EVM - Some Assembly Required

Alex Bazhenov
Lead Developer, Tally Ho

1. What is EVM Assembly?
2. How to read opcodes to trace a simple transaction.

Why do we care?

Section 1

What is EVM Assembly?

What is the EVM?

What is the EVM?

- EVM, zkEVM, Evmos

What is the EVM?

- EVM, zkEVM, Evmos
- At its core - the EVM is a stack machine.

What is the EVM?

- EVM, zkEVM, Evmos
- At its core - the EVM is a stack machine.
- Most operations consume values from the stack. (ADD, MUL, SUB)

What is the EVM?

- EVM, zkEVM, Evmos
- At its core - the EVM is a stack machine.
- Most operations consume values from the stack. (ADD, MUL, SUB)
- There are exceptions to this. (PUSH1, PUSH2, ..., PUSH32)

The Stack Machine

- Depth of 1024 Items

The Stack Machine

- Depth of 1024 Items
- Each item is a 256-bit word

The Stack Machine

- Depth of 1024 Items
- Each item is a 256-bit word
- During execution, the EVM maintains a transient memory (as a word-addressed byte array), which does not persist between transactions.

The Stack Machine

- Depth of 1024 Items
- Each item is a 256-bit word
- During execution, the EVM maintains a transient memory (as a word-addressed byte array), which does not persist between transactions.
- Contracts contain a Merkle Patricia storage trie (as a word-addressable word array), which associated with the account in question and is part of the global state.

The Stack Machine

- Depth of 1024 Items
- Each item is a 256-bit word
- During execution, the EVM maintains a transient memory (as a word-addressed byte array), which does not persist between transactions.
- Contracts contain a Merkle Patricia storage trie (as a word-addressable word array), which associated with the account in question and is part of the global state.
- Compiled smart contract bytecode executes as a number of EVM opcodes, which perform standard stack operations like XOR, AND, ADD, SUB, etc.

The Stack Machine

- Depth of 1024 Items
- Each item is a 256-bit word
- During execution, the EVM maintains a transient memory (as a word-addressed byte array), which does not persist between transactions.
- Contracts contain a Merkle Patricia storage trie (as a word-addressable word array), which associated with the account in question and is part of the global state.
- Compiled smart contract bytecode executes as a number of EVM opcodes, which perform standard stack operations like XOR, AND, ADD, SUB, etc.
- The EVM also implements a number of blockchain-specific stack operations (More on these later).

The Stack Machine

- Depth of 1024 Items
- Each item is a 256-bit word
- During execution, the EVM maintains a transient memory (as a word-addressed byte array), which does not persist between transactions.
- Contracts contain a Merkle Patricia storage trie (as a word-addressable word array), which associated with the account in question and is part of the global state.
- Compiled smart contract bytecode executes as a number of EVM opcodes, which perform standard stack operations like XOR, AND, ADD, SUB, etc.
- The EVM also implements a number of blockchain-specific stack operations (More on these later).
- Each operation costs a certain number of gas.

Assembly

Assembly

Solidity

```
function setOne() public {
  myVar = 1;
}
```

Assembly

Solidity

Bytecode

```
function setOne() public {
  myVar = 1;
}
```

0x5b01010100819055

Assembly

Solidity

```
function setOne() public {
  myVar = 1;
}
```

Assembly

```
JUMPDEST
PUSH1 0x1
PUSH1 0x0
DUP2
SWAP1
SSTORE
```

Bytecode

0x5b01010100819055

Section 2

Tracing a Transaction

www.evm.codes

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.9;


contract UltraSoundMoney {
    uint totalSupply;

    function setTotalSupply() public {
        totalSupply = 8;
    }
}
```

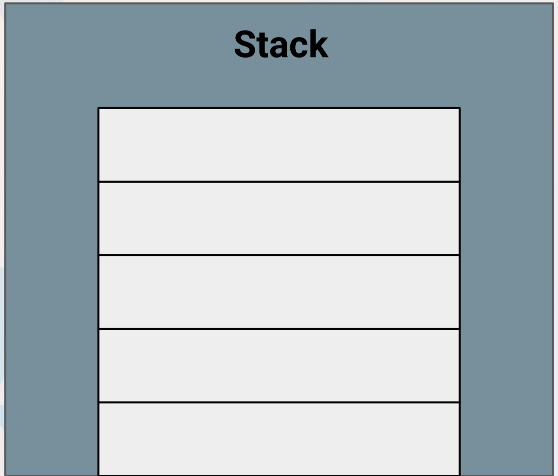
```
solc contracts/UltraSoundMoney.sol --opcodes
```

```
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH1
0x14 JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x75
DUP1 PUSH2 0x23 PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN INVALID
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH1 0xF
JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP PUSH1 0x4
CALLDATASIZE LT PUSH1 0x28 JUMPI PUSH1 0x0 CALLDATALOAD
PUSH1 0xE0 SHR DUP1 PUSH4 0x6057D3EE EQ PUSH1 0x2D JUMPI
JUMPDEST PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH1 0x33 PUSH1
0x35 JUMP JUMPDEST STOP JUMPDEST PUSH1 0x8 PUSH1 0x0 DUP2
SWAP1 SSTORE POP JUMP INVALID LOG2 PUSH5 0x6970667358 0x22
SLT KECCAK256 PUSH1 0xA5 RETURN 0xBD LOG4 0xC1 0xB6 PUSH8
0xD47AC4FEDCFA3F11 PUSH10 0x7B65BE5AD57BD09B3C35 0xEB LOG2
SWAP2 0x5D 0xE3 PUSH5 0x736F6C6343 STOP ADDMOD GT STOP
CALLER
```

```
const ultraSoundMoney = await ethers.getContractAt("0x....");
await ultraSoundMoney.setTotalSupply();
```

```
console.log(
  await ethers.provider.send("debug_traceTransaction", [
    "0xa5d745ae8c5373317a8624bc2f1ee31c50f98f2b0d77095069ad728ccdc27054",
  ])
);
```

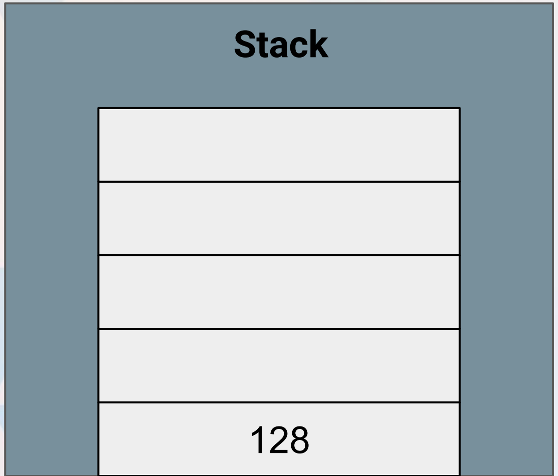
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



Let's trace the opcodes of a real transaction.

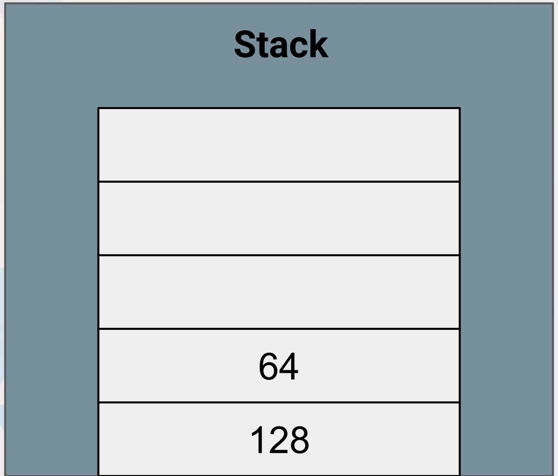
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

PUSH1 0x80: Push `128` onto the stack



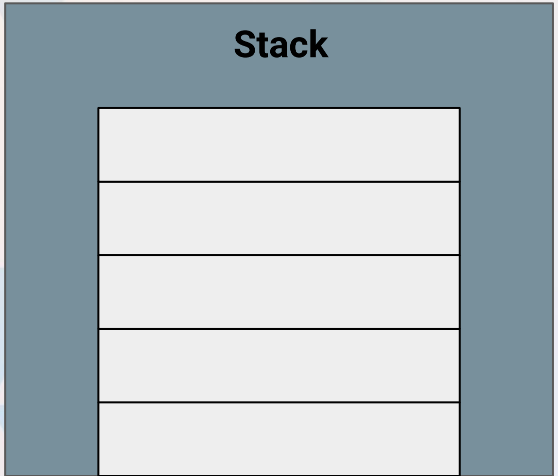
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

PUSH1 0x80: Push `128` onto the stack
PUSH1 0x40: Push `64` onto the stack

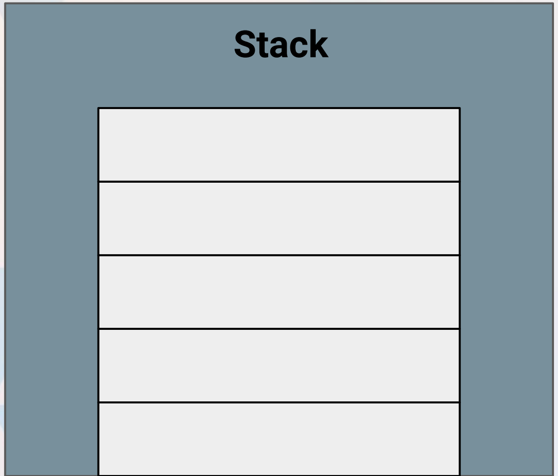


```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

PUSH1 0x80: Push `128` onto the stack
PUSH1 0x40: Push `64` onto the stack
MSTORE: Store `128` at an offset of `64` in memory



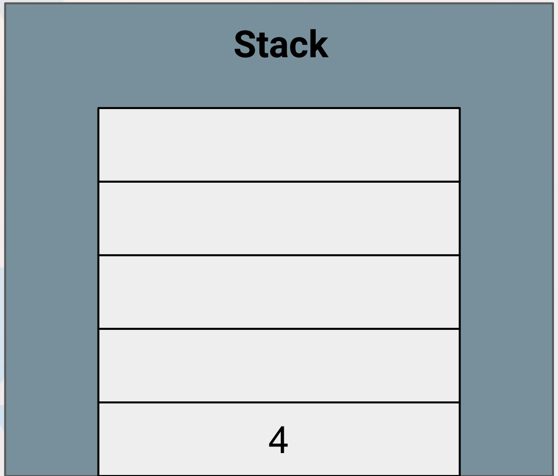
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



PUSH1 0x80: Push `128` onto the stack
PUSH1 0x40: Push `64` onto the stack
MSTORE: Store `128` at an offset of `64` in memory

Solidity uses the memory area between address zero and address `0x7F` for internal purposes, and stores data starting at address `0x80`

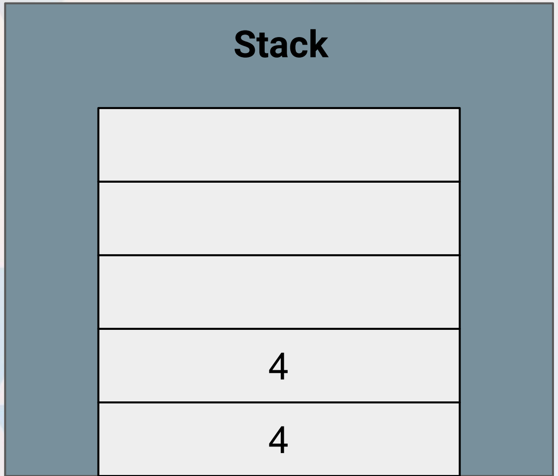
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



PUSH1 0x4: Push `4` onto the stack.

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

PUSH1 0x4: Push `4` onto the stack.
CALLDATASIZE: Push size of input data onto stack.



1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE

25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI

32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI

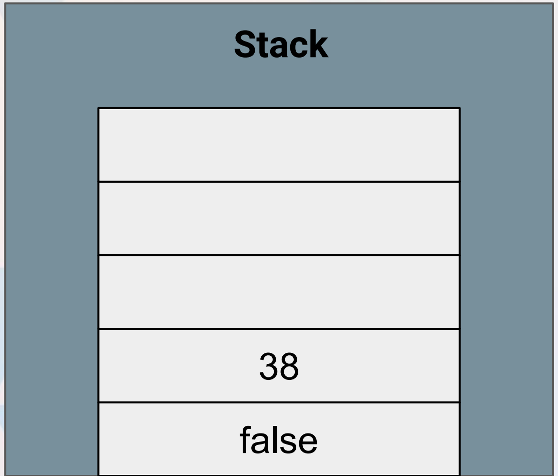
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP

PUSH1 0x4: Push `4` onto the stack.
CALLDATASIZE: Push size of input data onto stack.
LT: Check if input data is less than 4.

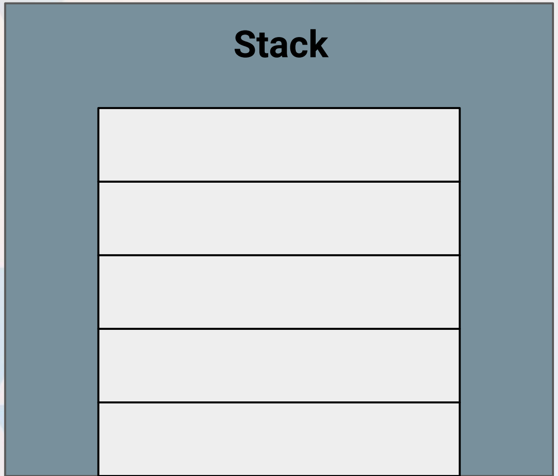
Stack
false

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

PUSH1 0x4: Push `4` onto the stack.
CALLDATASIZE: Push size of input data onto stack.
LT: Check if input data is less than 4.
PUSH1 0x28: Push `38` onto the stack.

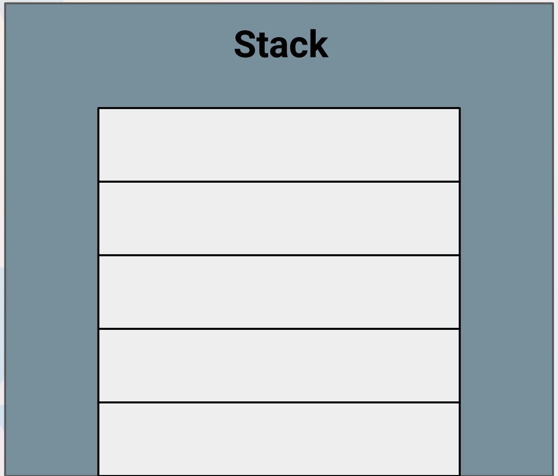


```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



PUSH1 0x4: Push `4` onto the stack.
CALLDATASIZE: Push size of input data onto stack.
LT: Check if input data is less than 4.
PUSH1 0x28: Push `38` onto the stack.
JUMPI: Jump to `38` (revert) if call data size is less than 4.

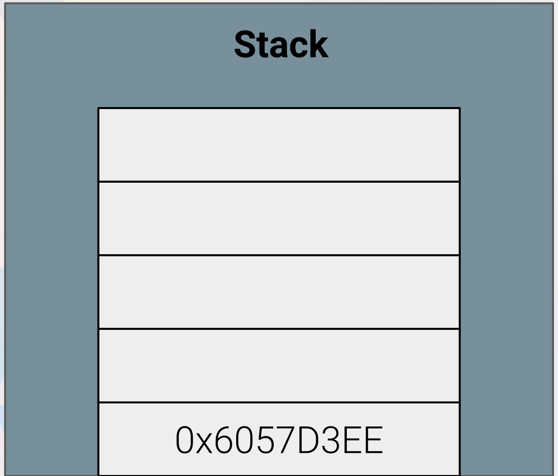
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



PUSH1 0x4: Push `4` onto the stack.
CALLDATASIZE: Push size of input data onto stack.
LT: Check if input data is less than 4.
PUSH1 0x28: Push `38` onto the stack.
JUMPI: Jump to `38` (revert) if call data size is less than 4.

Since function signatures are 4 bytes in length - if the CALLDATASIZE is less than 4 bytes it is impossible to determine which function is intended to be called.

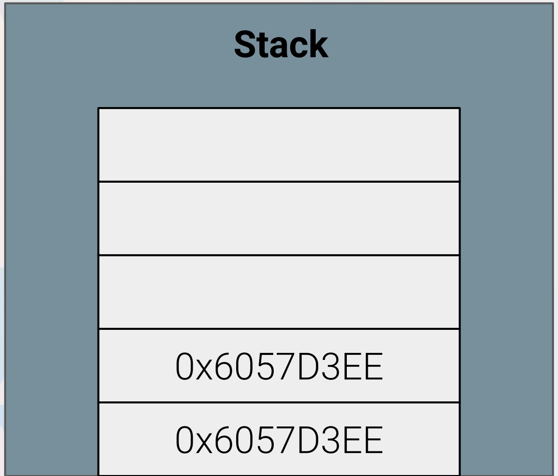
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



CALLDATALOAD: Push the calldata onto the stack,

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

CALLDATALOAD: Push the calldata onto the stack,
PUSH4 0x6057D3EE: push 0x6057D3EE onto the stack.

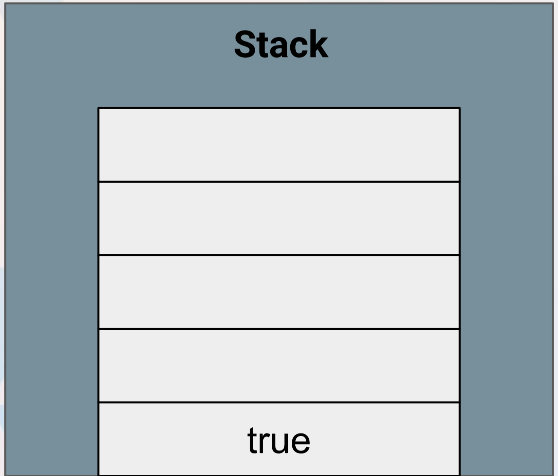


1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE

25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI

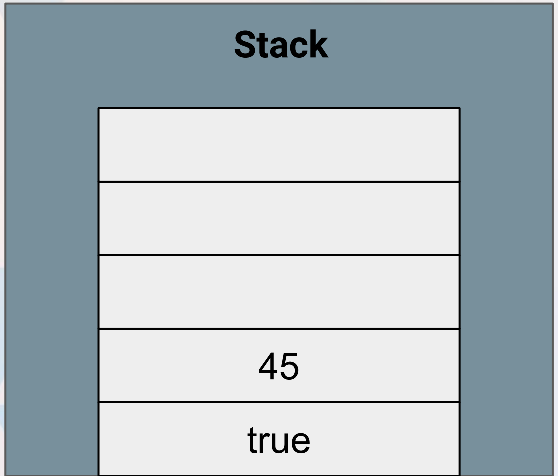
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI

45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP



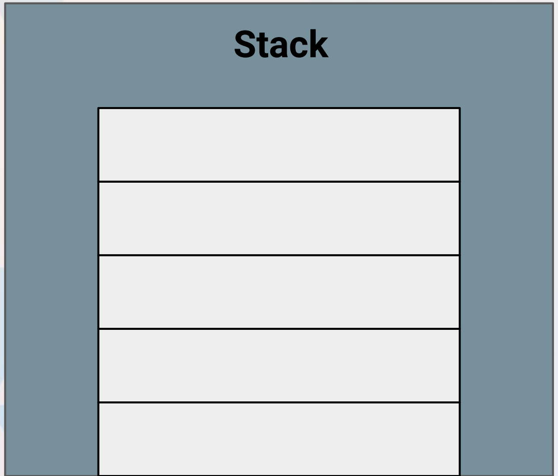
CALLDATALOAD: Push the calldata onto the stack,
PUSH4 0x6057D3EE: push 0x6057D3EE onto the stack.
EQ: Check if the calldata is equal to 0x6057D3EE

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



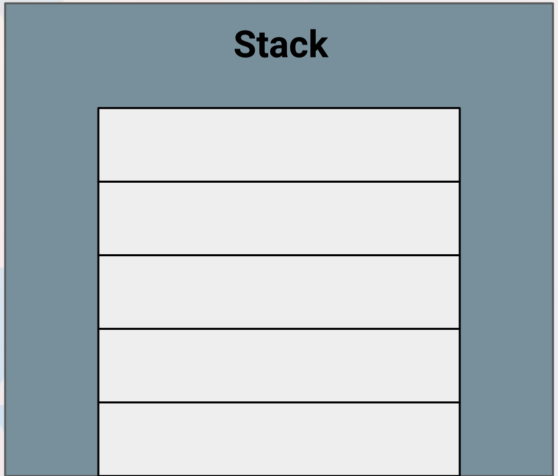
CALLDATALOAD: Push the calldata onto the stack,
PUSH4 0x6057D3EE: push 0x6057D3EE onto the stack.
EQ: Check if the calldata is equal to 0x6057D3EE
PUSH1 0x2D: Push `45` onto the stack.

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



CALLDATALOAD: Push the calldata onto the stack,
PUSH4 0x6057D3EE: push 0x6057D3EE onto the stack.
EQ: Check if the calldata is equal to 0x6057D3EE
PUSH1 0x2D: Push `45` onto the stack.
JUMPI: Jump to `45` (setTotalSupply) if calldata is equal to 0x6057D3EE

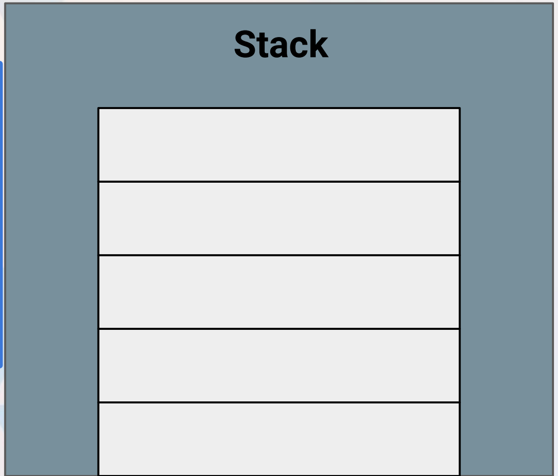
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



CALLDATALOAD: Push the calldata onto the stack,
PUSH4 0x6057D3EE: push 0x6057D3EE onto the stack.
EQ: Check if the calldata is equal to 0x6057D3EE
PUSH1 0x2D: Push `45` onto the stack.
JUMPI: Jump to `45` (setTotalSupply) if calldata is equal to 0x6057D3EE

This is how the EVM determines which function to call.

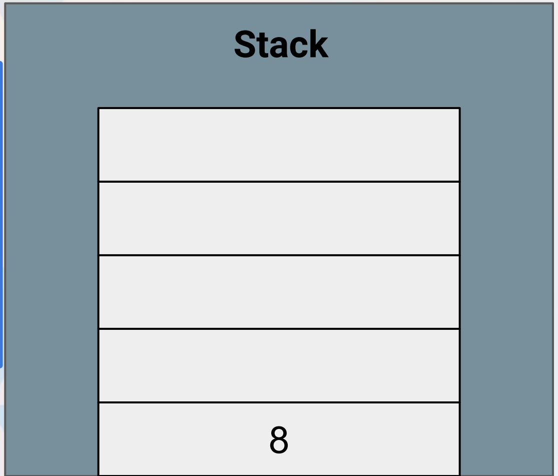
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



JUMPDEST: Marks a valid jump destination.

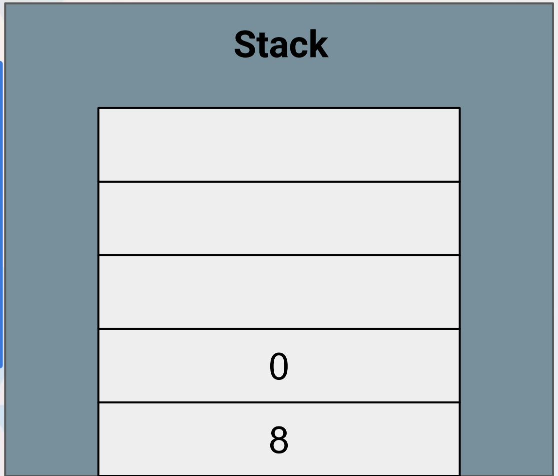
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

JUMPDEST: Marks a valid jump destination.
PUSH1 0x8: push 8 onto the stack.



```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

JUMPDEST: Marks a valid jump destination.
PUSH1 0x8: push 8 onto the stack.
PUSH1 0x0: push 0 onto the stack.

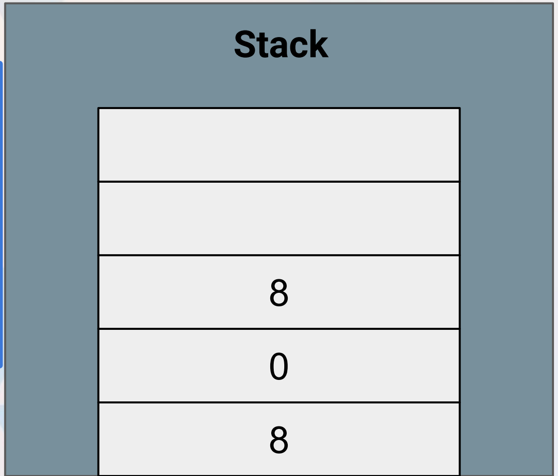


1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE

25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI

32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI

45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP



JUMPDEST: Marks a valid jump destination.
PUSH1 0x8: push 8 onto the stack.
PUSH1 0x0: push 0 onto the stack.
DUP2: Duplicate the 2nd-from-the-top word of stack.

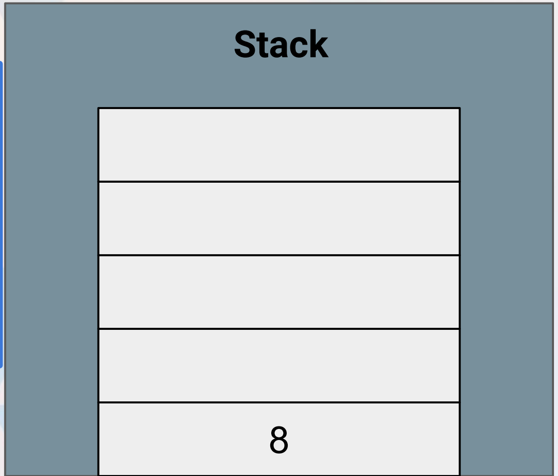
```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```

Stack

0
8
8

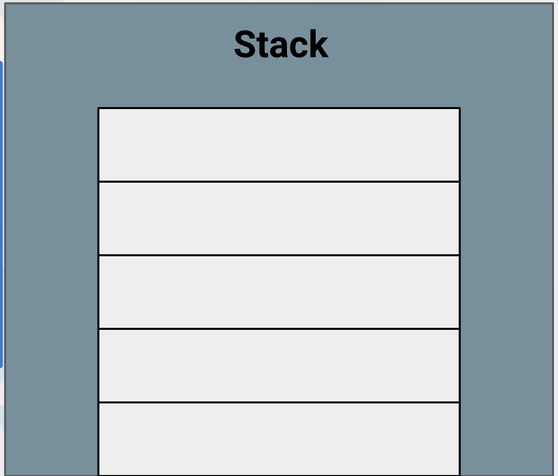
JUMPDEST: Marks a valid jump destination.
PUSH1 0x8: push 8 onto the stack.
PUSH1 0x0: push 0 onto the stack.
DUP2: Duplicate the 2nd-from-the-top word of stack.
SWAP1: Swap 1st and 2nd words on the stack.

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



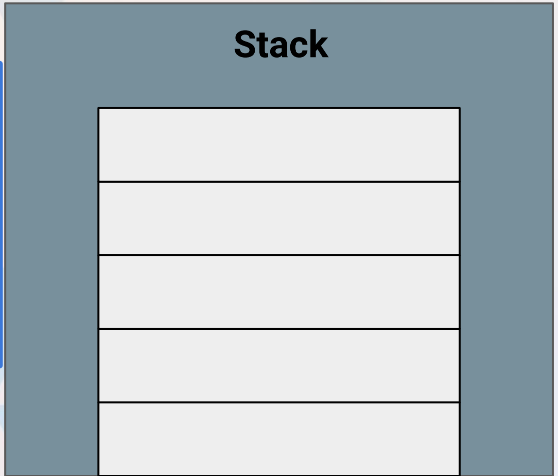
JUMPDEST: Marks a valid jump destination.
PUSH1 0x8: push 8 onto the stack.
PUSH1 0x0: push 0 onto the stack.
DUP2: Duplicate the 2nd-from-the-top word of stack.
SWAP1: Swap 1st and 2nd words on the stack.
SSTORE: Save `8` to storage.

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



JUMPDEST: Marks a valid jump destination.
PUSH1 0x8: push 8 onto the stack.
PUSH1 0x0: push 0 onto the stack.
DUP2: Duplicate the 2nd-from-the-top word of stack.
SWAP1: Swap 1st and 2nd words on the stack.
SSTORE: Save `8` to storage.
POP: Remove the word on top of the stack.

```
1: PUSH1 0x80
2: PUSH1 0x40
3: MSTORE
------
25: PUSH1 0x04
26: CALLDATASIZE
27: LT
28: PUSH1 0x28
29: JUMPI
------
32: CALLDATALOAD
33: PUSH4 0x6057D3EE
34: EQ
35: PUSH1 0x2D
36: JUMPI
------
45: JUMPDEST
46: PUSH1 0x08
47: PUSH1 0x00
48: DUP2
49: SWAP1
50: SSTORE
51: POP
```



JUMPDEST: Marks a valid jump destination.
PUSH1 0x8: push 8 onto the stack.
PUSH1 0x0: push 0 onto the stack.
DUP2: Duplicate the 2nd-from-the-top word of stack.
SWAP1: Swap 1st and 2nd words on the stack.
SSTORE: Save `8` to storage.
POP: Remove the word on top of the stack.

Why are we duplicating and swapping here?

Section 3

Gas Optimization Using Yul

```
assembly {
    // Get a location of some free memory and store it in result as
    // Solidity does for memory variables.
    bs := mload(0x40)
    // Put 0x20 at the first word, the length of bytes for uint256 value
    mstore(bs, 0x20)
    //In the next word, put value in bytes format to the next 32 bytes
    mstore(add(bs, 0x20), _value)
    // Update the free-memory pointer by padding our last write location to 32 bytes
    mstore(0x40, add(bs, 0x40))
}
```

```solidity
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.9;


contract UltraSoundMoney {
    uint totalSupply;

    function setTotalSupply() public {
        totalSupply = 8;
    }

    function optimizedSetTotalSupply() public {
        assembly {
            sstore(0x00, 0x08)
        }
    }
}
```

```
solc contracts/UltraSoundMoney.sol --opcodes
```

setTotalSupply

```
JUMPDEST
PUSH1 0x8
PUSH1 0x0
DUP2
SWAP1
SSTORE
POP
```

optimizedSetTotalSupply

```
JUMPDEST
PUSH1 0x8
PUSH1 0x0
SSTORE
```

Solc version: 0.8.17		Optimizer enabled: false		Runs: 200	Block limit: 30000000 gas	
Methods						
Contract	**Method**	Min	Max	Avg	**# calls**	**usd (avg)**
UltraSoundMoney	setTotalSupply	23400	43300	23599	100	-

Solc version: 0.8.17	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas

Methods

Contract	Method	Min	Max	Avg	# calls	usd (avg)
UltraSoundMoney	setTotalSupply	23400	43300	23599	100	-

Solc version: 0.8.17	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas

Methods

Contract	Method	Min	Max	Avg	# calls	usd (avg)
UltraSoundMoney	optimizedSetTotalSupply	23392	43292	23591	100	-

Solc version: 0.8.17	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas			
Methods						
Contract	**Method**	Min	Max	Avg	**# calls**	**usd (avg)**
UltraSoundMoney	setTotalSupply	23400	43300	23599	100	-

Solc version: 0.8.17	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas			
Methods						
Contract	**Method**	Min	Max	Avg	**# calls**	**usd (avg)**
UltraSoundMoney	optimizedSetTotalSupply	23392	43292	23591	100	-

```
const config: HardhatUserConfig = {
  solidity: {
    version: "0.8.17",
    settings: {
      optimizer: {
        enabled: true, // <--
        runs: 200,
      },
    },
  },
  gasReporter: {
    enabled: true,
  },
};
```

Solc version: 0.8.17		Optimizer enabled: true		Runs: 200	Block limit: 30000000 gas	
Methods						
Contract	**Method**	Min	Max	Avg	**# calls**	**usd (avg)**
UltraSoundMoney	setTotalSupply	23380	43280	23579	100	-

Solc version: 0.8.17		Optimizer enabled: true		Runs: 200	Block limit: 30000000 gas	
Methods						
Contract	**Method**	Min	Max	Avg	**# calls**	**usd (avg)**
UltraSoundMoney	optimizedSetTotalSupply	23380	43280	23579	100	-

Optimizing contracts is hard - chances are you are not going to do a better job than the compiler unless you really know what you're doing.

Contracts containing assembly are generally harder to reason about and harder to audit than contracts written in Solidity or Vyper.

If you're writing your own assembly code - always measure and make sure that your implementation is better than the compilers.

Remember that a lot of the memory management stuff Solidity does under the hood is there for safety reasons - and just because an opcode looks like its unnecessary does not mean that it actually is.

Thank you!

Alex Bazhenov
Lead Developer, Tally Ho
alex@tally.cash
@0xDaedalus

Appendix

Gilbert Garza (@soundly_typed)

https://leftasexercise.com/2021/09/05/a-deep-dive-into-solidity-contract-creation-and-the-init-code/

https://ethereum.org/en/developers/docs/evm/

https://jeancvllr.medium.com/solidity-tutorial-all-about-assembly-5acdfefde05c

https://github.com/crytic/evm-opcodes

https://hackmd.io/@gn56kcRBQc6mOi7LCgbv1g/rJez8O8st