Ethereum and EVM

●●●

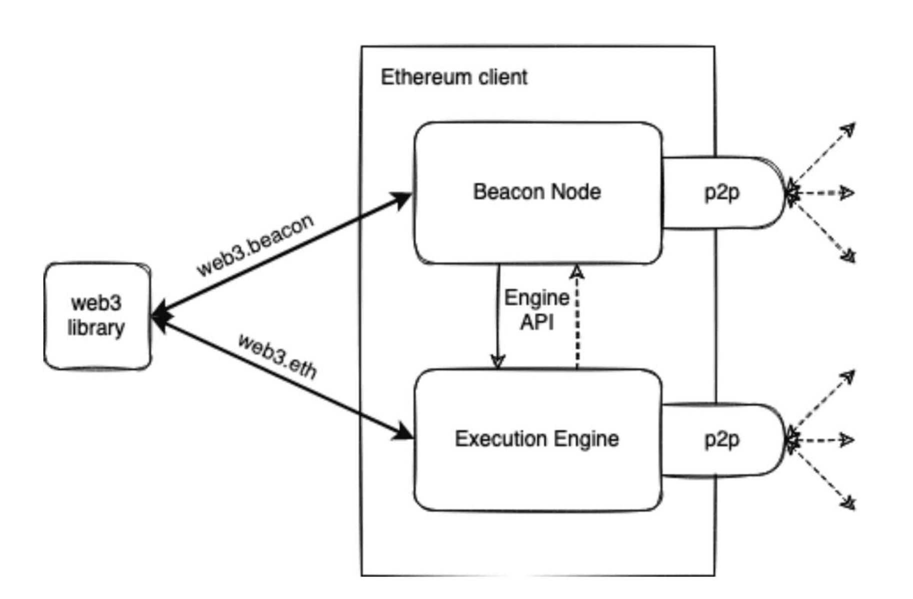
nguyenvietdungcs52@gmail.com

Agenda

1. Introduction to Ethereum and EVM based blockchains
2. EVM and Solidity
3. EVM and Stack-based virtual machine
4. EVM in Go-Ethereum
5. Q&A

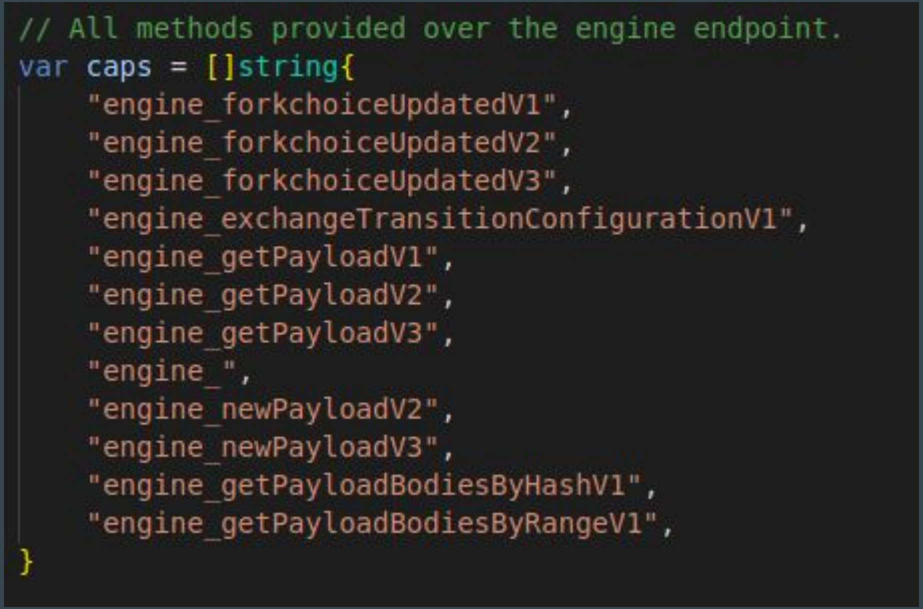
1. Ethereum and EVM based blockchains

- Blockchain network: P2P networks
- Execute smart contracts by using Ethereum Virtual Machine
- Ethereum client
  - Consensus client
  - Execution client



1. Ethereum and EVM based blockchains

- Consensus client:
- Execution client: Go-Ethereum
  - eth/catalyst/api.go



```go
// All methods provided over the engine endpoint.
var caps = []string{
	"engine_forkchoiceUpdatedV1",
	"engine_forkchoiceUpdatedV2",
	"engine_forkchoiceUpdatedV3",
	"engine_exchangeTransitionConfigurationV1",
	"engine_getPayloadV1",
	"engine_getPayloadV2",
	"engine_getPayloadV3",
	"engine_",
	"engine_newPayloadV2",
	"engine_newPayloadV3",
	"engine_getPayloadBodiesByHashV1",
	"engine_getPayloadBodiesByRangeV1",
}
```

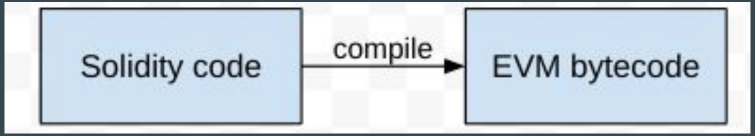
1. Ethereum and EVM based blockchains

- EVM chains: use Ethereum Virtual Machine for the execution of Smart Contracts
- Smart Contracts are programs run on EVM
- Programming languages:
  - Solidity
  - Vyper

Source link

2. EVM and Solidity

- EVM is a virtual machine. EVM can only “understand" EVM bytecodes
  - EVM bytecode is not convenient for programming
- Solidity is a high level programming language
  - Solidity code need to be compiled into bytecode

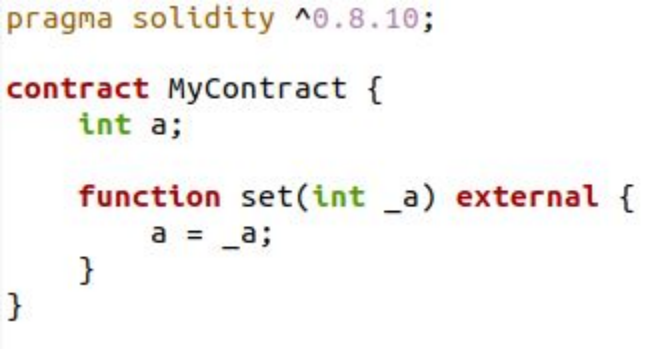


2. EVM and Solidity

- EVM vs Solidity
  - EVM is virtual machine
  - Solidity is used for writing code that EVM can understand
- Compile process: Solidity -> EVM bytecode
- Deploy contracts: store bytecode on blockchain
- Execute transactions(sender, to, value, input):
  - Load bytecode by contract address
  - Execute with input, value and context (block, sender...)
- How those steps actual be implemented?

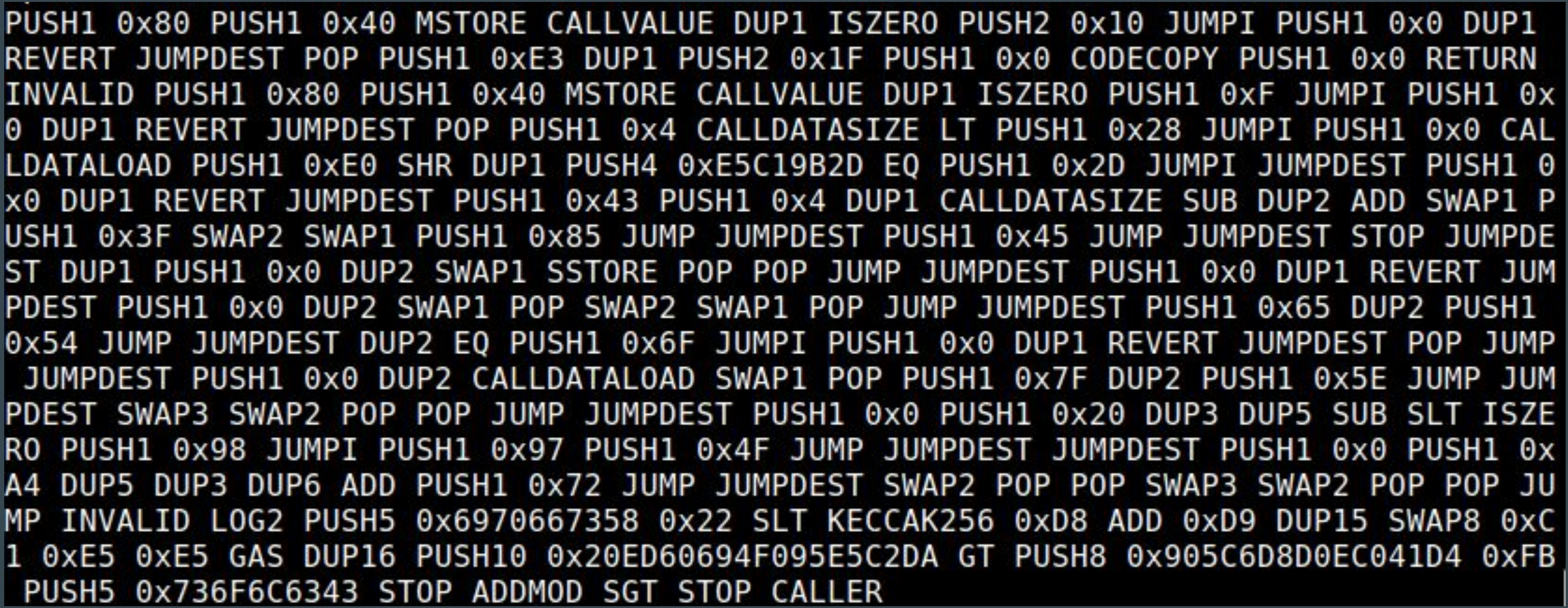
2. EVM and Solidity

- An example
  - Compile: solc –bin MyContract.sol
  - View in assembly: solc –opcode MyContract.sol



```
pragma solidity ^0.8.10;

contract MyContract {
    int a;

    function set(int _a) external {
        a = _a;
    }
}
```

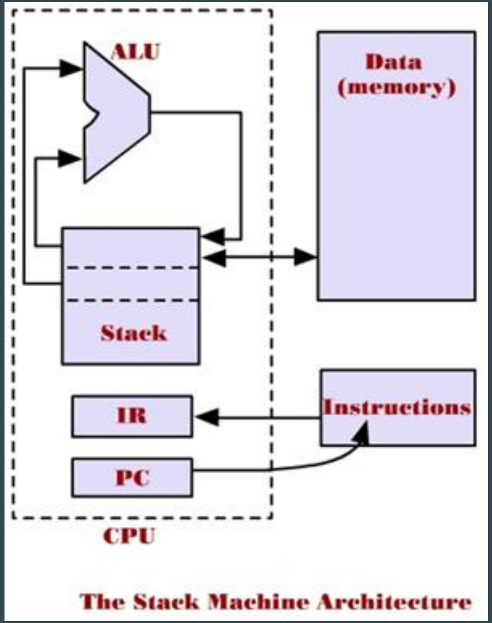
2. EVM and Solidity

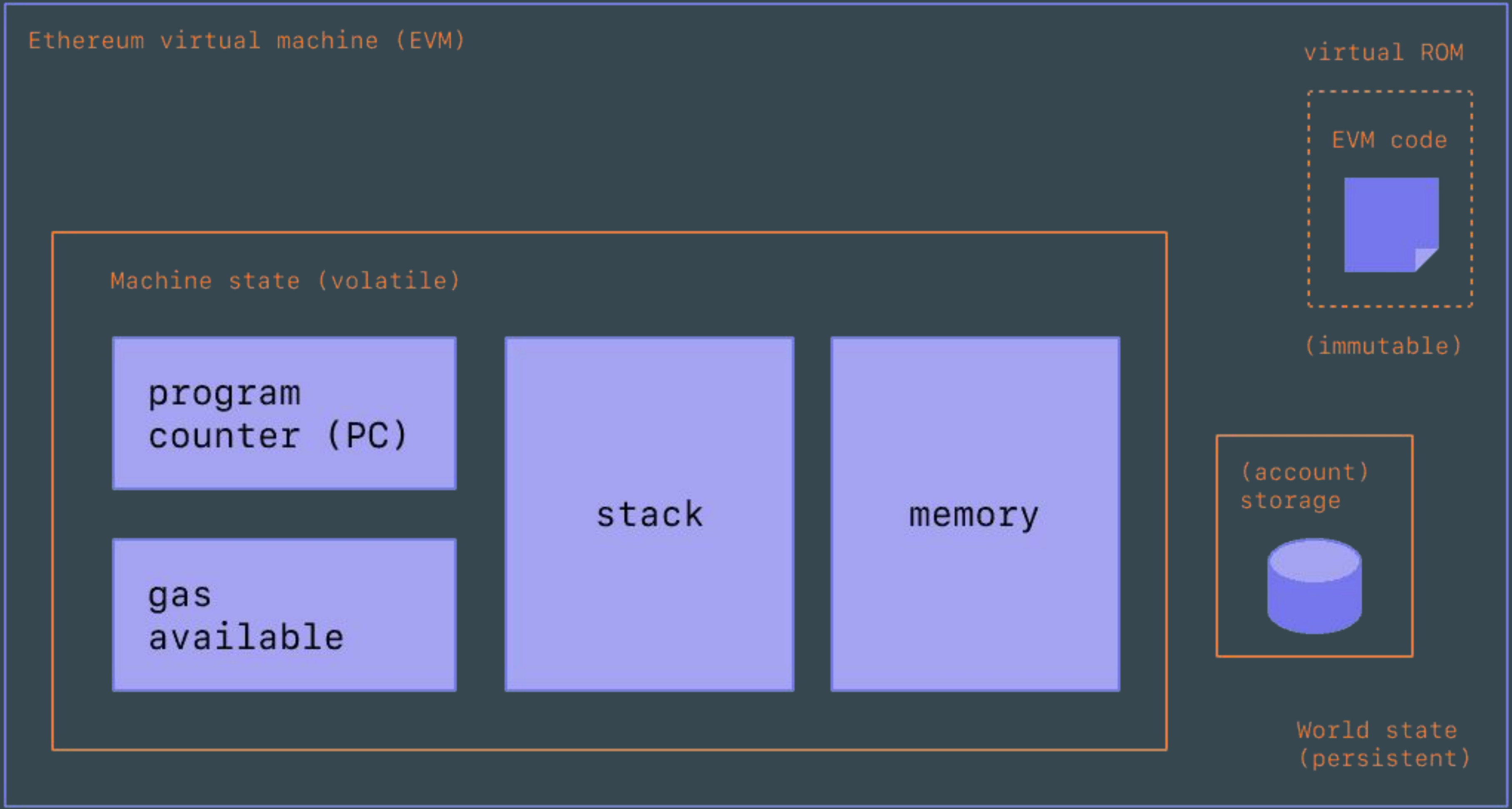


```
PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH2 0x10 JUMPI PUSH1 0x0 DUP1
REVERT JUMPDEST POP PUSH1 0xE3 DUP1 PUSH2 0x1F PUSH1 0x0 CODECOPY PUSH1 0x0 RETURN
INVALID PUSH1 0x80 PUSH1 0x40 MSTORE CALLVALUE DUP1 ISZERO PUSH1 0xF JUMPI PUSH1 0x
0 DUP1 REVERT JUMPDEST POP PUSH1 0x4 CALLDATASIZE LT PUSH1 0x28 JUMPI PUSH1 0x0 CAL
LDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0xE5C19B2D EQ PUSH1 0x2D JUMPI JUMPDEST PUSH1 0
x0 DUP1 REVERT JUMPDEST PUSH1 0x43 PUSH1 0x4 DUP1 CALLDATASIZE SUB DUP2 ADD SWAP1 P
USH1 0x3F SWAP2 SWAP1 PUSH1 0x85 JUMP JUMPDEST PUSH1 0x45 JUMP JUMPDEST STOP JUMPDE
ST DUP1 PUSH1 0x0 DUP2 SWAP1 SSTORE POP POP JUMP JUMPDEST PUSH1 0x0 DUP1 REVERT JUM
PDEST PUSH1 0x0 DUP2 SWAP1 POP SWAP2 SWAP1 POP JUMP JUMPDEST PUSH1 0x65 DUP2 PUSH1
0x54 JUMP JUMPDEST DUP2 EQ PUSH1 0x6F JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST POP JUMP
 JUMPDEST PUSH1 0x0 DUP2 CALLDATALOAD SWAP1 POP PUSH1 0x7F DUP2 PUSH1 0x5E JUMP JUM
PDEST SWAP3 SWAP2 POP POP JUMP JUMPDEST PUSH1 0x0 PUSH1 0x20 DUP3 DUP5 SUB SLT ISZE
RO PUSH1 0x98 JUMPI PUSH1 0x97 PUSH1 0x4F JUMP JUMPDEST JUMPDEST PUSH1 0x0 PUSH1 0x
A4 DUP5 DUP3 DUP6 ADD PUSH1 0x72 JUMP JUMPDEST SWAP2 POP POP SWAP3 SWAP2 POP POP JU
MP INVALID LOG2 PUSH5 0x6970667358 0x22 SLT KECCAK256 0xD8 ADD 0xD9 DUP15 SWAP8 0xC
1 0xE5 0xE5 GAS DUP16 PUSH10 0x20ED60694F095E5C2DA GT PUSH8 0x905C6D8D0EC041D4 0xFB
 PUSH5 0x736F6C6343 STOP ADDMOD SGT STOP CALLER
```

3. EVM and Stack-based virtual machine

- The architecture of EVM is a stack-based virtual machine
- Stack based architecture: EVM, WASM, JVM
- What is a stack-based virtual machine? (source)
  - CPU
    - ALU
    - Stack
    - IR (Instruction register)
    - PC
  - Memory
  - ROM



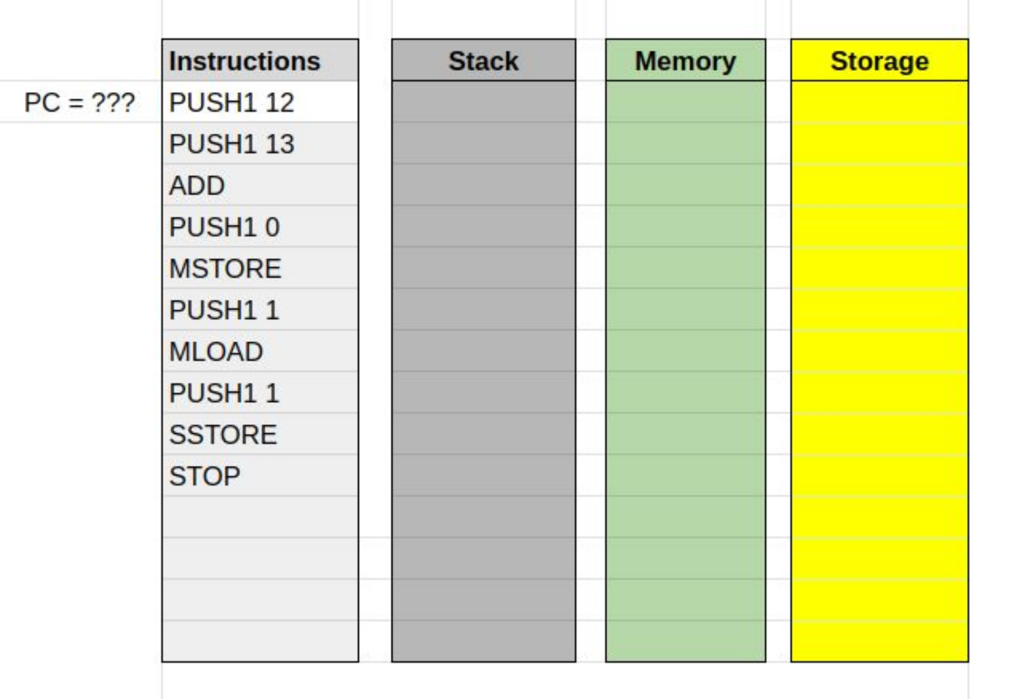

Ethereum virtual machine (EVM)
virtual ROM
EVM code
(immutable)
Machine state (volatile)
program counter (PC)
gas available
stack
memory
(account) storage
World state (persistent)

EVM opcodes

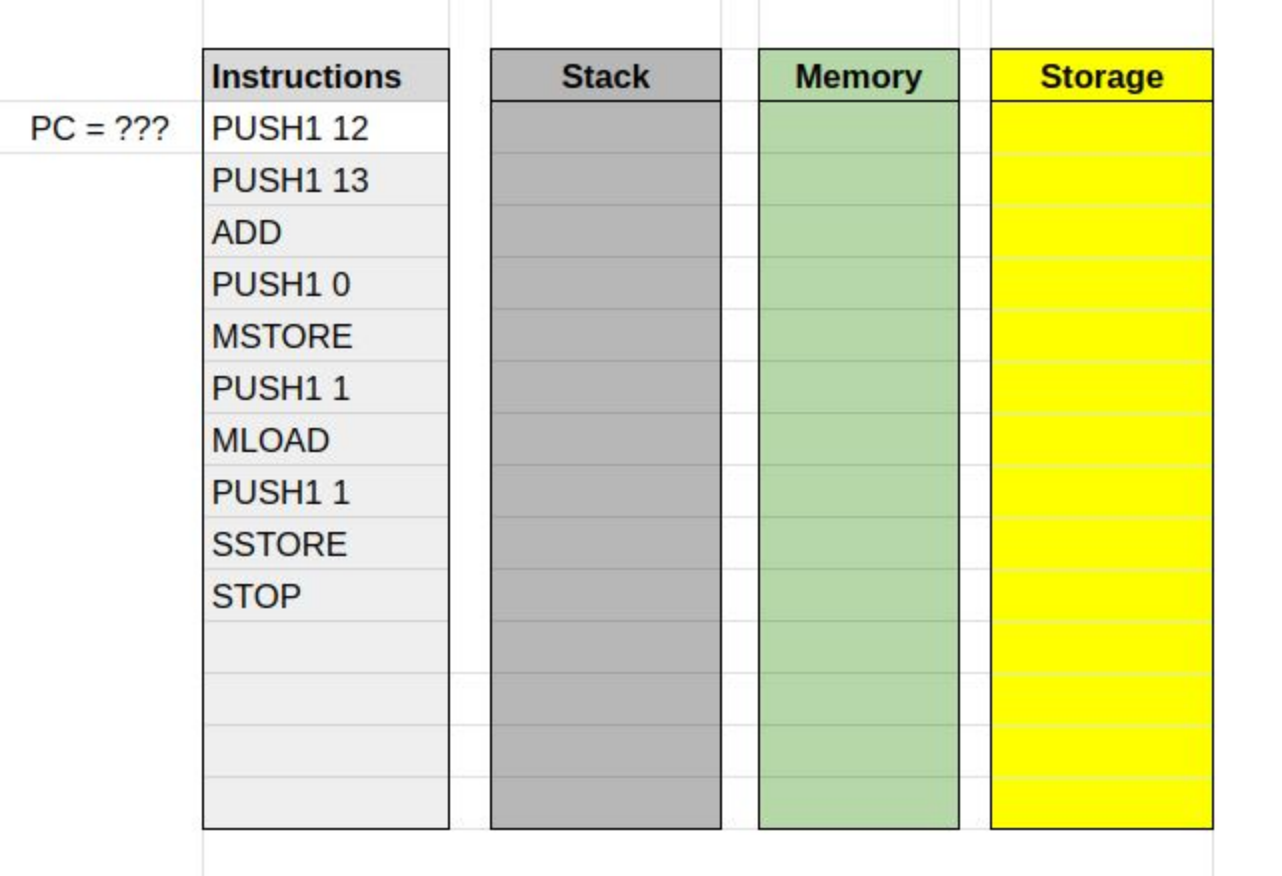
- PUSH1, PUSH2
- MLOAD, MSTORE, MSTORE8
- SSTORE, SLOAD
- ADD, SUB, SHR, AND, OR
- JUMP, JUMPI
- CALLDATASIZE, CALLDATACOPY
- CALL, DELEGATECALL
- RETURN
- REVERT

Example: execute instructions in EVM

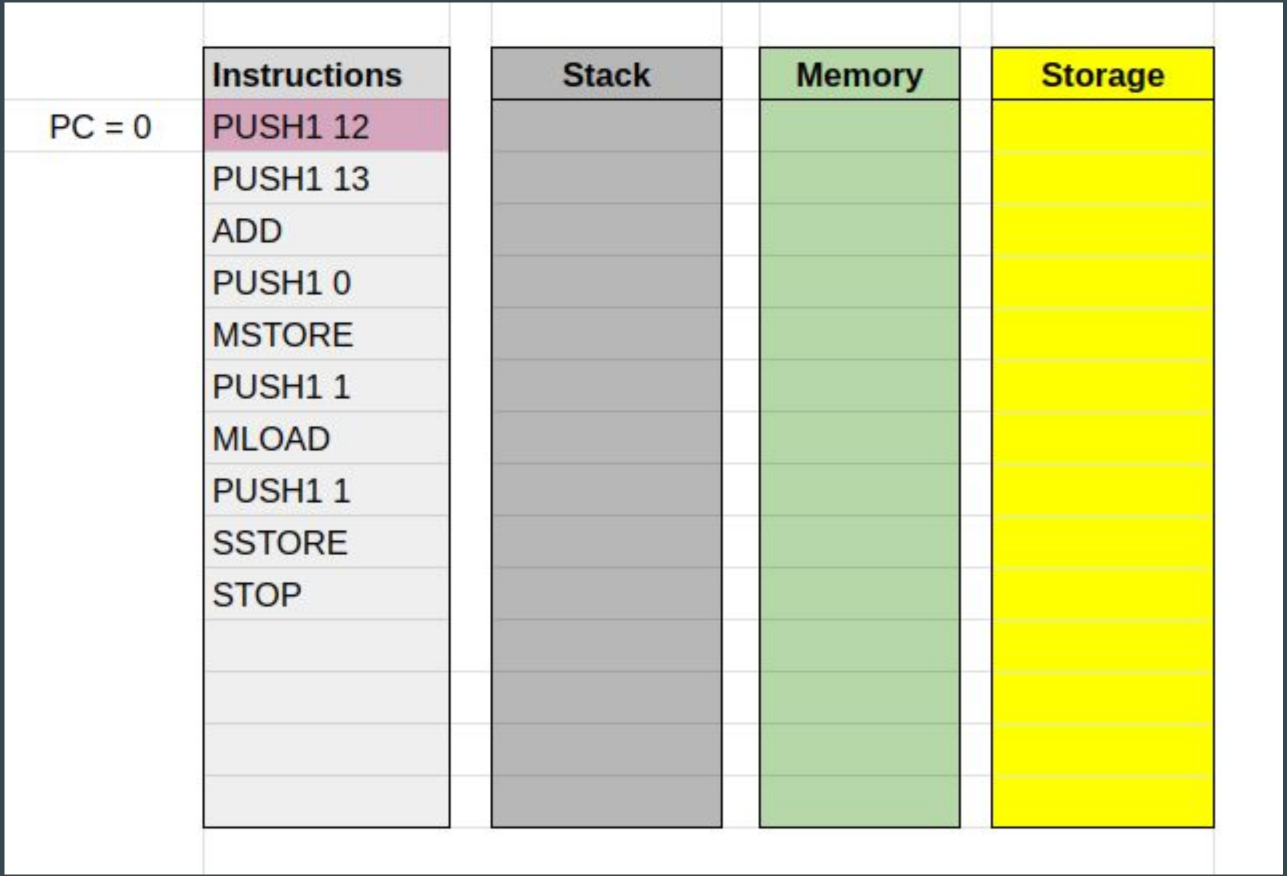
- Bytecode
- Stack
- Memory
- Storage
- PC



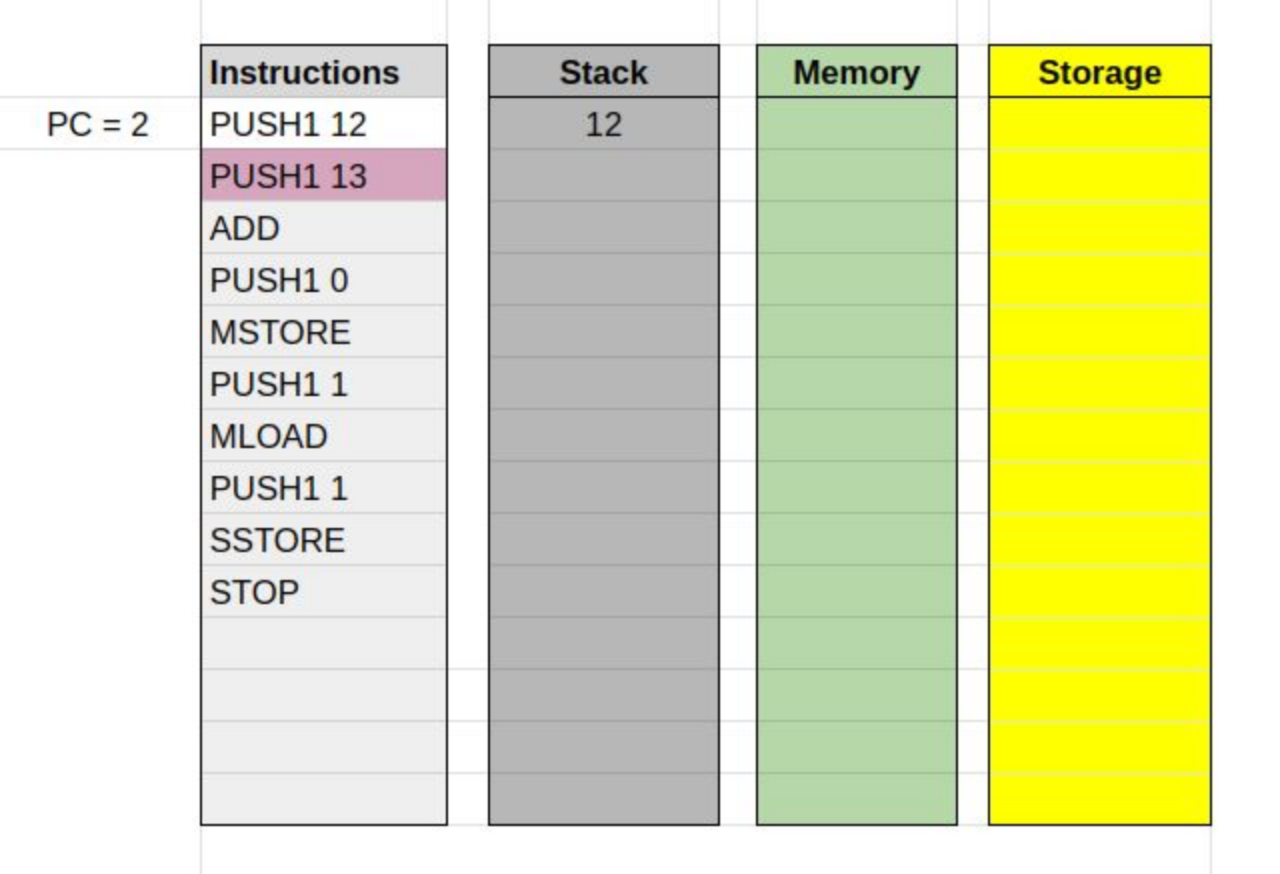
	Instructions	Stack	Memory	Storage
PC = ???	PUSH1 12			
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



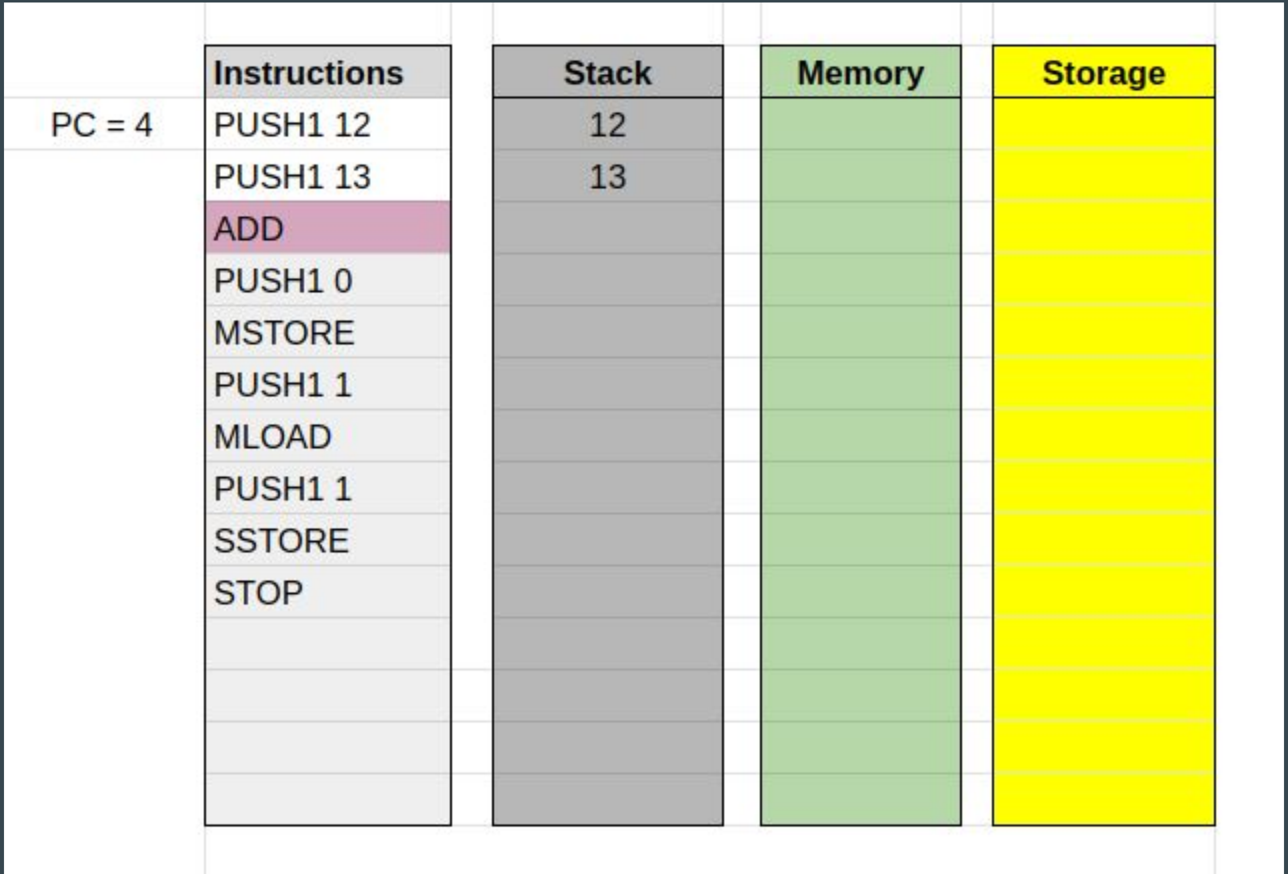
	Instructions	Stack	Memory	Storage
PC = ???	PUSH1 12			
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



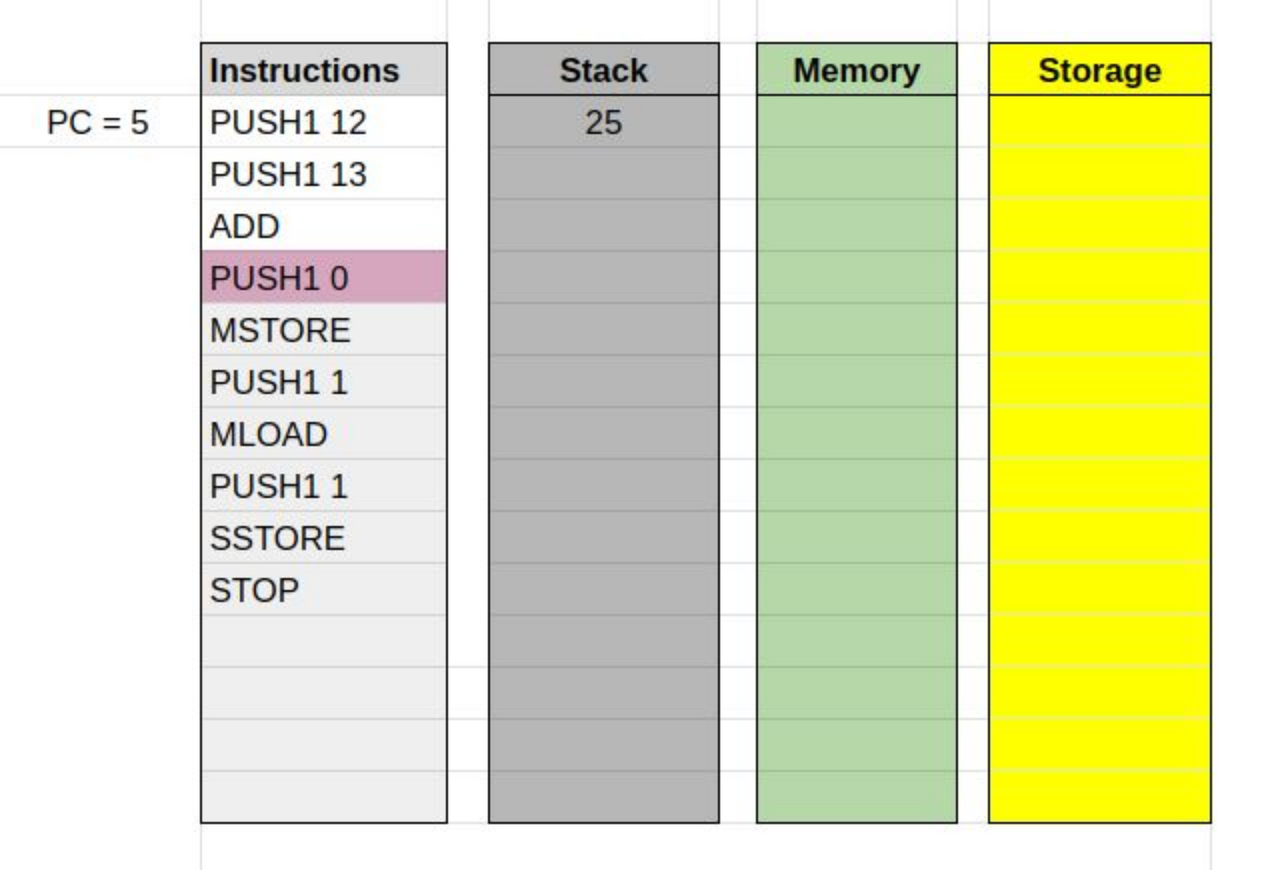
	Instructions	Stack	Memory	Storage
PC = 0	PUSH1 12			
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



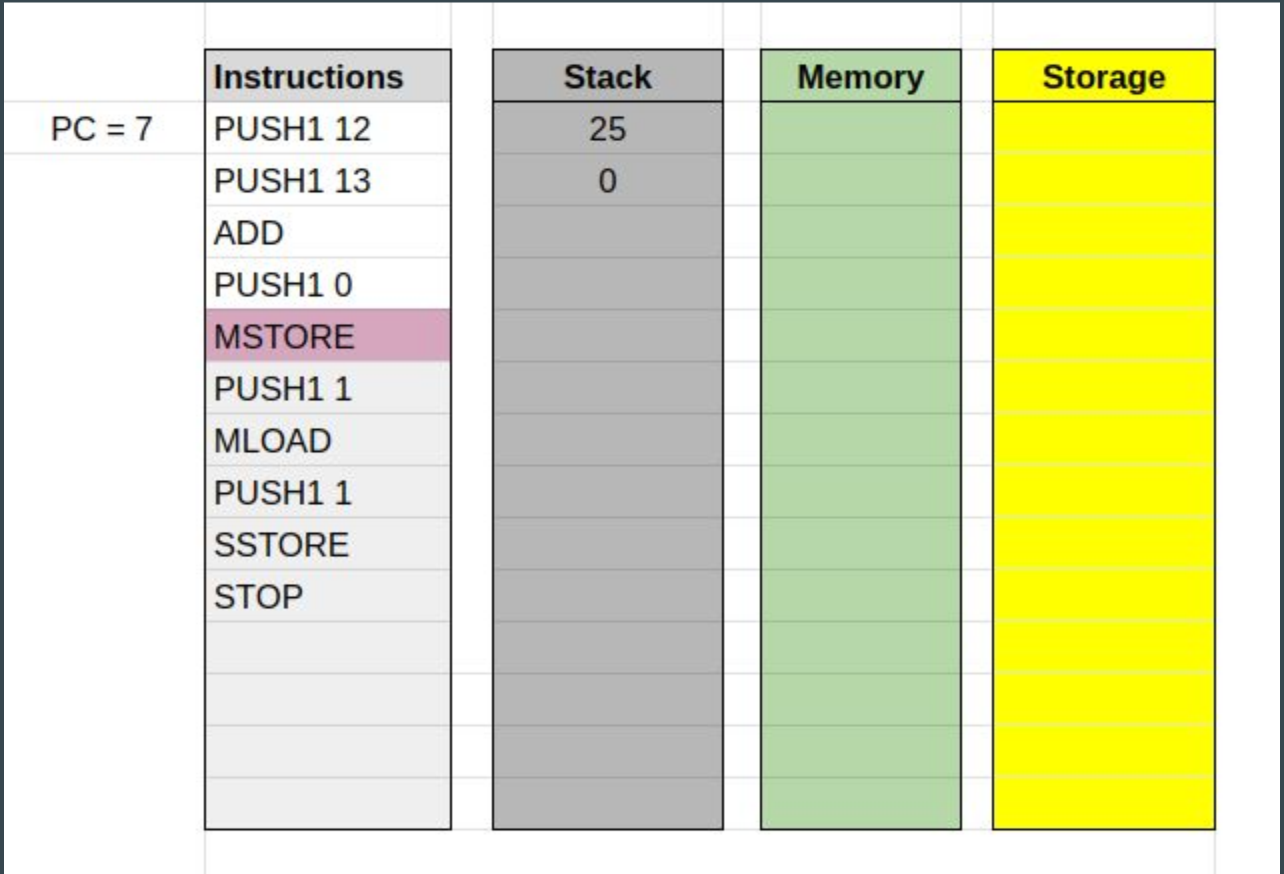
	Instructions	Stack	Memory	Storage
PC = 2	PUSH1 12	12		
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



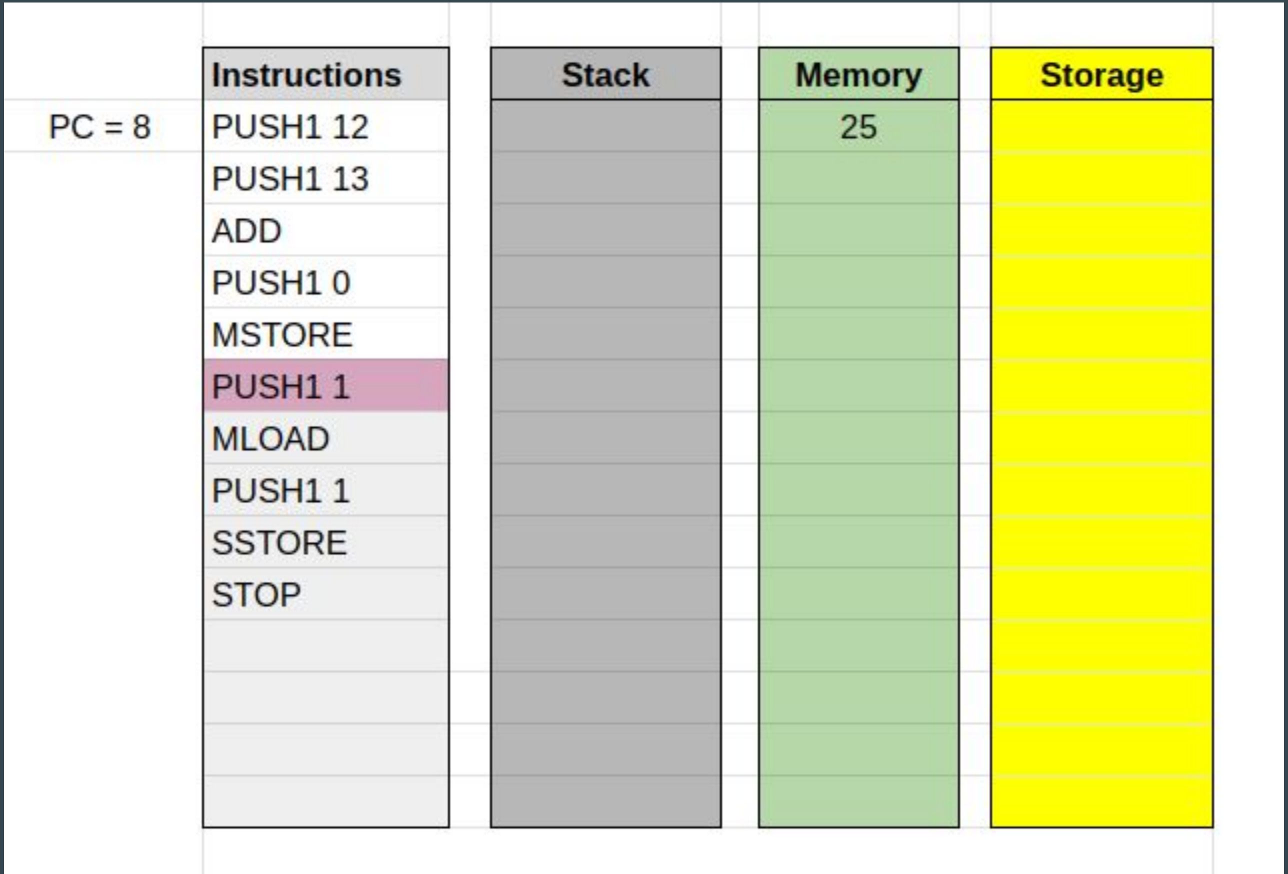
	Instructions	Stack	Memory	Storage
PC = 4	PUSH1 12	12		
	PUSH1 13	13		
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



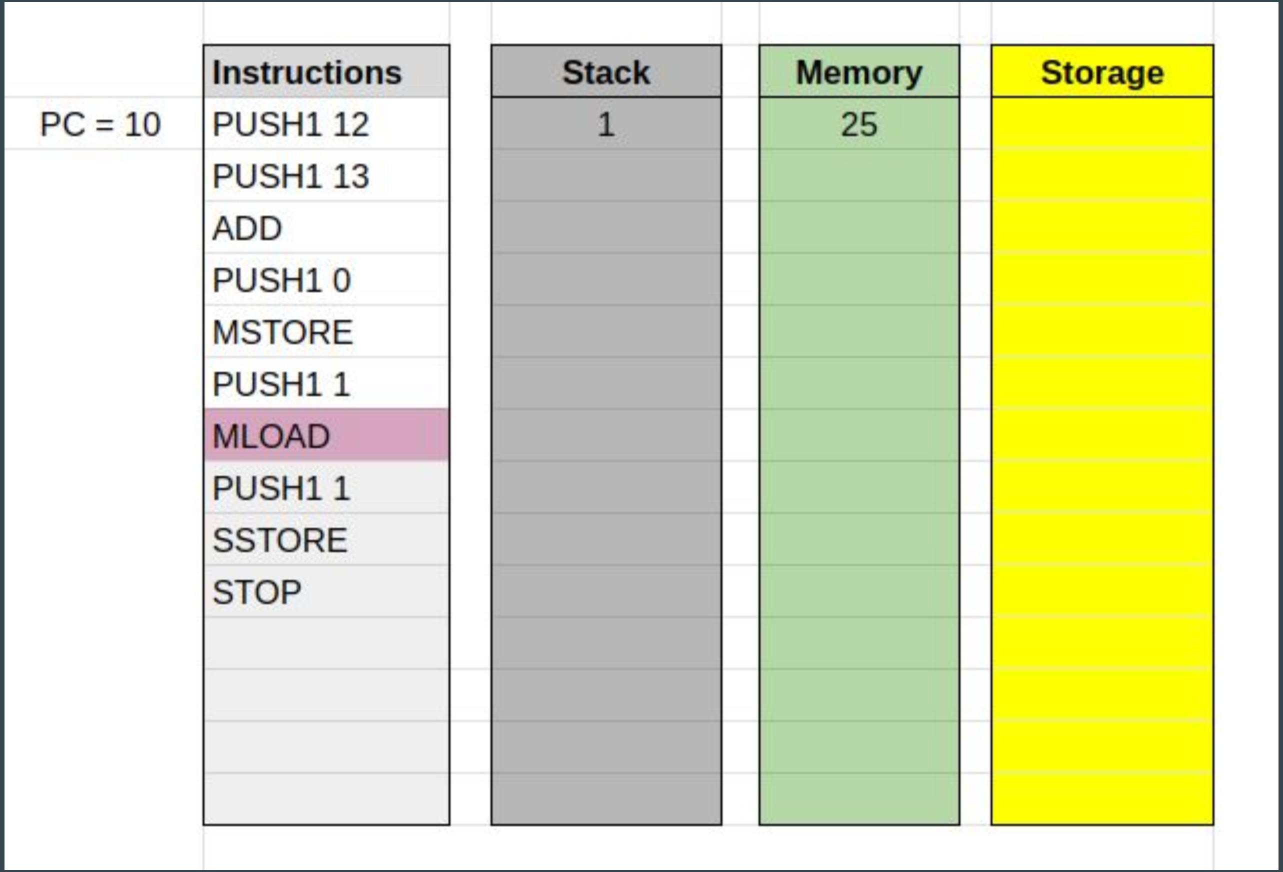
	Instructions	Stack	Memory	Storage
PC = 5	PUSH1 12	25		
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



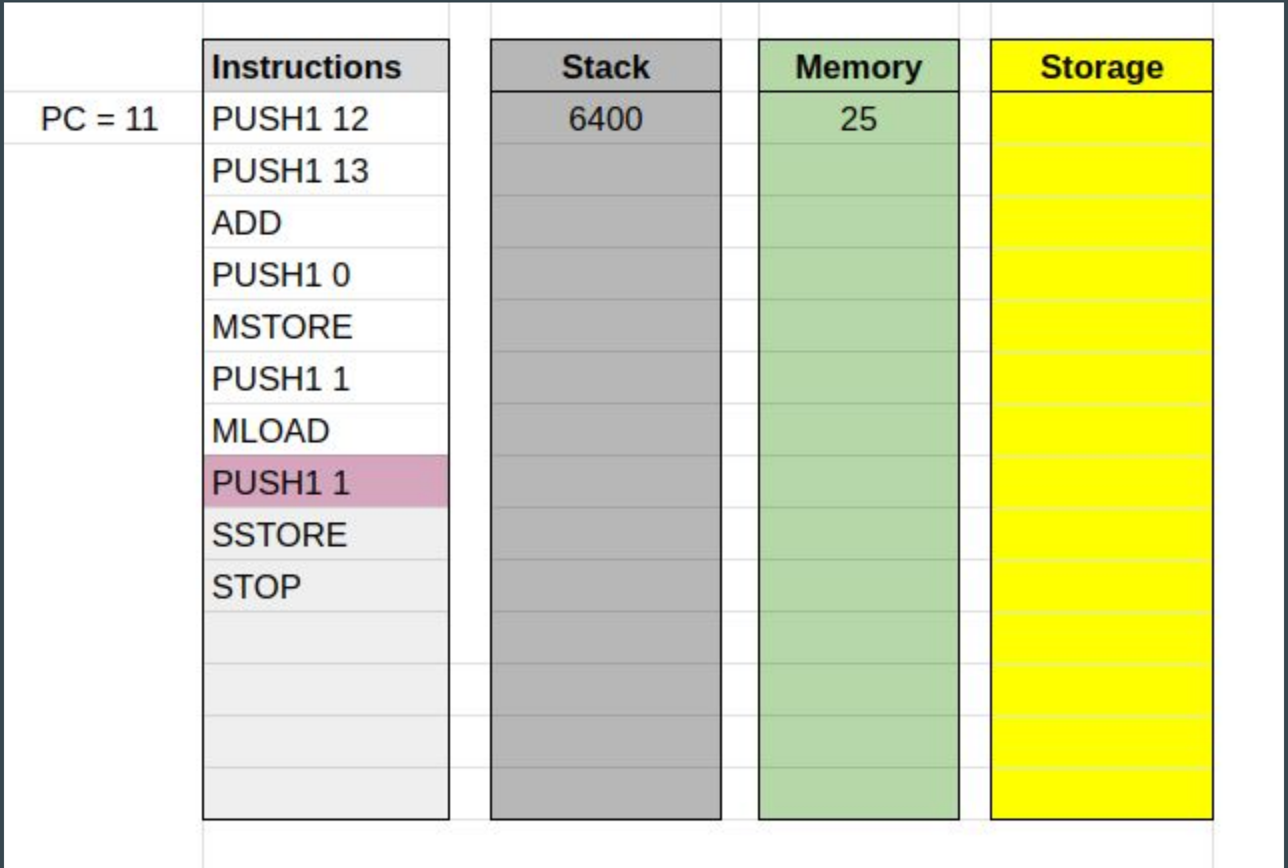
	Instructions	Stack	Memory	Storage
PC = 7	PUSH1 12	25		
	PUSH1 13	0		
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



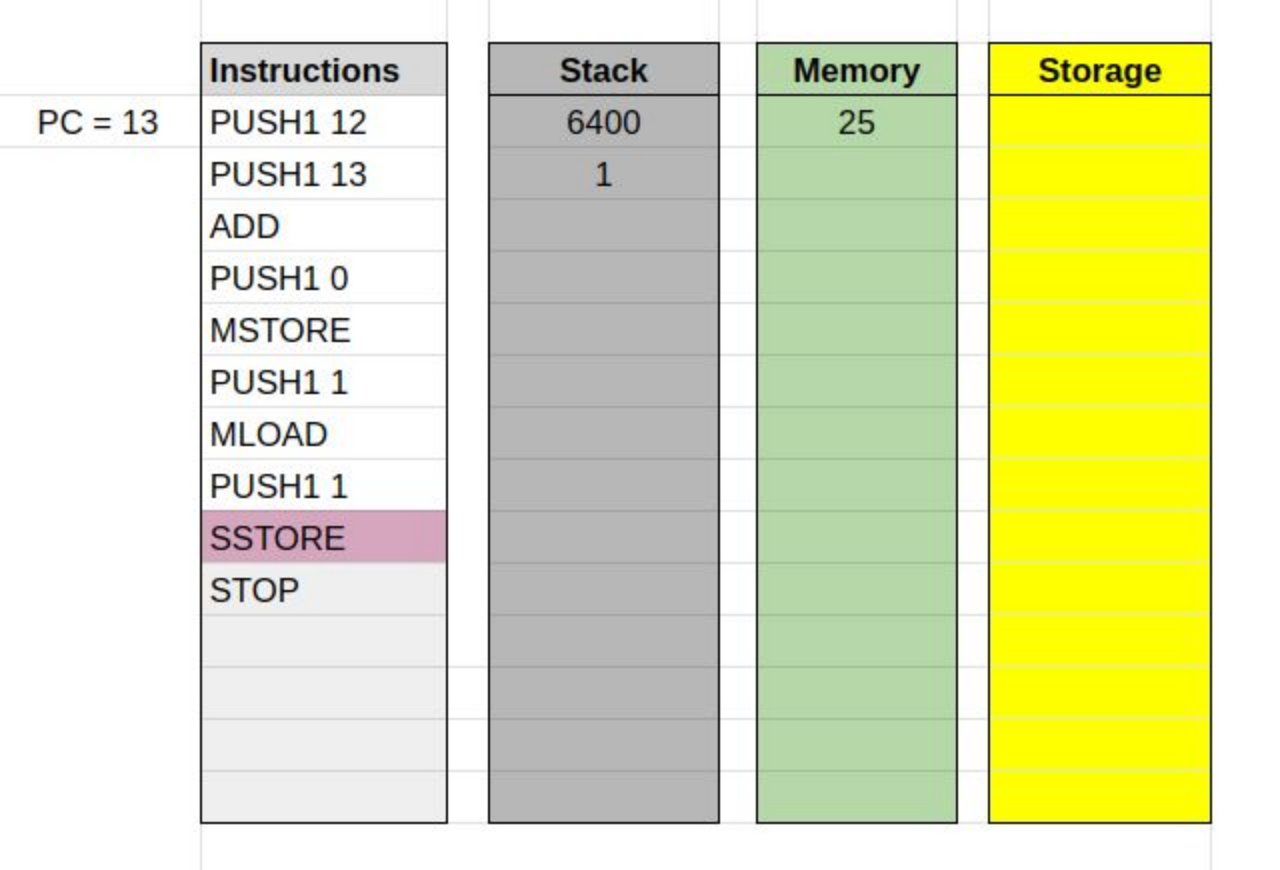
	Instructions	Stack	Memory	Storage
PC = 8	PUSH1 12		25	
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



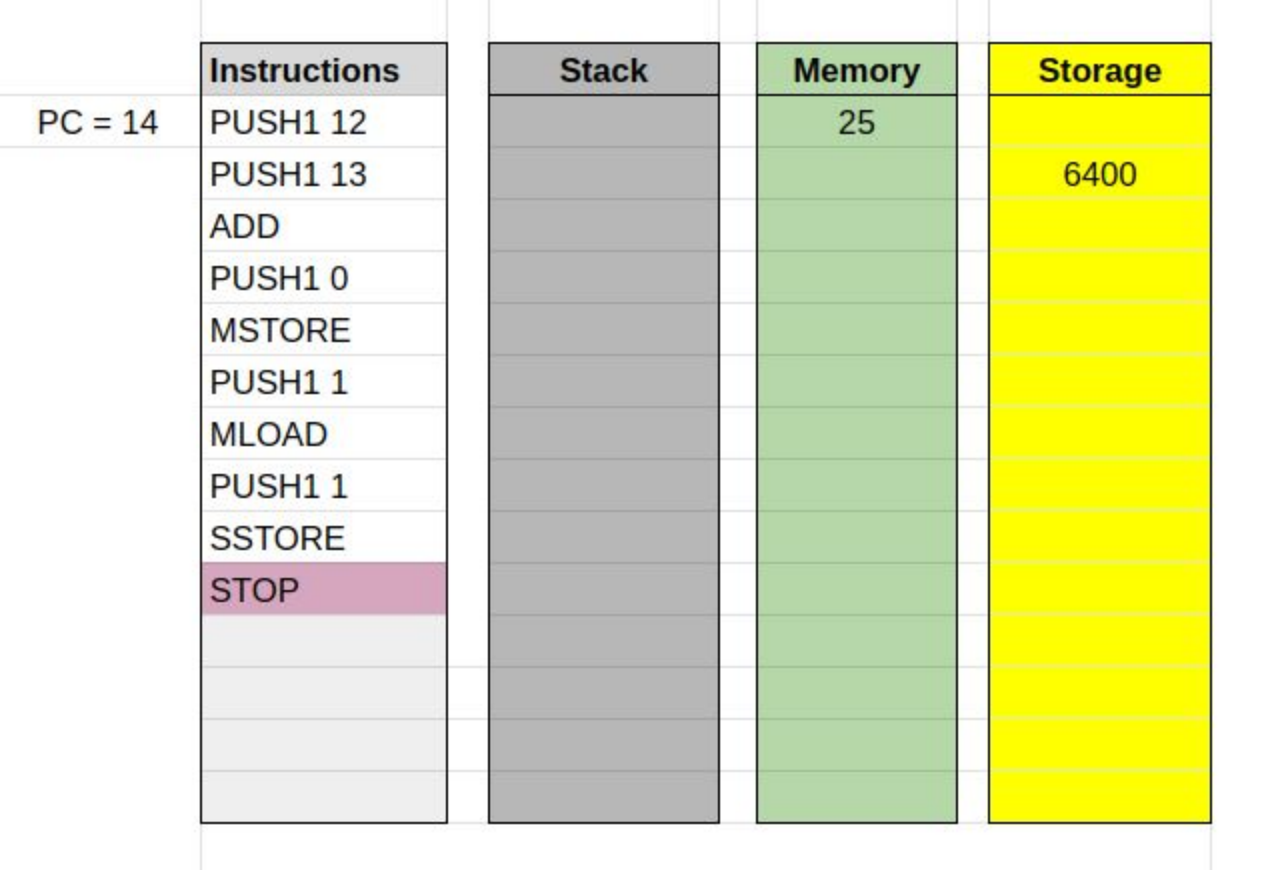
	Instructions	Stack	Memory	Storage
PC = 10	PUSH1 12	1	25	
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



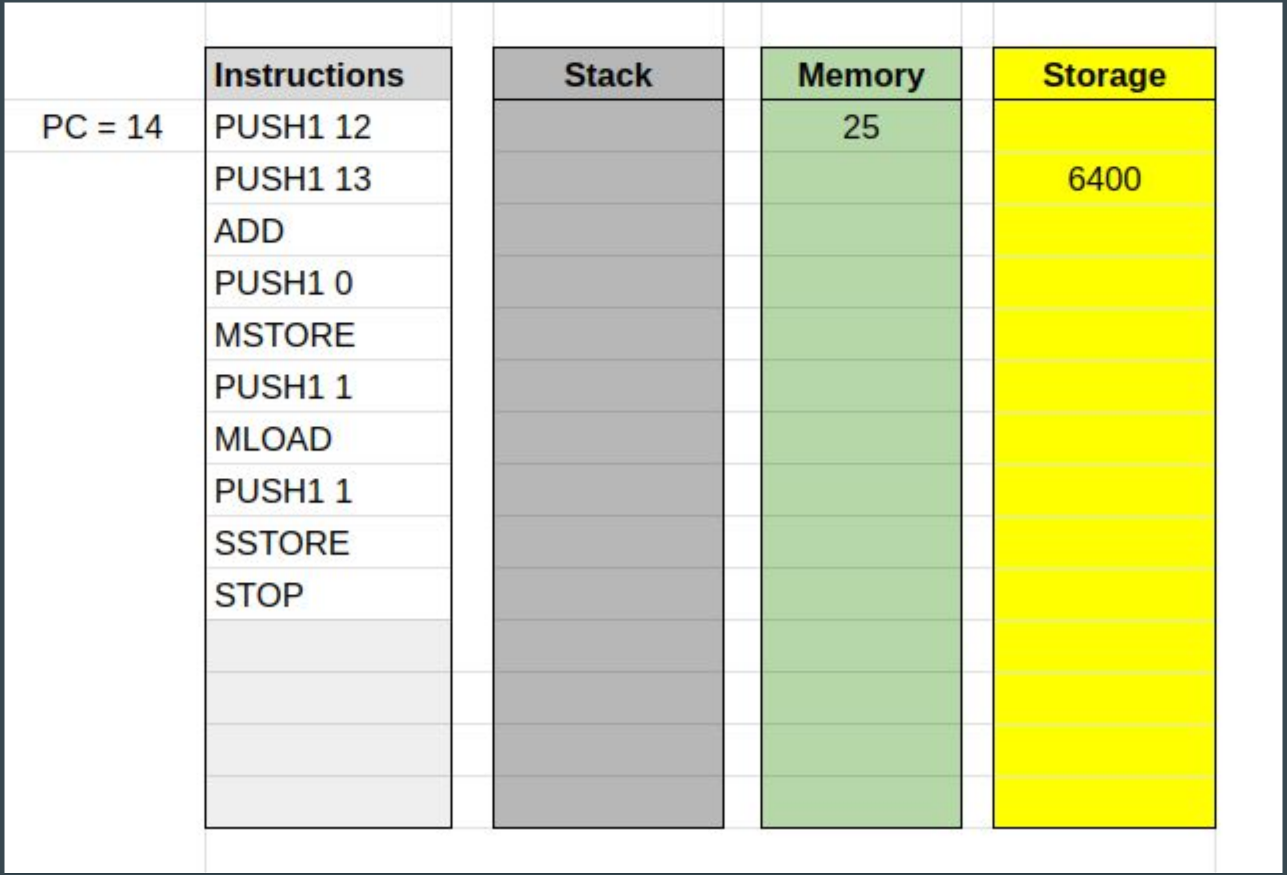
	Instructions	Stack	Memory	Storage
PC = 11	PUSH1 12	6400	25	
	PUSH1 13			
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



	Instructions	Stack	Memory	Storage
PC = 13	PUSH1 12	6400	25	
	PUSH1 13	1		
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



	Instructions	Stack	Memory	Storage
PC = 14	PUSH1 12		25	
	PUSH1 13			6400
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			



	Instructions	Stack	Memory	Storage
PC = 14	PUSH1 12		25	
	PUSH1 13			6400
	ADD			
	PUSH1 0			
	MSTORE			
	PUSH1 1			
	MLOAD			
	PUSH1 1			
	SSTORE			
	STOP			

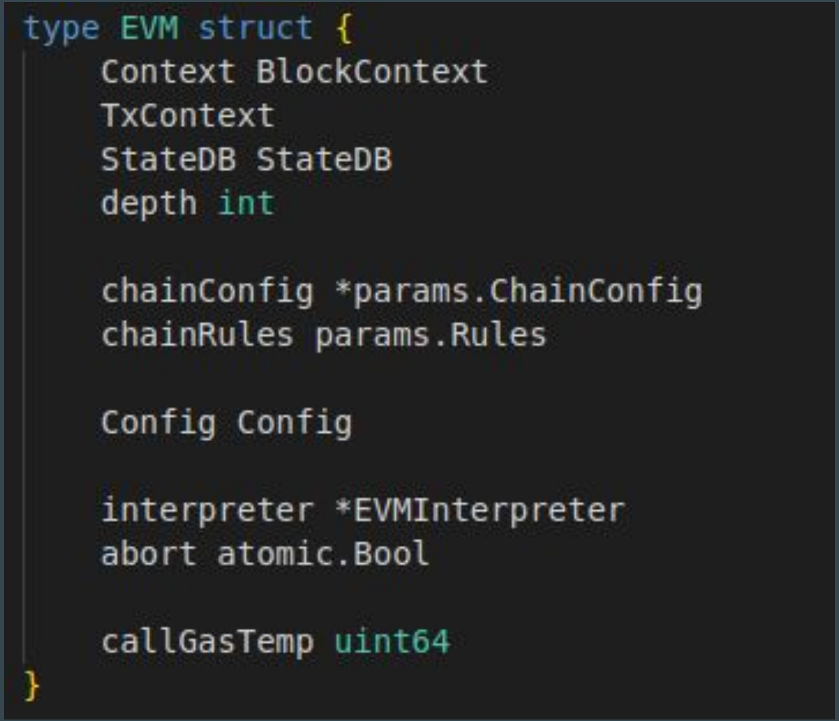
4. EVM in Go-Ethereum

- Go-Ethereum(Geth) is a Execution Engine
- Execution Engine
  - Listen new transactions
  - Execute transactions in EVM
  - Hold storage data
- Source code: https://github.com/ethereum/go-ethereum
- Use commit ***333dd956bfdf1d5086d38cceedbba25a366fb6ac*** in this slide

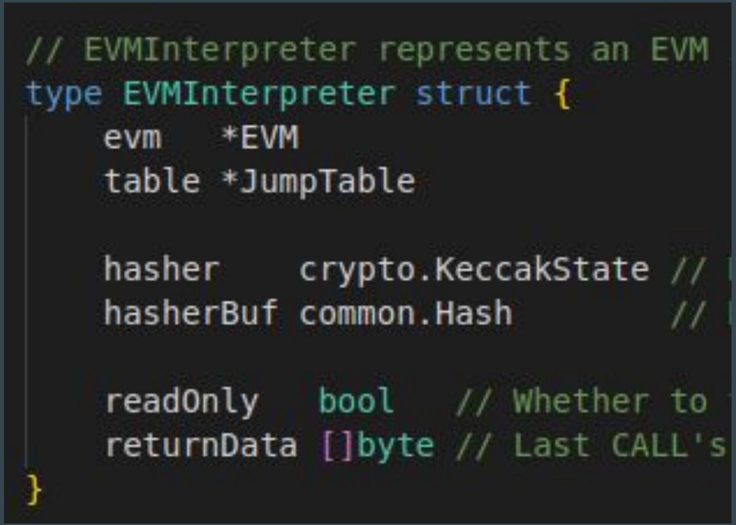
4. EVM in Go-Ethereum

- Content:
  - Data structures
    - EVM, EVM Interpreter, Transaction, Transaction processing flow
  - Deploy Smart Contract
  - Execute another kind of transactions
    - Send Ether
    - Call Smart Contract function

EVM (core/vm/evm.go) & EVM Interpreter (core/vm/interpreter.go)

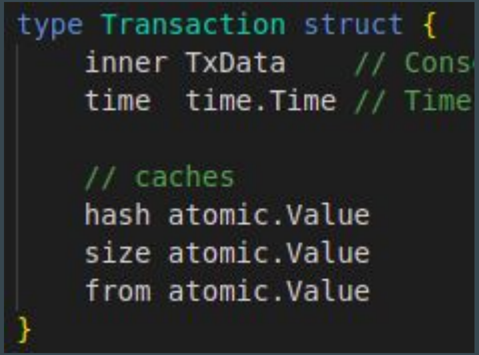


```go
type EVM struct {
    Context BlockContext
    TxContext
    StateDB StateDB
    depth int

    chainConfig *params.ChainConfig
    chainRules params.Rules

    Config Config

    interpreter *EVMInterpreter
    abort atomic.Bool

    callGasTemp uint64
}
```

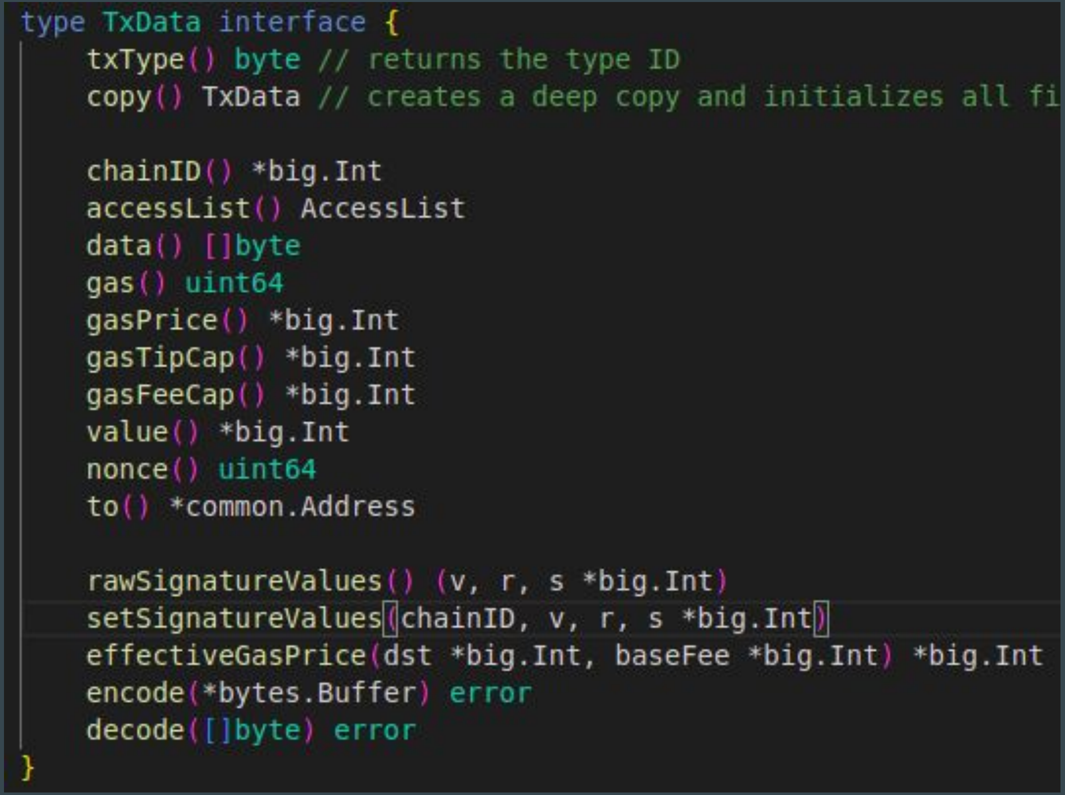


```go
// EVMInterpreter represents an EVM
type EVMInterpreter struct {
    evm   *EVM
    table *JumpTable

    hasher    crypto.KeccakState //
    hasherBuf common.Hash        //

    readOnly   bool   // Whether to
    returnData []byte // Last CALL's
}
```

Transaction



```
type Transaction struct {
    inner TxData    // Cons
    time  time.Time // Time

    // caches
    hash atomic.Value
    size atomic.Value
    from atomic.Value
}
```



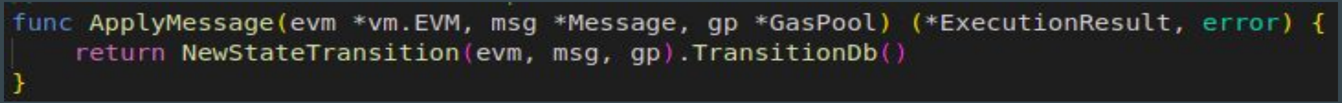
```
type TxData interface {
    txType() byte // returns the type ID
    copy() TxData // creates a deep copy and initializes all fi

    chainID() *big.Int
    accessList() AccessList
    data() []byte
    gas() uint64
    gasPrice() *big.Int
    gasTipCap() *big.Int
    gasFeeCap() *big.Int
    value() *big.Int
    nonce() uint64
    to() *common.Address

    rawSignatureValues() (v, r, s *big.Int)
    setSignatureValues(chainID, v, r, s *big.Int)
    effectiveGasPrice(dst *big.Int, baseFee *big.Int) *big.Int
    encode(*bytes.Buffer) error
    decode([]byte) error
}
```

Transaction

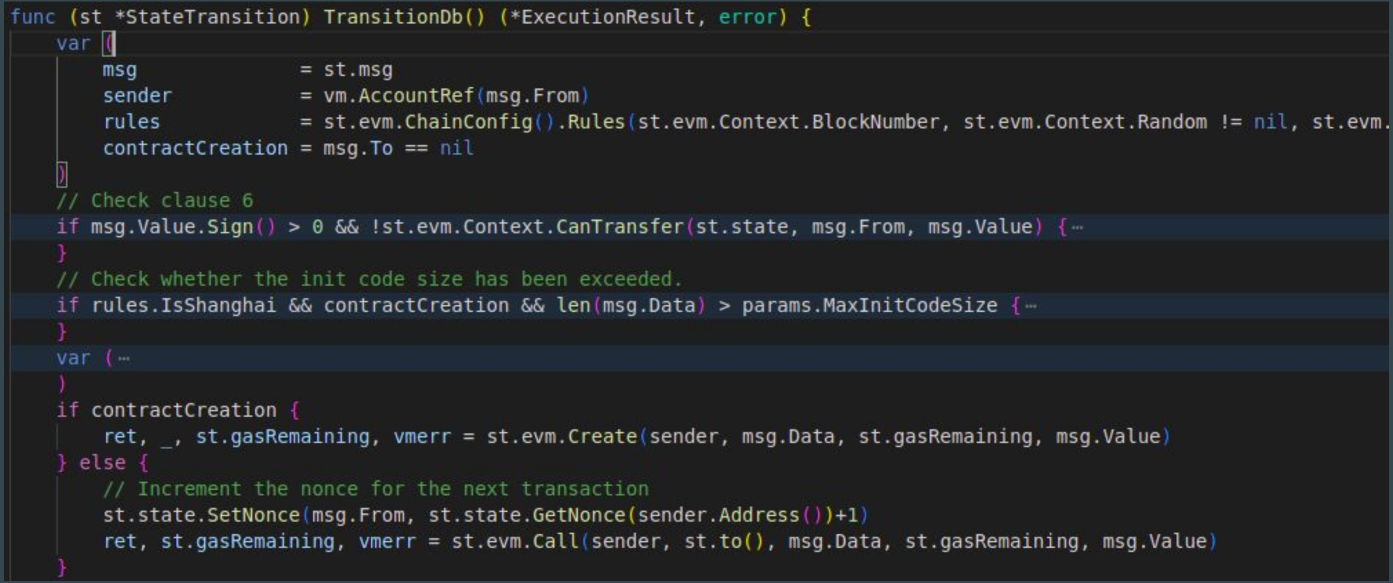
- Some of important fields:
  - From
  - To
  - Value
  - Input
  - ...
- Type of transactions:
  - To = null => deploy contract
  - To != null
    - Input = 0x => transfer Ether
    - Input != 0x => execute smart contract

Transaction processing flow in Geth

- Function ApplyMessage() in core/state_transition.go



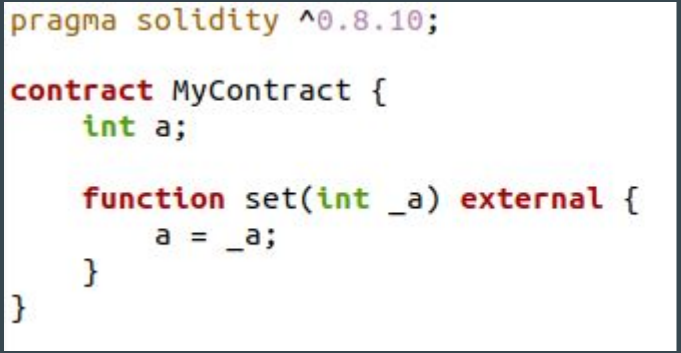
```go
func ApplyMessage(evm *vm.EVM, msg *Message, gp *GasPool) (*ExecutionResult, error) {
    return NewStateTransition(evm, msg, gp).TransitionDb()
}
```



```go
func (st *StateTransition) TransitionDb() (*ExecutionResult, error) {
    var (
        msg              = st.msg
        sender           = vm.AccountRef(msg.From)
        rules            = st.evm.ChainConfig().Rules(st.evm.Context.BlockNumber, st.evm.Context.Random != nil, st.evm.
        contractCreation = msg.To == nil
    )
    // Check clause 6
    if msg.Value.Sign() > 0 && !st.evm.Context.CanTransfer(st.state, msg.From, msg.Value) {⋯
    }
    // Check whether the init code size has been exceeded.
    if rules.IsShanghai && contractCreation && len(msg.Data) > params.MaxInitCodeSize {⋯
    }
    var (⋯
    )
    if contractCreation {
        ret, _, st.gasRemaining, vmerr = st.evm.Create(sender, msg.Data, st.gasRemaining, msg.Value)
    } else {
        // Increment the nonce for the next transaction
        st.state.SetNonce(msg.From, st.state.GetNonce(sender.Address())+1)
        ret, st.gasRemaining, vmerr = st.evm.Call(sender, st.to(), msg.Data, st.gasRemaining, msg.Value)
    }
```

Deploy Contract

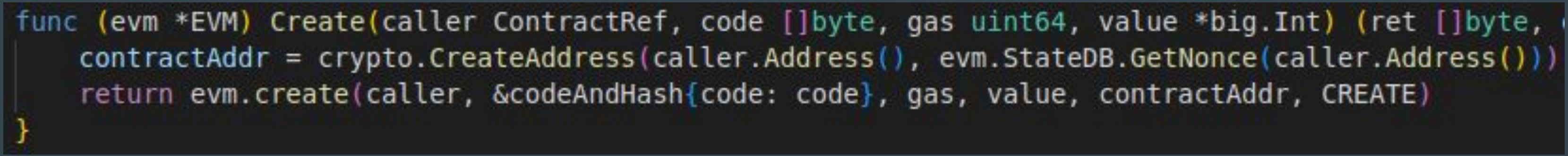
- Contract: is a program that can run on EVM
- There are 2 segments
    - Init segment
    - Runtime segment
- Demo: https://remix.ethereum.org/
- Source code:



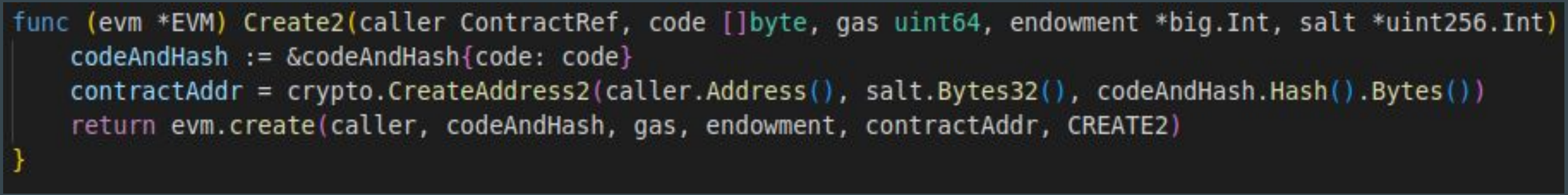
```
pragma solidity ^0.8.10;

contract MyContract {
    int a;

    function set(int _a) external {
        a = _a;
    }
}
```

Deploy Contract

- Handle by Create and Create2

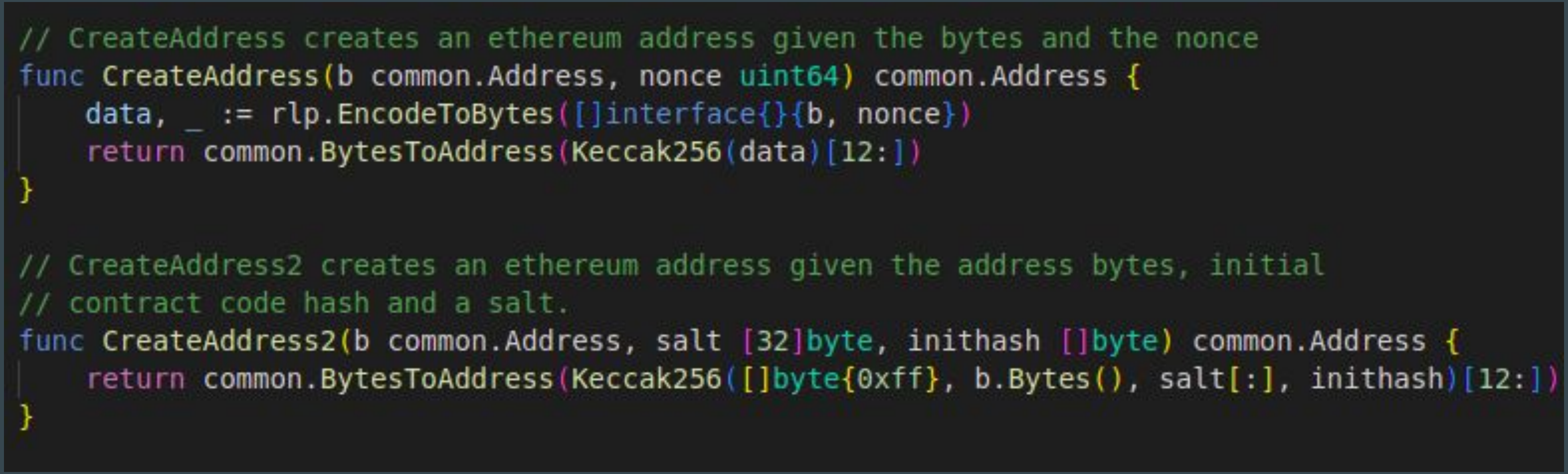


```go
func (evm *EVM) Create(caller ContractRef, code []byte, gas uint64, value *big.Int) (ret []byte, 
    contractAddr = crypto.CreateAddress(caller.Address(), evm.StateDB.GetNonce(caller.Address()))
    return evm.create(caller, &codeAndHash{code: code}, gas, value, contractAddr, CREATE)
}
```



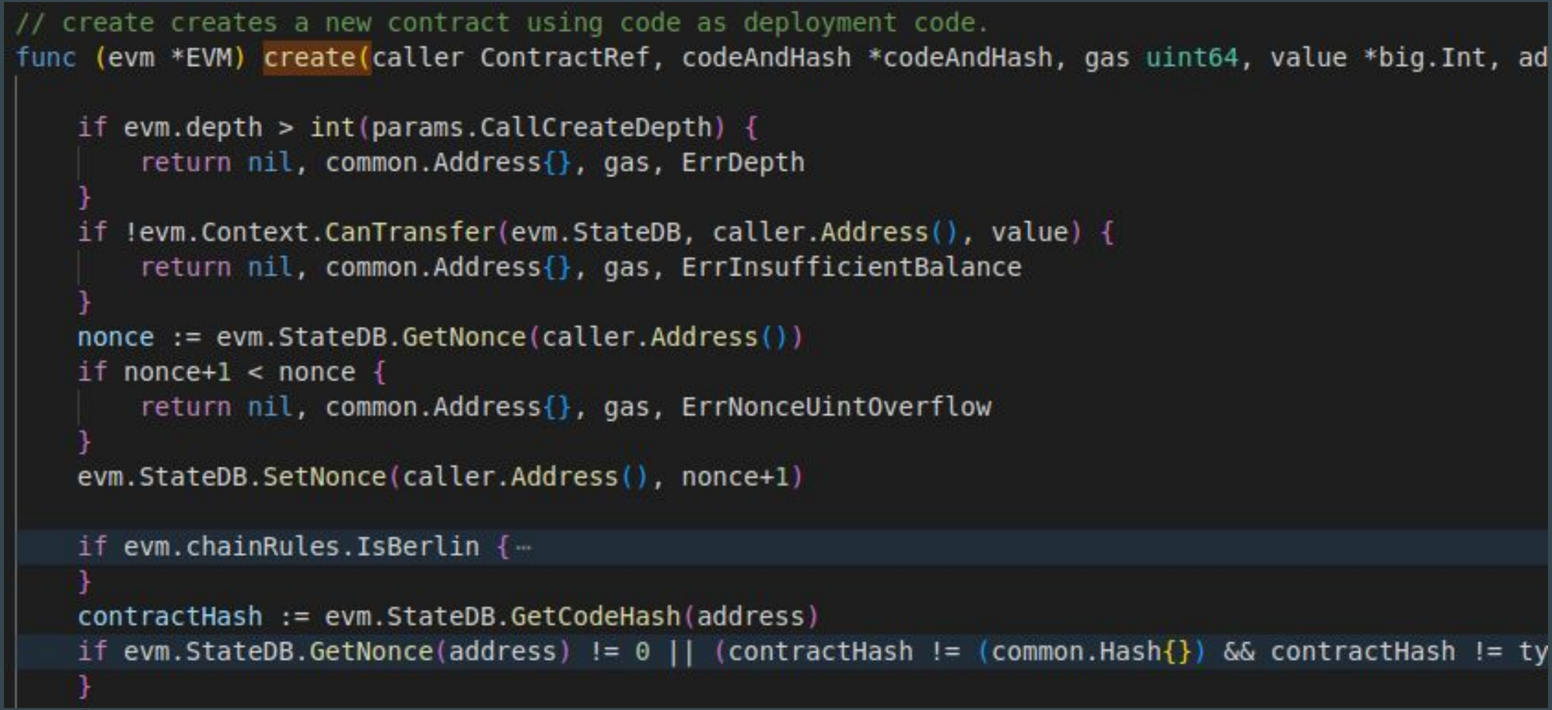
```go
func (evm *EVM) Create2(caller ContractRef, code []byte, gas uint64, endowment *big.Int, salt *uint256.Int)
    codeAndHash := &codeAndHash{code: code}
    contractAddr = crypto.CreateAddress2(caller.Address(), salt.Bytes32(), codeAndHash.Hash().Bytes())
    return evm.create(caller, codeAndHash, gas, endowment, contractAddr, CREATE2)
}
```

Deploy Contract: create contract’s address



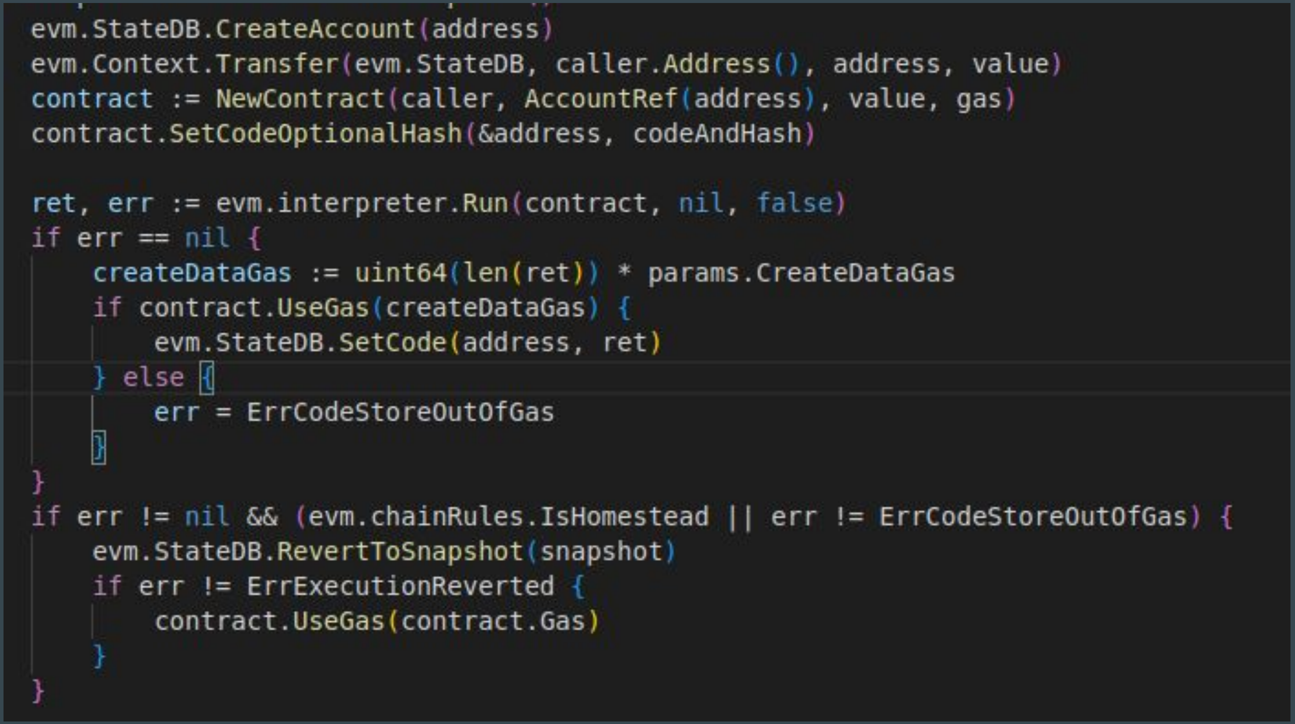
```go
// CreateAddress creates an ethereum address given the bytes and the nonce
func CreateAddress(b common.Address, nonce uint64) common.Address {
	data, _ := rlp.EncodeToBytes([]interface{}{b, nonce})
	return common.BytesToAddress(Keccak256(data)[12:])
}

// CreateAddress2 creates an ethereum address given the address bytes, initial
// contract code hash and a salt.
func CreateAddress2(b common.Address, salt [32]byte, inithash []byte) common.Address {
	return common.BytesToAddress(Keccak256([]byte{0xff}, b.Bytes(), salt[:], inithash)[12:])
}
```

Deploy contract: function create() (core/vm/evm.go)



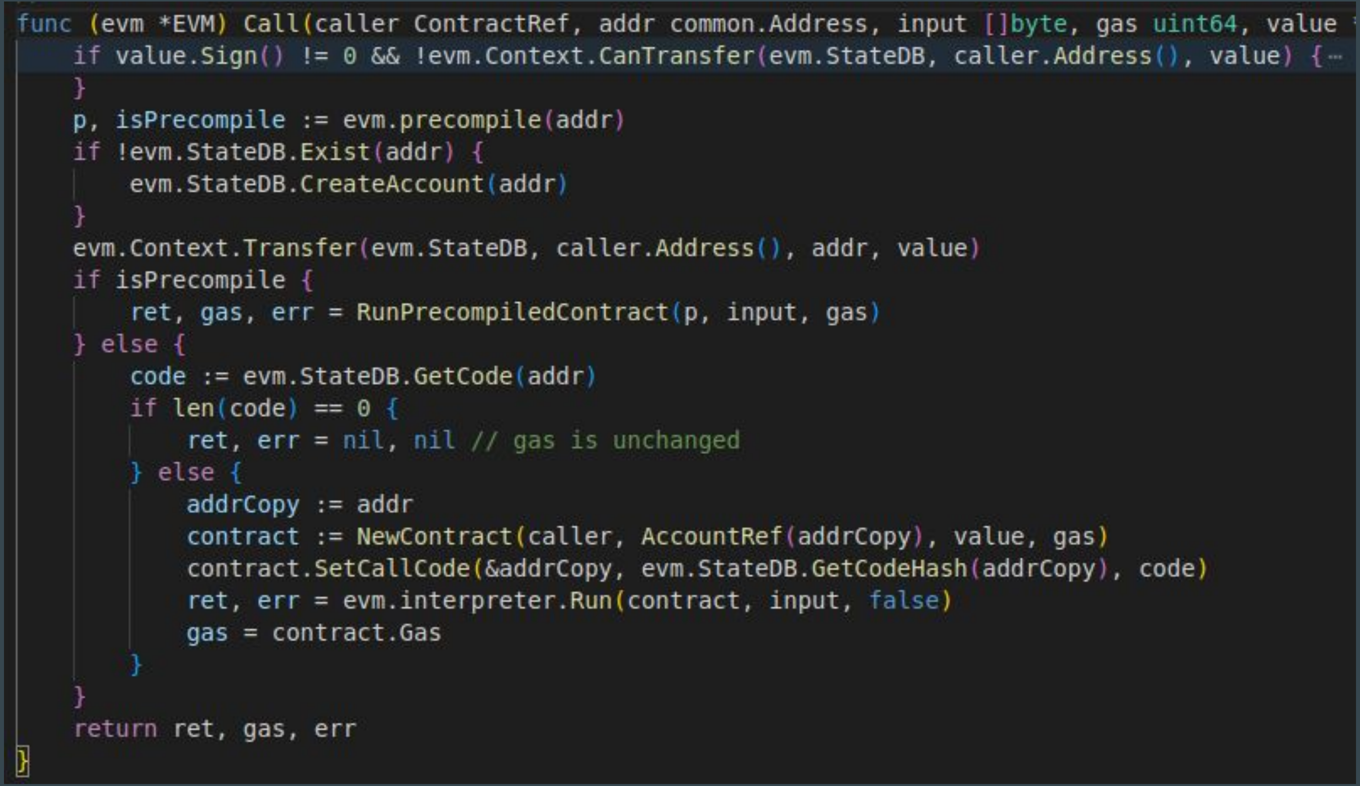
```go
// create creates a new contract using code as deployment code.
func (evm *EVM) create(caller ContractRef, codeAndHash *codeAndHash, gas uint64, value *big.Int, ad

    if evm.depth > int(params.CallCreateDepth) {
        return nil, common.Address{}, gas, ErrDepth
    }
    if !evm.Context.CanTransfer(evm.StateDB, caller.Address(), value) {
        return nil, common.Address{}, gas, ErrInsufficientBalance
    }
    nonce := evm.StateDB.GetNonce(caller.Address())
    if nonce+1 < nonce {
        return nil, common.Address{}, gas, ErrNonceUintOverflow
    }
    evm.StateDB.SetNonce(caller.Address(), nonce+1)

    if evm.chainRules.IsBerlin {⋯
    }
    contractHash := evm.StateDB.GetCodeHash(address)
    if evm.StateDB.GetNonce(address) != 0 || (contractHash != (common.Hash{}) && contractHash != ty
    }
```

Deploy contract: function create() (core/vm/evm.go)



```go
evm.StateDB.CreateAccount(address)
evm.Context.Transfer(evm.StateDB, caller.Address(), address, value)
contract := NewContract(caller, AccountRef(address), value, gas)
contract.SetCodeOptionalHash(&address, codeAndHash)

ret, err := evm.interpreter.Run(contract, nil, false)
if err == nil {
    createDataGas := uint64(len(ret)) * params.CreateDataGas
    if contract.UseGas(createDataGas) {
        evm.StateDB.SetCode(address, ret)
    } else {
        err = ErrCodeStoreOutOfGas
    }
}
if err != nil && (evm.chainRules.IsHomestead || err != ErrCodeStoreOutOfGas) {
    evm.StateDB.RevertToSnapshot(snapshot)
    if err != ErrExecutionReverted {
        contract.UseGas(contract.Gas)
    }
}
```

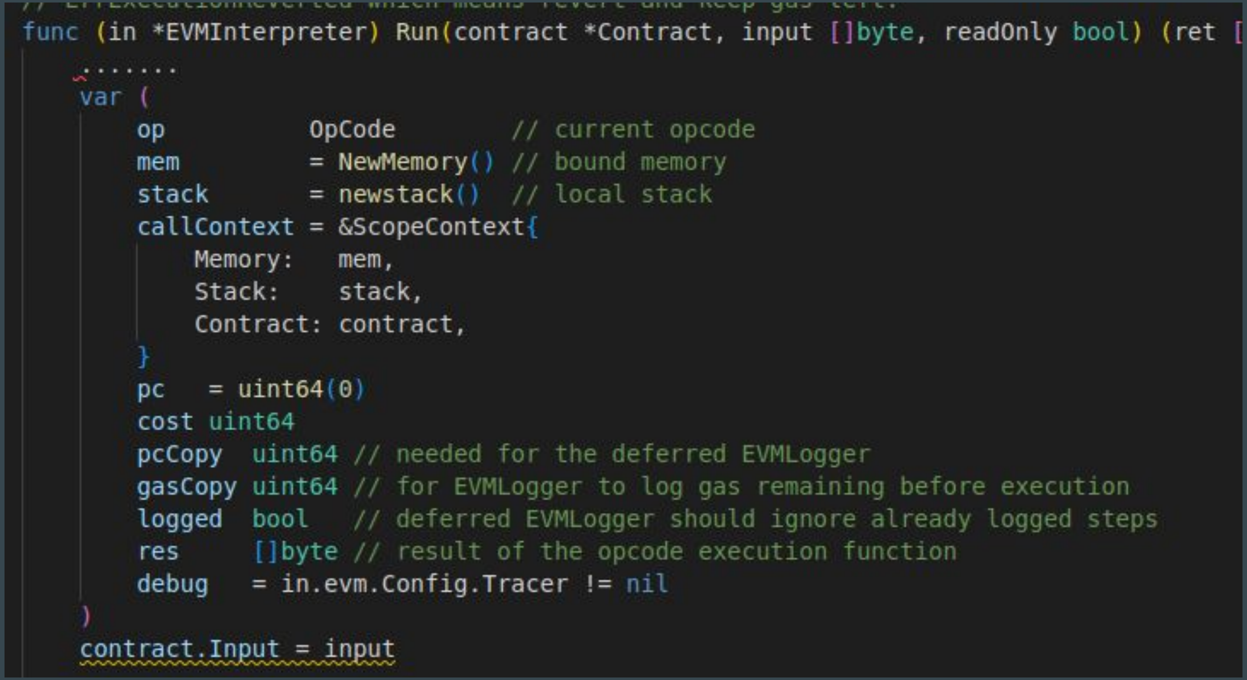
Execute transaction



```
func (evm *EVM) Call(caller ContractRef, addr common.Address, input []byte, gas uint64, value
    if value.Sign() != 0 && !evm.Context.CanTransfer(evm.StateDB, caller.Address(), value) {⋯
    }
    p, isPrecompile := evm.precompile(addr)
    if !evm.StateDB.Exist(addr) {
        evm.StateDB.CreateAccount(addr)
    }
    evm.Context.Transfer(evm.StateDB, caller.Address(), addr, value)
    if isPrecompile {
        ret, gas, err = RunPrecompiledContract(p, input, gas)
    } else {
        code := evm.StateDB.GetCode(addr)
        if len(code) == 0 {
            ret, err = nil, nil // gas is unchanged
        } else {
            addrCopy := addr
            contract := NewContract(caller, AccountRef(addrCopy), value, gas)
            contract.SetCallCode(&addrCopy, evm.StateDB.GetCodeHash(addrCopy), code)
            ret, err = evm.interpreter.Run(contract, input, false)
            gas = contract.Gas
        }
    }
    return ret, gas, err
}
```

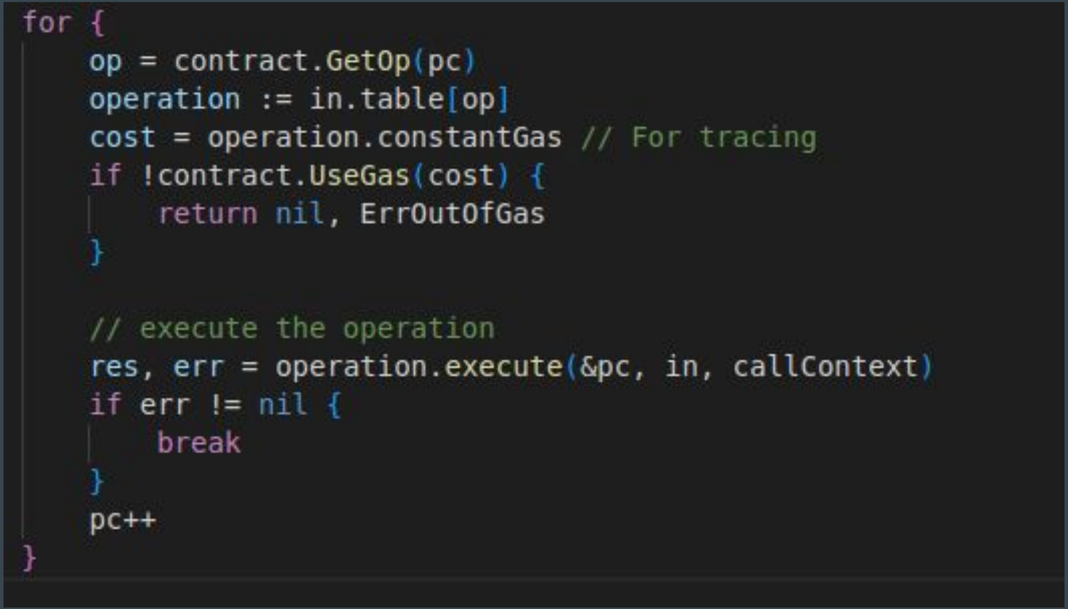
Execute transaction: run smart contract with EVM

- Run() in core/vm/interpreter.go



```go
// ErrExecutionReverted which means revert and keep gas left.
func (in *EVMInterpreter) Run(contract *Contract, input []byte, readOnly bool) (ret [
    .......
    var (
        op          OpCode        // current opcode
        mem         = NewMemory() // bound memory
        stack       = newstack()  // local stack
        callContext = &ScopeContext{
            Memory:   mem,
            Stack:    stack,
            Contract: contract,
        }
        pc   = uint64(0)
        cost uint64
        pcCopy  uint64 // needed for the deferred EVMLogger
        gasCopy uint64 // for EVMLogger to log gas remaining before execution
        logged  bool   // deferred EVMLogger should ignore already logged steps
        res     []byte // result of the opcode execution function
        debug   = in.evm.Config.Tracer != nil
    )
    contract.Input = input
```

Execute transaction: run smart contract with EVM



```
for {
    op = contract.GetOp(pc)
    operation := in.table[op]
    cost = operation.constantGas // For tracing
    if !contract.UseGas(cost) {
        return nil, ErrOutOfGas
    }

    // execute the operation
    res, err = operation.execute(&pc, in, callContext)
    if err != nil {
        break
    }
    pc++
}
```

5. Q&A

THANK YOU

References

1. https://ethereum.org/en/
2. https://ethervm.io/
3. https://en.wikipedia.org/wiki/Stack_machine
4. https://github.com/ethereum/go-ethereum
5. https://github.com/nguyenzung/blockchain-training