Ethereum EVM illustrated

exploring some mental models and implementations

Takenobu T.
NOTE
- Please refer to the official documents in detail.
- This information is current as of Mar, 2018.
- Still work in progress.
1. Introduction
- Blockchain
- World state
- Account
- Transaction
- Message
- Decentralised database
- Atomicity and order

2. Virtual machine
- Ethereum virtual machine (EVM)
- Message call
- Exception
- Gas and fee
- Input and output
- Byte order
- Instruction set
- Miscellaneous

Appendix A : Implementation
- Source code in Geth
- EVM developer utility
- Solidity ABI

Appendix B : User interface
- Web3 API
- Geth, Mist, Solc, Remix, Truffle, ...

Appendix C :
- Markle tree and RLP
- Consensus

References
1. Introduction
1. Introduction

Blockchain
Ethereum can be viewed as a transaction-based state machine.
A transaction represents a valid arc between two states.

References: [E1] Ch.2, [E3]
Transactions are collated into blocks.
A block is a package of data.

References: [E1] Ch.2, Ch.4, [E2], [E3]
From the viewpoint of the states, Ethereum can be seen as a state chain.
From the viewpoint of the implementation, Ethereum can also be seen as a chain of blocks, so it is `BLOCKCHAIN`.

References: [E1] Ch.2, Ch.4, [E2], [E3], [W3]
From the viewpoint of the ledger, Ethereum can also be seen as a stack of transactions.

References: [E1] Ch.2, Ch.4, [E2], [E3], [W3]
1. Introduction

World state
The world state is a mapping between address and account state.
Several views of world state

Mapping view

Table view

Object view

References: [E1] Ch.4
1. Introduction

Account
An account is an object in the world state.
An account is a mapping between address and account state.
Account state

An account state could contain EVM code and storage.
Two practical types of account

Externally owned account (EOA)
- EOA is controlled by a private key.

Contract account
- Contains EVM code.
- Contains storage.
- Autonomous object.

References: [E1] Ch.4
Two practical types of account

Externally owned account (EOA)
- Address
  - nonce
  - balance
  - storage hash
  - code hash

Contract account
- Address
  - nonce
  - balance
  - storage hash
  - code hash

EOA is controlled by a private key.
EOA cannot contain EVM code.

Contract contains EVM code.
Contract is controlled by EVM code.

References: [E1] Ch.4
Address of account

Externally owned account (EOA)

- Private key
- Public key
- hash
- Address
  160 bits

Contract account

- Sender address
- Nonce
- RLP, KEC, right 160 bits
- Address
  160 bits

A 160-bit code used for identifying accounts.

References: [E1] Ch.4, Ch.7, [E2]
1. Introduction

Transaction
A transaction is a single cryptographically-signed instruction.
A transaction to world state

A transaction is submitted by external actor.

References: [E1] Appendix A
There are two practical types of transaction, contract creation and message call.
Contract creation


drawings of accounts and world states

Transaction

\textbf{init code}

World state $\sigma_t$

World state $\sigma_{t+1}$

References : [E1] Ch.7
Field of a transaction

- nonce
- gasPrice
- gasLimit
- to
- value
- v, r, s
- init or data

160 bits address or 0 if contract creation
transferred wei (ether)
contract creation or message call

References: [E1] Ch.4
1. Introduction

Message
Message is passed between two Accounts.
Message is Data (as a set of bytes) and Value (specified as Ether).
Transaction triggers an associated message.

EVM can also send a message.
Four cases of message

By Transaction  From EOA

To EOA

By EVM code  From CA

To CA

References: [E1] Ch.8
1. Introduction

Decentralised database
A blockchain is a globally shared, transactional database.
A blockchain is a globally shared, decentralised, transactional database.

References: [E3], [E7] Ch.7
Decentralised nodes constitute Ethereum P2P network.
External actors access the Ethereum world through Ethereum nodes.
1. Introduction

Atomicity and order
A transaction is an atomic operation. Can't divide or interrupt.

That is, All (complete done) or Nothing (zero effect).
Transactions cannot be overlapped.
Transactions must be executed sequentially.
Order of transactions

Transaction order is not guaranteed.

External actor A
1st submitted
Transaction

External actor B
2nd submitted
Transaction

Blockchain
3rd submitted
Transaction

Transaction order is not guaranteed.
Miner can determine the order of transactions in a block.
The order between blocks is determined by a consensus algorithm such as PoW.
2. Virtual machine
2. Virtual machine

Ethereum virtual machine (EVM)
Ethereum virtual machine

EVM code is executed on Ethereum Virtual Machine (EVM).

References: [E1] Ch.9, Appendix H
The Ethereum Virtual Machine is the runtime environment for smart contracts in Ethereum.
The EVM is a simple stack-based architecture.
There are several resources as space.

References: [E1] Ch.9, Appendix H, [E3], [E7], [W2]
All operations are performed on the stack.
Access with many instructions such as PUSH/POP/COPY/SWAP, ...
Memory is linear and can be addressed at byte level. Access with MSTORE/MSTORE8/MLOAD instructions. All locations in memory are well-defined initially as zero.

References: [E1] Ch.9, Appendix H, [E7], [W2]
Storage is a key-value store that maps 256-bit words to 256-bit words. Access with SSTORE/SLOAD instructions. All locations in storage are well-defined initially as zero.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key 1</td>
<td>Value 1</td>
</tr>
<tr>
<td>Key 2</td>
<td>Value 2</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Key n</td>
<td>Value n</td>
</tr>
</tbody>
</table>
EVM Code is the bytecode that the EVM can natively execute.

Assembly view

PUSH1 e0
PUSH1 02
EXP
PUSH1 00
CALLDATALOAD :

Bytecode view

0x60e060020a600035...

References: [E1] Ch.9, Appendix H
Execution model

EVM

EVM code

instructions

PC

operations

Stack

stack top

Memory

Gas avail

push/pop/...

random access

(Account) storage

References: [E1] Ch.9, Appendix H
2. Virtual machine

Message call
EVM can send a message to other account.
The depth of message call is limited to less than 1024 levels.

References: [E1] Ch.8, Appendix A
Instructions for Message call

Message call is triggered by CALL instruction.
Arguments and return values are passed using memory.

References: [E1] Ch.8, Ch.9
2. Virtual machine

Exception
If an exception occurs in the EVM, the state is not updated.

The behavior if an exception occurs is subtle. [E9]

References: [E9], [E1] Ch.9, Appendix H
Exception

EVM

PC

EVM code

operations

Stack

Memory

Gas avail

out-of-gas

invalid jump destination

invalid instruction

stack underflow

References : [E1] Ch.9, Appendix H
2. Virtual machine

Gas and fee
All programmable computation in Ethereum is subject to fees (denominated in gas).
2. Virtual machine

Input and output
EVM can input external data from a message call.
EVM can output log. EVM can also return values to Caller EVM.

References : [E1] Ch.9, Appendix H
Instructions for input data

input data

CALLDATALOAD

CALLDATACOPY

Stack

Memory

EVM

References: [E1] Ch.7, Ch.9
2. Virtual machine

Byte order
EVM is big endian order (network byte order).

References: [E1] Ch.9, Appendix H, [E7], [W2]
EVM is big endian order (network byte order).

References: [E1] Ch.9, Appendix H, [E7], [W2]
EVM is big endian order (network byte order).

References: [E1] Ch.9, Appendix H, [E7], [W2]
Byte order of BYTE and SIGNEXTEND instruction

BYTE instruction counts from MSB.

SIGNEXTEND instruction counts from LSB.

References: [E1] Ch.9, Appendix H, [E7], [W2]
Byte order of PUSH instructions

**PUSH1** 0x01

```
<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
</tr>
<tr>
<td>MSB</td>
</tr>
<tr>
<td>LSB</td>
</tr>
</tbody>
</table>
```

right-aligned, big endian

**PUSH4** 0x01020304

```
<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>01020304</td>
</tr>
<tr>
<td>MSB</td>
</tr>
<tr>
<td>LSB</td>
</tr>
</tbody>
</table>
```

right-aligned, big endian

**PUSH32** 0x010203041f20

```
<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>010203041f20</td>
</tr>
<tr>
<td>MSB</td>
</tr>
<tr>
<td>LSB</td>
</tr>
</tbody>
</table>
```

right-aligned, big endian

References: [E1] Ch.9, Appendix H, [E7], [W2]
2. Virtual machine

Instruction set
* Basically, 256-bit operation.

* Contract creation and destruct
  * CREATE, DELEGATECALL

* Hash
  * SHA3

* Shift operation
  * using MUL or DIV, SDIV

* Div operation
  * without zero divisional exception

* ...
There are several copy instructions for inter spaces.

References: [E1] Ch.8, Ch.9
Shift by MUL, DIV and SDIV

Left shift is represented by MUL instruction.

Right shift is represented by DIV and SDIV instruction.

References: [E1] Ch.9, Appendix H, [E7], [W2]
2. Virtual machine

Miscellaneous
EVM code generation

Solidity source -> Solidity compiler
Viper source -> Viper compiler
LLL source -> LLL compiler

EVM code

Ethereum virtual machine code

References: [E7]
Ethereum virtual machine layer

- **Code**: EVM code
- **Virtual Machine**: Ethereum Virtual Machine
- **Runtime System (Process)**: Ethereum node (Geth, Parity, ...)
- **Software**
- **Hardware**: Physical Processor (x86, ARM, ...)

References: [E1] Ch.9
The eWASM is next generation VM.
Appendix A
Appendix A

Source code in Geth
Block header

```go
type Header struct {
    ParentHash  common.Hash   `json:"parentHash"       gencodec:"required"`
    UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
    Coinbase   common.Address `json:"miner"            gencodec:"required"`
    Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
    TxHash      common.Hash    `json:"transactionsRoot" gencodec:"required"`
    ReceiptHash common.Hash    `json:"receiptsRoot"     gencodec:"required"`
    Bloom       Bloom          `json:"logsBloom"        gencodec:"required"`
    Difficulty  *big.Int       `json:"difficulty"       gencodec:"required"`
    Number      *big.Int       `json:"number"            gencodec:"required"`
    GasLimit    uint64         `json:"gasLimit"         gencodec:"required"`
    GasUsed     uint64         `json:"gasUsed"          gencodec:"required"`
    Time        *big.Int       `json:"timestamp"        gencodec:"required"`
    Extra       []byte         `json:"extraData"        gencodec:"required"`
    MixDigest   common.Hash    `json:"mixHash"          gencodec:"required"`
    Nonce       BlockNonce     `json:"nonce"            gencodec:"required"`
}
```
Transaction

[core/types/transaction.go]

type txdata struct {
    AccountNonce uint64 `json:"nonce" gencodec:"required"`
    Price        *big.Int   `json:"gasPrice" gencodec:"required"`
    GasLimit     uint64     `json:"gas" gencodec:"required"`
    Recipient    *common.Address `json:"to" rlp:"nil"`
    Amount       *big.Int    `json:"value" gencodec:"required"`
    Payload      []byte     `json:"input" gencodec:"required"`

    // Signature values
    V  *big.Int  `json:"v" gencodec:"required"`
    R  *big.Int  `json:"r" gencodec:"required"`
    S  *big.Int  `json:"s" gencodec:"required"`

    // This is only used when marshaling to JSON.
    Hash *common.Hash `json:"hash" rlp:"-"`
}

References: [C1]
World state

(core/state/statedb.go)

type StateDB struct {
    db    Database
    trie  Trie

    stateObjects  map[common.Address]*stateObject
    stateObjectsDirty  map[common.Address]struct{}

    dbErr  error

    refund  uint64

    thash, bhash  common.Hash
    txIndex       int
    logs  map[common.Hash][]*types.Log
    logSize       uint

    preimages  map[common.Hash][]byte
}

References: [C1]
Account object (state object)

[c0re/state/state_object.go]

```go
type stateObject struct {
    address common.Address
    addrHash common.Hash
    data Account
    db       *StateDB
    dbErr error
    trie Trie // storage trie, which becomes non-nil on first access
    code Code // contract bytecode, which gets set when code is loaded
    cachedStorage Storage // Storage entry cache to avoid duplicate reads
    dirtyStorage Storage // Storage entries that need to be flushed to disk
    dirtyCode bool // true if the code was updated
    suicided bool
    touched bool
    deleted bool
    onDirty  func(addr common.Address)
}
```

References: [C1]
Account state, Code and Storage

(core/state/state_object.go)

```go
type Account struct {
    Nonce    uint64
    Balance  *big.Int
    Root     common.Hash // merkle root of the storage trie
    CodeHash []byte
}
```

- **Account state**

```go
type Code []byte
```
- **EVM code**

```go
type Storage map[common.Hash]common.Hash
```
- **Account storage**
Stack and Memory

```
[core/vm/stack.go]

type Stack struct {
    data [*big.Int]
}

func newstack() *Stack {
    return &Stack{data: make([*big.Int, 0, 1024])}
}
```

```
[core/vm/memory.go]

type Memory struct {
    store       []byte
    lastGasCost uint64
}

func NewMemory() *Memory {
    return &Memory{
}
}
```
Instruction operation (arithmetic and stack)

[core/vm/instruction.go]

Arithmetic operation

```go
func opAdd(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack) ([]byte, error) {
    x, y := stack.pop(), stack.pop()
    stack.push(math.U256(x.Add(x, y)))

    evm.interpreter.intPool.put(y)

    return nil, nil
}
```

Stack operation

```go
func opPop(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack) ([]byte, error) {
    evm.interpreter.intPool.put(stack.pop())
    return nil, nil
}
```
Instruction operation (memory and storage)

Memory operation

```go
func opMload(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack) ([]byte, error) {
    offset := stack.pop()
    stack.push(val)

    evm.interpreter.intPool.put(offset)
    return nil, nil
}
```

Storage operation

```go
func opSload(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack) ([]byte, error) {
    loc := common.BigToHash(stack.pop())
    val := evm.StateDB.GetState(contract.Address(), loc).Big()
    stack.push(val)
    return nil, nil
}
```
func opCall(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack *Stack) ([]byte, error) {
    evm.interpreter.intPool.put(stack.pop())
    gas := evm.callGasTemp
    // Pop other call parameters.
    addr, value, inOffset, inSize, retOffset, retSize := stack.pop(),
        stack.pop(), stack.pop(), stack.pop(), stack.pop(), stack.pop()
    toAddr := common.BigToAddress(addr)
    value = math.U256(value)
    // Get the arguments from the memory.
    args := memory.Get(inOffset.Int64(), inSize.Int64())

    if value.Sign() != 0 {
        gas += params.CallStipend
    }
    ret, returnGas, err := evm.Call(contract, toAddr, args, gas, value)
    if err != nil {
        return nil
    }
    return ret
}
Gas

const (
    GasQuickStep uint64 = 2
    GasFastestStep uint64 = 3
    GasFastStep uint64 = 5
    GasMidStep uint64 = 8
    GasSlowStep uint64 = 10
    GasExtStep uint64 = 20
    GasReturn uint64 = 0
    GasStop uint64 = 0
    GasContractByte uint64 = 200
)

func gasSStore(gt params.GasTable, evm *EVM, contract *Contract, stack *Stack, mem *Memory, memorySize uint64) (uint64, error) {
    var (
        y, x = stack.Back(1), stack.Back(0)
        val = evm.StateDB.GetState(contract.Address(),
    )
func (in *Interpreter) Run(contract *Contract, input []byte) (ret []byte, err error) {
    // Increment the call depth which is restricted to 1024
    in.evm.depth++
    defer func() { in.evm.depth-- }()

    in.returnData = nil

    if len(contract.Code) == 0 {
        return nil, nil
    }

    codehash := contract.CodeHash // codehash is used when doing jump dest caching
    if codehash == (common.Hash{}) {
        codehash = crypto.Keccak256Hash(contract.Code)
    }

    var (
        op OpCode,  // current opcode
        mem = NewMemory() // bound memory
        stack = newstack() // local stack
    )

    increment call depth

create Memory
create Stack
func ApplyTransaction(config *params.ChainConfig, bc *BlockChain, author *common.Address, gp *GasPool, statedb *state.StateDB, header *types.Header, tx *types.Transaction, usedGas *uint64, cfg vm.Config) (*types.Receipt, uint64, error) {

    msg, err := tx.AsMessage(types.MakeSigner(config, header.Number))
    if err != nil {
        return nil, 0, err
    }

    // Create a new context to be used in the EVM environment
    context := NewEVMContext(msg, header, bc, author)

    // Create a new environment which holds all relevant information
    // about the transaction and calling mechanisms.
    vmenv := vm.NewEVM(context, statedb, config, cfg)

    // Apply the transaction to the current state (included in the env)
    _, gas, failed, err := ApplyMessage(vmenv, msg, gp)
    if err != nil {
        return nil, 0, err
    }

    // Update the state with pending changes
    var root []byte
    if config.IsByzantium(header.Number) {
    
    
create EVM}
func NewInterpreter(evm *EVM, cfg Config) *Interpreter {
    if !cfg.JumpTable[STOP].valid {
        switch {
            case evm.ChainConfig().IsByzantium(evm.BlockNumber):
                cfg.JumpTable = byzantiumInstructionSet
            case evm.ChainConfig().IsHomestead(evm.BlockNumber):
                cfg.JumpTable = homesteadInstructionSet
            default:
                cfg.JumpTable = frontierInstructionSet
        }
    }

    var MainnetChainConfig = &ChainConfig{
        ChainId:       big.NewInt(1),
        HomesteadBlock: big.NewInt(1150000),
        DAOForkBlock:   big.NewInt(1920000),
        DAOForkSupport: true,
        EIP150Block:    big.NewInt(2463000),
        EIP150Hash:     common.HexToHash("0x2086799aeebeae135c246c65021c82b4e15a2c451340993a"),
        EIP155Block:    big.NewInt(2675000),
        EIP158Block:    big.NewInt(2675000),
        ByzantiumBlock: big.NewInt(4370000),
        :
Bootstrap of EVM in Geth

References: [C1]
Appendix A

EVM developer utility
The go-ethereum project provides evm utility command.

Compile EVM assembly code

```bash
$ cat sample.asm
push 0x1
push 0x2
add

$ evm compile sample.asm
6001600201
```

Disassemble EVM bytecode

```bash
$ cat sample.bin
6001600201

$ evm disasm sample.bin
000000: PUSH1 0x01
000002: PUSH1 0x02
000004: ADD
```
Example of evm command

Run EVM assembly code

$ evm --debug run sample.asm

#### TRACE ####
PUSH1 pc=00000000 gas=10000000000 cost=3
PUSH1 pc=00000002 gas=999999997 cost=3
Stack:
00000000 0000000000000000000000000000000000000000000000000000001
ADD pc=00000004 gas=999999994 cost=3
Stack:
00000000 0000000000000000000000000000000000000000000000000000002
00000001 0000000000000000000000000000000000000000000000000000001
STOP pc=00000005 gas=999999991 cost=0
Stack:
00000000 0000000000000000000000000000000000000000000000000000003

#### LOGS ####
Appendix A

Solidity ABI
Solidity Application Binary Interface

Function selector
(SHA-3 hash)

Arguments, ...

input data

CALLDATALOAD
CALLDATALOAD
CALLDATACOPY

 CALLDATALOAD

function dispatching

JUMPI

function A

function B

EVM code

Stack or Memory

EVM

References: [E7]. Ch.7, [E1] Ch.9, Appendix H, [W4], [W2]
Appendix B
Appendix B

Web3 API
Web3 API and client

Ethereum clients access to Ethereum network via Web3 API.

References: [E8], [C1], [C3], [C4], [C5], [C6]
Ethereum clients access to Ethereum network via Web3 API.
Appendix B

Geth, Mist, Solc, Remix, Truffle, ...
Geth

Ethereum client

Geth

console

node

Web3 I/F

EVM

P2P

Web3 API

Ethereum network

References: [C1]
Mist

Ethereum network (Geth, Parity, ...)

Ethereum client

private key

Web3 API

References: [C3]
Solc

Ethereum network

Ethereum node
(Geth, Parity, ...)

Web3 API

EVM code

Solc
(Solidity compiler)

Solidity source

References: [C2]
References
References

[E1] Ethereum Yellow Paper
ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

[E2] Glossary
https://github.com/ethereum/wiki/wiki/Glossary

A Next-Generation Smart Contract and Decentralized Application Platform
https://github.com/ethereum/wiki/wiki/White-Paper

[E4] Design Rationale

[E5] Ethereum Development Tutorial

[E6] Ethereum Introduction
https://github.com/ethereum/wiki/wiki/Ethereum-introduction

[E7] Solidity Documentation
https://solidity.readthedocs.io/en/develop/

[E8] Web3 JavaScript app API for 0.2x.x
https://github.com/ethereum/wiki/wiki/JavaScript-API

[E9] ethereum/wiki Subtleties
https://github.com/ethereum/wiki/wiki/Subtleties#exceptional-conditions
References

[W1] Awesome Ethereum Virtual Machine
https://github.com/pirapira/awesome-ethereum-virtual-machine

[W2] Diving Into The Ethereum VM
https://blog.qtum.org/diving-into-the-ethereum-vm-6e8d5d2f3c30

[W3] Stack Exchange: Ethereum block architecture
https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture/6413

[W4] Porosity
References

[C1] Go Ethereum
https://github.com/ethereum/go-ethereum

[C2] Solc (Solidity compiler)
https://github.com/ethereum/solidity

[C3] Mist (Ethereum Wallet)
https://github.com/ethereum/mist

[C4] MetaMask
https://github.com/MetaMask/metamask-extension

[C5] Remix
https://github.com/ethereum/browser-solidity

[C6] Truffle
https://github.com/trufflesuite/truffle