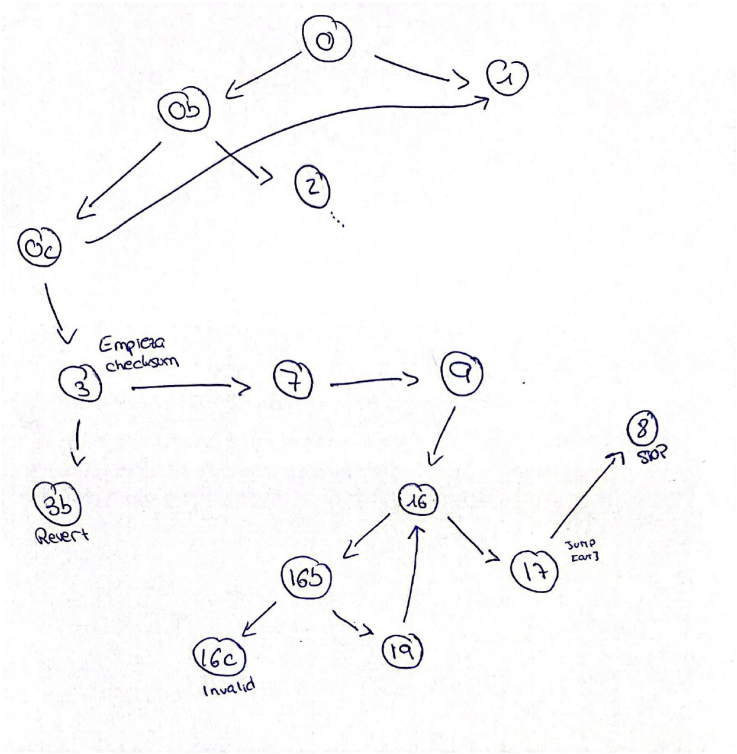
2 ByteCode EVM, consumo de gas

2.1 Ejercicio 1

Compilamos el código de arriba y obtenemos el bytecode almacenado en el fichero bytecode.txt

2.2 Ejercicio 2

El runtime del bytecode está guardado en el fichero runtimecode.txt El **CFG** es el siguiente



Scanned with CamScanner

2.3 Ejercicio 3

Hemos encontrado las instrucciones que acceden a storage de los nodos del CFG calculado en el ejercicio anterior. En el primer caso, como seguir el código no ha sido complicado, ponemos la traducción de la instrucción. El resto sería de forma análoga siguiendo todas las instrucciones que genera el compilador.

Instrucción	Nodo	Explicación
SSTORE	9	storage[1] = 0, es decir, sum = 0
SLOAD	16	cargar arr.length apara comprobar salida del bucle
SLOAD	16b	cargar arr.length para comprobar que el elemento es accesible
SLOAD	19	carga arr[i] en la pila
SLOAD	19	carga sum en la pila
SSTORE	19	actualiza valor de storage[1], es decir, de sum

2.4 Ejercicio 4

El coste de computeSum() es de 312481 gas.

- Una primera mejora sería utilizar dos variables locales que se almacenan en memoria: aux, para llevar la suma del array y lenarray para evitar acceder a array.length en cada iteración. Coste: 281107
- La segunda mejora consiste en copiar el array en otro localizado en memoria. Así evitamos que en cada iteración se acceda a storage por arr[i] y también que acceda a arrr.length para comprobar que realmente i está en rango. Coste: 274217

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity ^0.4.0;
3  contract aux2 {
4      uint[] arr;
5      uint sum;
6
7      function generate(uint n) external {
8          for (uint i = 0; i < n; i++) {
9              arr.push(i*i);
10         }
11     }
12
13      function computeSum() external {
14         uint auxvar = 0;
15         uint[] memory arrAux = arr;
16         uint lenarray = arrAux.length;
17         for (uint i = 0; i < lenarray; i++) {
18             auxvar = auxvar + arrAux[i];
19         }
20         sum = auxvar;
21     }
22 }
```

3 Programación con Yul

3.1 Ejercicio 5

```
1
2  // SPDX-License-Identifier: GPL-3.0
3  pragma solidity ^0.4.0;
4  contract lab6 {
5      function maxMinMemory(uint[] memory arr) public pure returns (
           uint maxmin) {
6          assembly{
7              function fmaxmin (array_pointer) -> maxVal, minVal
8                  {
9                      let len := mload(array_pointer)
10                     let data := add(array_pointer, 0x20)
11                     maxVal := mload(data)
12                     minVal := mload(data)
13                     let i := 1
14                     for {} lt(i,len) {i:= add(i,1)}
15                     {
16                         let elem := mload(add(data,mul(i,0x20)))
17                         if gt(elem,maxVal) { maxVal := elem }
18                         if lt(elem,minVal) { minVal := elem }
19                     }
20                 }
21
22             let max, min := fmaxmin (arr)
23             maxmin := sub(max,min)
24         }
25     }
26
27
28 }
```

3.2 Ejercicio 6

```
1
2  // SPDX-License-Identifier: GPL-3.0
3  pragma solidity ^0.8.0;
4
5  contract lab6ex6 {
6      uint[] public arr;
7
8      function generate(uint n) external {
9          // Populates the array with some weird small numbers.
10         bytes32 b = keccak256("seed");
11         for (uint i = 0; i < n; i++) {
12             uint8 number = uint8(b[i % 32]);
13             arr.push(number);
14         }
15     }
16
17     function maxMinStorage() public view returns (uint maxmin){
18         assembly{
19             function fmaxmin (slot) -> maxVal, minVal
```

```
20                  {
21                      let len := sload(slot)
22                      mstore(0x0, slot)
23                      let data := keccak256(0x0, 0x20)
24                      //posicion de local memory donde tenemos el
                            dato del que queremos calcular el hash y la
                             longitud de la pos de memoria donde se
                            guarda
25                      maxVal := sload(data)
26                      minVal := maxVal
27                      let i := 1
28                      for {} lt(i,len) {i:= add(i,1)}
29                      {
30                          let elem := sload(add(data,i)) //en storage
                                 la memorya va de 1 en 1
31                          if gt(elem,maxVal) { maxVal := elem }
32                          if lt(elem,minVal) { minVal := elem }
33                      }
34                  }
35
36              let max, min := fmaxmin (arr.slot)
37              maxmin := sub(max,min)
38          }
39      }
40  }
```

3.3 Ejercicio 7

Implementación	Coste
YUL	245026
Implementación simple	314282

En la implementación simple accedemos 5 veces a storage por iteración (length y 4*arr[i]). Con Yul se accede a storage 2 veces al principio (length y maxVal) y tan solo 1 en el bucle (elem). Por lo tanto, al reducir los accesos a storage, el coste es más bajo.