EVM

From Solidity to bytes code, memory and storage

Peter Robinson & David Hyland-Wood

10 June 2020

PEGASYS

CONSENSYS

Acknowledgements

Thank you to the following people for reviewing these slides:

- Ben Edgington
- Nicolas Liochon
- Meredith Baxter
- Akua Nti
- Joseph Chow

PEGASYS

CONSENSYS

Agenda

- Introduction
- 以太坊交易
- 以太坊黄皮书中的操作码
- 堆栈、内存、存储、代码、调用数据和日志
- 合约部署、构造函数和初始化代码片段
- 简单合约中的函数调用
- 可支付的回退函数
- 存储
- 内存
- CodeCopy和ExtCodeCopy：使用合约作为静态存储
- Aux Data / Solidity元数据

PEGASYS

CONSENSYS

Examples (all Apache 2 Licensed)

所有示例代码都可以在这里找到：

- https://github.com/ConsenSys/EthEngGroupSolidityExamples/tree/master/evm

字节码输出是使用以下方式创建的:

- ByteCodeDump in this repo: https://github.com/PegaSysEng/codewitness

堆栈分析是在以下的帮助下创建的:

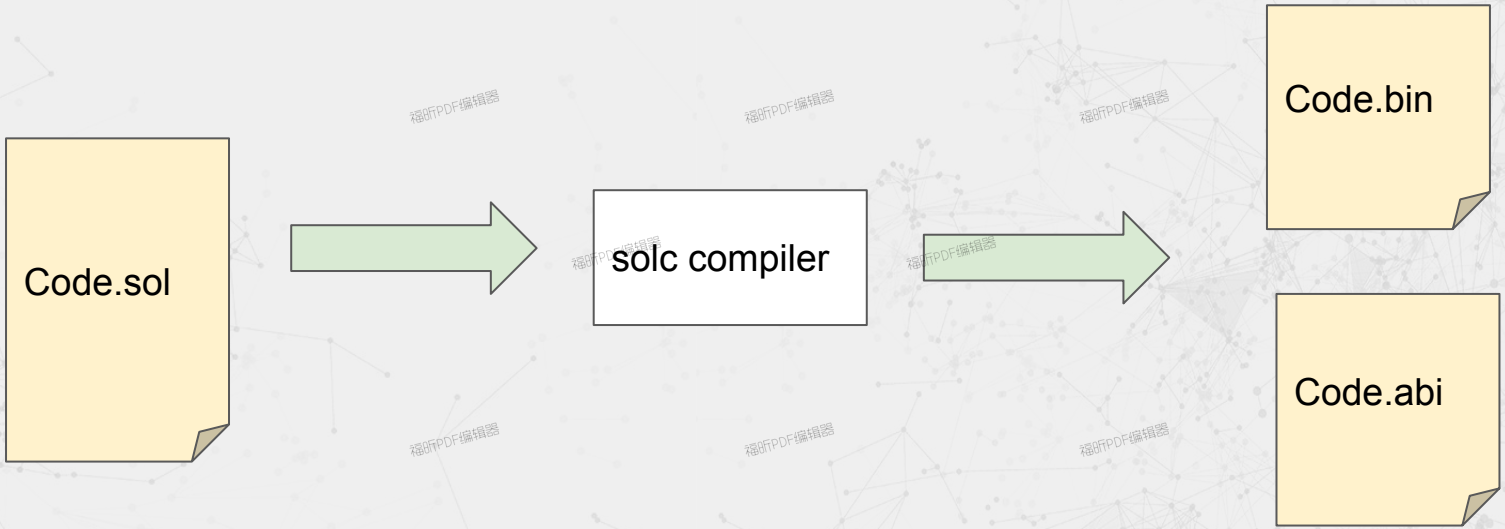
- FunctionIdAnalysis in this repo: https://github.com/PegaSysEng/codewitness

PEGASYS

CONSENSYS

Introduction

编译过程



PEGASYS

Solidity: https://solidity.readthedocs.io/en/v0.6.9/index.html

CONSENSYS

Compilation Process

删除标识 | 福昕编辑器

Solidity只是EVM语言之一

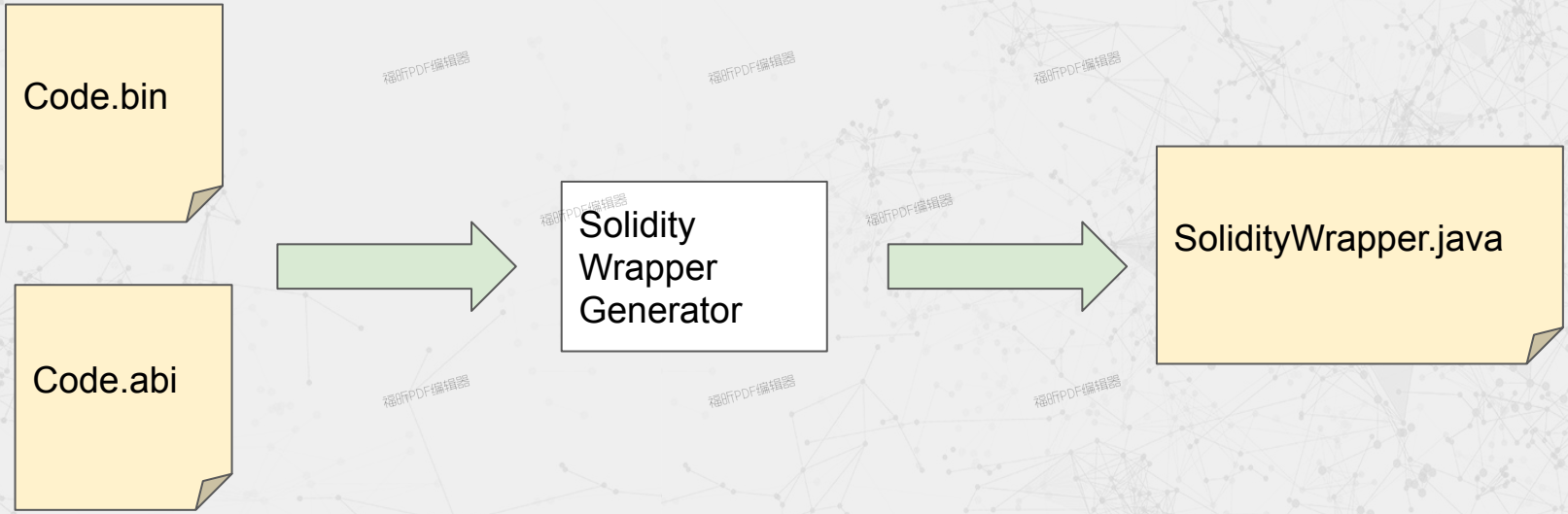
对于Vyper:
Code.vy是使用vyper编译器进行编译的。

PEGASYS

Vyper: https://vyper.readthedocs.io/en/v0.1.0-beta.17/index.html

CONSENSYS

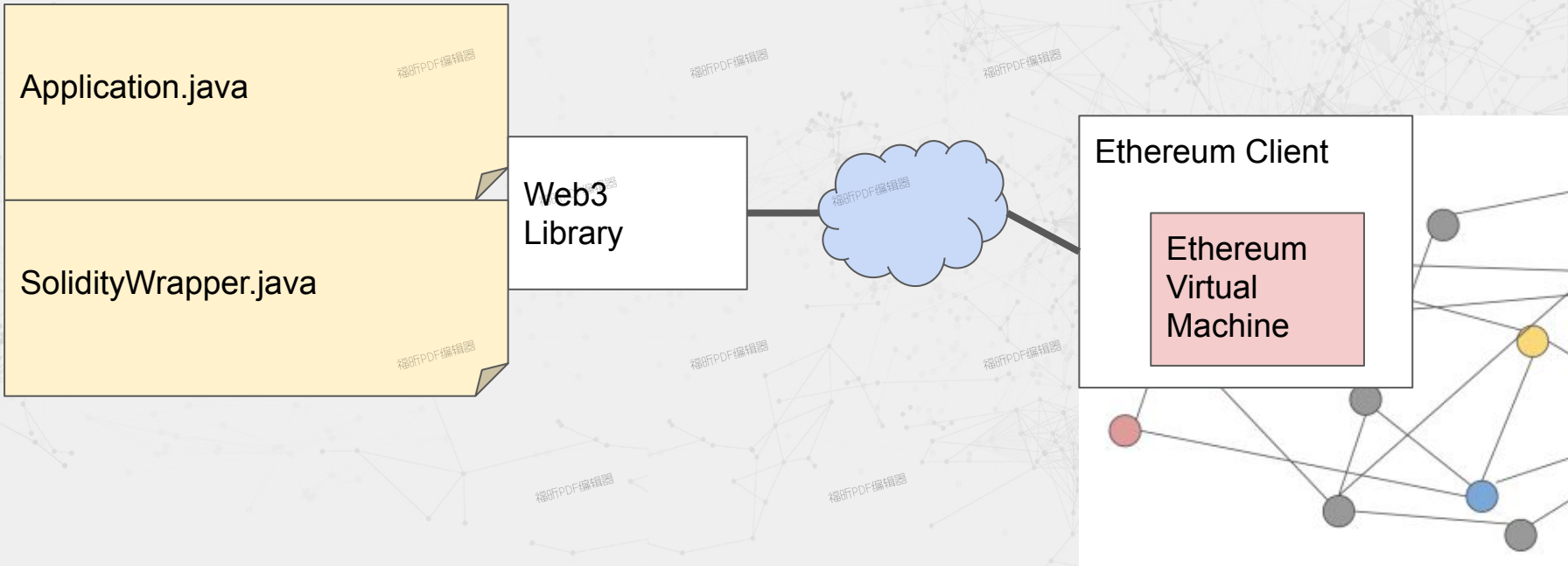
应用程序创建过程



PEGASYS

CONSENSYS

部署架构

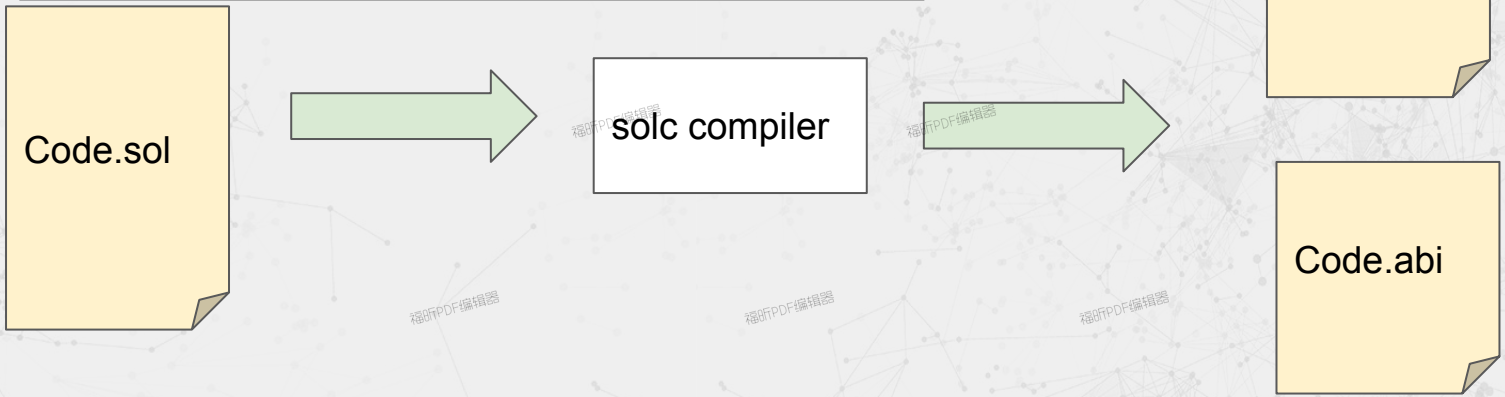


PEGASYS

CONSENSYS

Co

这次讲座将讨论字节码以及它与以太坊虚拟机（EVM）的交互方式。



PEGASYS

CONSENSYS

Ethereum Transactions

交易字段

- Nonce: 该账户的交易编号，从0开始计数。
- Gas Price: 每单位燃料所支付的以太币价格（以Wei为单位）。
- Gas Limit: 该交易可以使用的最大燃料数量。
- To: 交易发送到的地址。
- Value: 要发送的以太币数量（以Wei为单位）。
- Data: 发送到目标地址的数据。
- v、r、s: 交易签名的组成部分。

PEGASYS

CONSENSYS

交易字段：转账

- **Nonce**: 该账户的交易编号，从0开始计数，用于确保交易的唯一性。
- **Gas Price**: 以Wei为单位的价格，用于支付每单位燃料的费用。
- **Gas Limit**:该交易可以使用的最大燃料数量，通常为21000。
- **To**: 要发送以太币的地址。
- **Value**: 要发送的以太币数量，以Wei为单位。
- **Data**: 空字段，没有要发送的附加数据。
- **v, r, s**: 交易签名的组成部分，用于验证交易的合法性。

PEGASYS

CONSENSYS

Transaction Fields: Value tra

- **Nonce**: the transaction number for this a
- **Gas Price**: the price in Wei that this
- **Gas Limit**: 21000
- **To**: the address to send th ther to.
- **Value**: the amount o Wei to send.
- **Data**: Empty
- **v, r, s**: Components of the transaction signature.

实际上，并不一定要为空。例如，如果您只是想在区块链上记录一个承诺，您可以向自己发送一个值为零的转账，同时在此处添加数据。

… 但是….
交易不是世界状态的一部分。

PEGASYS

CONSENSYS

交易字段：合约部署

- **Nonce**: the transaction number for this account, starting with 0.
- **Gas Price**: the price in Wei that this transaction will pay for gas.
- **Gas Limit**: the maximum amount of gas that this transaction can use.
- **To**: 空
- **Value**: the amount of Wei to send.
- **Data**: 用于设置合约状态和部署合约的初始化代码。
- **v, r, s**: Components of the transaction signature.

PEGASYS

CONSENSYS

交易字段：函数调用

- **Nonce**: the transaction number for this account, starting with 0.
- **Gas Price**: the price in Wei that this transaction will pay for gas.
- **Gas Limit**: the maximum amount of gas that this transaction can use.
- **To**: the address the transaction is sent to.
- **Value**: the amount of Wei to send.
- **Data**: EVM中的代码处理并执行的数据。
- **v, r, s**: Components of the transaction signature.

PEGASYS

CONSENSYS

Transaction Fields: Function

- **Nonce**: the transaction number for this account, starting with 0.
- **Gas Price**: the price in Wei that this transaction will pay for gas.
- **Gas Limit**: the maximum amount of gas
- **To**: the address the transaction is sent to.
- **Value**: the amount of Wei to send.
- **Data**:要调用的函数和参数。
- **v, r, s**: Components of the transaction signature.

对于符合应用程序二进制接口（Application Binary Interface）的Solidity合约和其他合约

REF: https://solidity.readthedocs.io/en/v0.6.9/abi-spec.html?highlight=abi

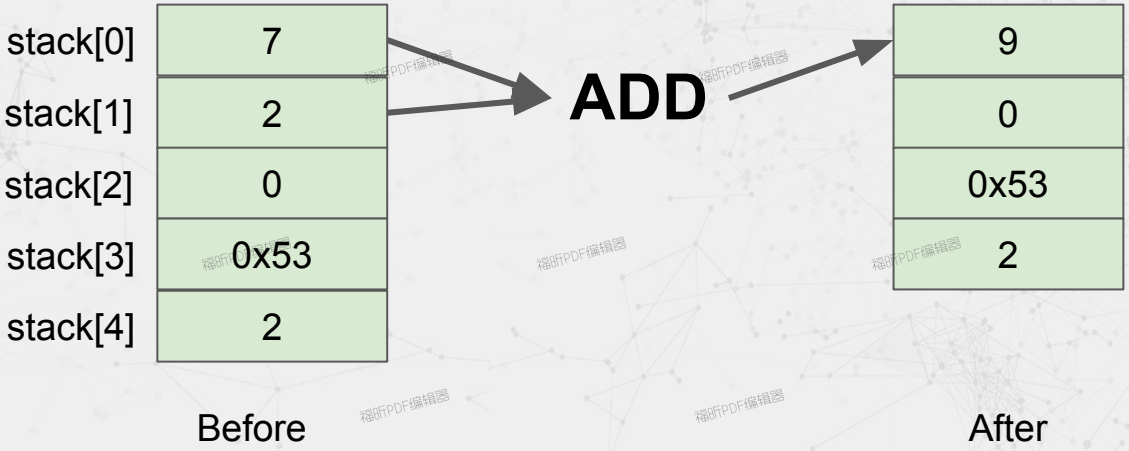
PEGASYS

CONSENSYS

Stack, Memory, Storage, Code, CallData & Logs

Stack, Memory, Storage, Code, CallData, Logs

- EVM（以太坊虚拟机）是一种基于堆栈的处理器。
- EVM操作码从堆栈中弹出信息并将信息推入堆栈。



PEGASYS

CONSENSYS

Stack, Memory, Storage, Code, CallData, Logs

EVM可以在六个位置访问和存储信息：

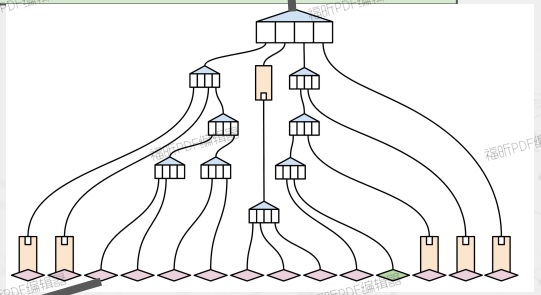
- Stack：EVM操作码从栈中弹出信息并将数据推送到栈中。
- CallData：交易的数据字段。这些是调用的参数。
- Memory：在交易期间可访问的信息存储区域。
- Storage：持久化数据存储区域。
- Code：执行代码和静态数据存储。
- Logs：只写日志器/事件输出。

PEGASYS

CONSENSYS

Block Header
Parent hash, ommers, block number,
time stamp, difficulty, nonce,
transaction root, receipts root...

State Root

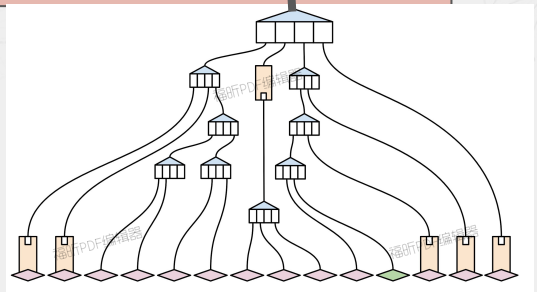


所有账户的MPT树，其中key是160位的账户编号。

Account State

nonce, balance, code hash,

Storage Root



所有账户内存储值的MPT树，其中key是256位的数字。

PEGASYS

CONSENSYS

OpCodes in the Ethereum Yellow Paper

https://github.com/ethereum/yellowpaper

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER
BYZANTIUM VERSION 7e819ec - 2019-10-20

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. Introduction

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated through the power of the default, consensus mechanisms, and voluntary respect of the social contract, that it is possible to use the internet to make a decentralised value-transfer system that can be shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographically secure, transaction-based state machine. Follow-up systems such as Namecoin adapted this original "currency application" of the technology into other applications albeit rather simplistic ones.

Ethereum is a project which attempts to build the generalised technology; technology on which all transaction-based state machine concepts may be built. Moreover it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto unexplored compute paradigm in the mainstream: a trustful object messaging compute framework.

1.1. **Driving Factors.** There are many goals of this project; one key goal is to facilitate transactions between consenting individuals who would otherwise have no means to trust one another. This may be due to geographical separation, interfacing difficulty, or perhaps the incompatibility, incompetence, unwillingness, expense, uncertainty, inconvenience, or corruption of existing legal systems. By specifying a state-change system through a rich and unam-

is often lacking, and plain old prejudices are difficult to shake.

Overall, we wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

1.2. **Previous Work.** Buterin [2013a] first proposed the kernel of this work in late November, 2013. Though now evolved in many ways, the key functionality of a blockchain with a Turing-complete language and an effectively unlimited inter-transaction storage capability remains unchanged.

Dwork and Naor [1992] provided the first work into the usage of a cryptographic proof of computational expenditure ("proof-of-work") as a means of transmitting a value signal over the Internet. The value-signal was utilised here as a spam deterrence mechanism rather than any kind of currency, but critically demonstrated the potential for a basic data channel to carry a *strong economic signal*, allowing a receiver to make a physical assertion without having to rely upon *trust*. Back [2002] later produced a system in a similar vein.

The first example of utilising the proof-of-work as a strong economic signal to secure a currency was by Vishnumurthy et al. [2003]. In this instance, the token was used to keep peer-to-peer file trading in check, providing "consumers" with the ability to make micro-payments to "suppliers" for their services. The security model afforded by the proof-of-work was augmented with digital signatures and a ledger in order to ensure that the historical record

PEGASYS

CONSENSYS

EVM OpCodes start on pg 28:

0s: Stop and Arithmetic Operations

All arithmetic is modulo 2^{256} unless otherwise noted. The zero-th power of zero 0^0 is defined to be one.

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\mu'_s[0] \equiv \mu_s[0] + \mu_s[1]$
0x02	MUL	2	1	Multiplication operation. $\mu'_s[0] \equiv \mu_s[0] \times \mu_s[1]$
0x03	SUB	2	1	Subtraction operation. $\mu'_s[0] \equiv \mu_s[0] - \mu_s[1]$
0x04	DIV	2	1	Integer division operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \lfloor \mu_s[0] \div \mu_s[1] \rfloor & \text{otherwise} \end{cases}$
0x05	SDIV	2	1	Signed integer division operation (truncated). $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ -2^{255} & \text{if } \mu_s[0] = -2^{255} \wedge \mu_s[1] = -1 \\ \mathbf{sgn}(\mu_s[0] \div \mu_s[1])\lfloor \lvert \mu_s[0] \div \mu_s[1] \rvert \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers. Note the overflow semantic when -2^{255} is negated.
0x06	MOD	2	1	Modulo remainder operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \mu_s[0] \bmod \mu_s[1] & \text{otherwise} \end{cases}$
0x07	SMOD	2	1	Signed modulo remainder operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \mathbf{sgn}(\mu_s[0])(\lvert \mu_s[0] \rvert \bmod \lvert \mu_s[1] \rvert) & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers.
0x08	ADDMOD	3	1	Modulo addition operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[2] = 0 \\ (\mu_s[0] + \mu_s[1]) \bmod \mu_s[2] & \text{otherwise} \end{cases}$ All intermediate calculations of this operation are not subject to the 2^{256} modulo.
0x09	MULMOD	3	1	Modulo multiplication operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[2] = 0 \\ (\mu_s[0] \times \mu_s[1]) \bmod \mu_s[2] & \text{otherwise} \end{cases}$ All intermediate calculations of this operation are not subject to the 2^{256} modulo.
0x0a	EXP	2	1	Exponential operation. $\mu'_s[0] \equiv \mu_s[0]^{\mu_s[1]}$
0x0b	SIGNEXTEND	2	1	Extend length of two's complement signed integer. $\forall i \in [0..255] : \mu'_s[0]_i \equiv \begin{cases} \mu_s[1]_i & \text{if } i \leqslant t \quad \text{where } t = 256 - 8(\mu_s[0] + 1) \\ \mu_s[1]_i & \text{otherwise} \end{cases}$ $\mu_s[x]_i$ gives the ith bit (counting from zero) of $\mu_s[x]$

PEGASYS

CONSENSYS

0s: Stop and Arithmetic Operations

All arithmetic is modulo 2^{256} unless otherwise noted. The zero-th power of zero 0^0 is defined to be one.

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] + \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x02	MUL	2	1	Multiplication operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] \times \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x03	SUB	2	1	Subtraction operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] - \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x04	DIV	2	1	Integer division operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \begin{cases} 0 & \text{if} \quad \boldsymbol{\mu}_{\mathbf{s}}[1] = 0 \\ \lfloor \boldsymbol{\mu}_{\mathbf{s}}[0] \div \boldsymbol{\mu}_{\mathbf{s}}[1] \rfloor & \text{otherwise} \end{cases}$
0x05	SDIV	2	1	Signed integer division operation (truncated).

Contract Deployment, Constructors & Init Code Fragments

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

PEGASYS

CONSENSYS

Example

```
pragma solidity >=0.4.23;

contract Simple {
   uint256 public val1;
   uint256 public val2;

   constructor() public {
     val2 = 3;
   }

   function set(uint256 _param) external {
     val1 = _param;
   }
}
```

The init function includes code to deploy the contract plus the constructor to set-up the contract state.

PEGASYS

CONSENSYS

Compiling

```
solc Simple.sol --bin --abi --optimize -o . --overwrite
```

Simple.bin

```
6080604052348015610010576000 80fd5b506003600155 60c1806100246000396000f3fe6080604052348015600f57600080fd5b506004361060 3c5760003560e01c806360fe47b1146041578063 95cacbe014605d578063c82fdf36146075575b600080fd5b605b60048036036020811015605557600080fd5b5035607b565b005b606360805565b6040805191825251908190036020019 0f35b60636086565b600055565b600154815565b6000548156fea265627a7a723058204e00129b67b55015b0de73c3167fb19ae30a4bb9b293318b7fb6c40bced080b864736f6c634300050a0032
```

PEGASYS

CONSENSYS

Simple.bin

初始化代码片段

```
608060405234801561001057600080fd5b506003600155560c1806100246000396000f3fe608060405234801561000f57600080fd5b506004361060
3c5760003560e01c806360fe47b1146041578063cacbe014605d578063c82fdf3614607557
5b600080fd5b605b60048036036020811015604f57600080fd5b5035607b565b005b606360
80565b6040805191825251908190036020019
0f35b60636086565b600055565b60015481565b6000548156fea265627a7a723058204e00129b67b55015b0de73c3167fb19ae30a4bb9b293318b7fb6c40bced080b864736f6c634300050a0032
```

要部署到区块链的代码

PEGASYS

CONSENSYS

Constructors

合约通过交易进行部署，在其中：

- 目标地址未指定。
- 数据是初始化代码片段，其中包括合约的二进制代码。这是编译器生成的*.bin文件中的输出。

Note:

- 初始化函数不会存储在区块链上。
- 交易的数据被视为合约部署的代码。

PEGASYS

CONSENSYS

Byte Code and OpCodes

6080604052348015610010576000 80fd5b506003600155 60c1806100246000396000f3fe6080604052348015600f57600080fd5b506004361060 3c5760003560e01c806360fe47b1146041578063 95cacbe014605d578063c82fdf36146075575b600080fd5b605b6004803603602081101560555760 0080fd5b5035607b565b005b606360805565b6040805191825190819003602001 90f35b60636086565b600055565b600154815565b600054815 65627a7a723058204e00129b67b55015b0de73c3167fb19ae30a4bb9b293318b7 0bced080b864736f6c634300050a0032

PUSH 0x80

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....
```

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....
```

将一个字（word）推入栈中。

栈的最大深度为1024个字。

stack[0]表示栈顶的值，

stack[1]表示栈顶下面的一个值。

PUSH1：将紧随该操作码的字节推入栈中。字的前31个字节填充为零。

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....
```

PUSH1 0x80

Push a new item onto the stack.

stack[0] = 0x80

PEGASYS

CONSENSYS

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

PUSH1 0x40

stack[1] = stack[0]
stack[0] = 0x40

Stack now looks like:
stack[0]: 0x40
stack[1]: 0x80

PEGASYS

CONSENSYS

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

MSTORE：在内存中存储一个字（word）。

从栈中弹出两个值。stack[0]表示要写入的位置，stack[1]表示要写入的值。

栈现在为空。

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....
```

设置自由内存指针

Memory [0x40] = 0x80

这意味着Solidity代码可以安全地从0x80开始分配内存。如果代码分配了内存，指针应该被更新。

注意：地址是以字节为单位的，但MSTORE存储32字节的字（word）。

PEGASYS

CONSENSYS

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

CALLVALUE

将发送交易时携带的Wei数量推入栈中，即VALUE交易字段。

栈现在如下所示：

stack[0]: 与交易一起发送的值

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

DUP1

复制栈顶的值。栈现在如下所示：

stack[0]: 与交易一起发送的值

stack[1]: 与交易一起发送的值

注意：DUP2将stack[1]的副本推入栈中，DUP3将stack[2]的副本推入栈中。

PEGASYS

CONSENSYS

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

ISZERO

从栈中弹出一个字（word）。如果该字为零，则将1推入栈中，否则将0推入栈中。

栈现在如下所示：

stack[0]: 如果value==0，则为1；如果value!=0，则为0

stack[1]: 与交易一起发送的值

PEGASYS

CONSENSYS

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

PUSH2 0x0010

将操作码后面的两个字节推入栈中。

栈现在如下所示：

stack[0]: 0x10

stack[1]: 如果value==0，则为1；如果value!=0，则为0

stack[2]: 与交易一起发送的值

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....
```

JUMPI

如果 stack[1] 不为零，则将程序计数器（PC）设置为 stack[0]。从栈中弹出两个值。

栈现在的情况如下：

stack[0]: 通过事务发送的值

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....
```

PUSH1 0x0, DUP1

将 0x0 压入栈中两次。

栈现在的情况如下：

stack[0]: 0

stack[1]: 0

stack[2]: 通过事务发送的值

PEGASYS

CONSENSYS

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

REVERT

停止执行并指示发生了回滚。使用 stack[0] 作为内存位置，stack[1] 作为回滚原因的长度。

执行之前，栈的情况如下：

stack[0]: 0

stack[1]: 0

stack[2]: 通过事务发送的值

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....
```

如果构造函数（不可支付）在合约部署交易中接收到以太币，则执行 REVERT。

PEGASYS

CONSENSYS

ByteCode

PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH2 0x0010
PC: 0xb, opcode: JUMPI
PC: 0xc, opcode: PUSH1 0x00
PC: 0xe, opcode: DUP1
PC: 0xf, opcode: REVERT
PC: 0x10, opcode: JUMPDEST
....

JUMPDEST

如果跳转到程序计数器 0x10，则会到达此处。

有效的跳转目标由 JUMPDEST 操作码表示。

栈现在的情况如下：

stack[0]: 通过事务发送的值

PEGASYS

CONSENSYS

ByteCode

PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....

POP

从栈中弹出栈顶的值。

栈现在为空。

PEGASYS

CONSENSYS

Recall Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

注意：构造函数将常量 3 存储到第二个存储变量 val2 中。

PEGASYS

CONSENSYS

ByteCode

PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....

SSTORE：将一个字写入storage。

从栈中弹出两个值。stack[0] 是要写入的位置，stack[1] 是要写入的值。

Storage[1] = 3

栈现在为空。

PEGASYS

CONSENSYS

ByteCode

PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....

为 CODECOPY 操作码和
RETURN 操作码做好栈的准备。

栈现在的情况如下：

stack[0]: 0

stack[1]: 0x24

stack[2]: 0xC1

stack[3]: 0xC1

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....
```

CODECOPY

从正在执行的代码中复制到内存。
stack[0] 是要写入的内存偏移量，
stack[1] 是要读取的代码偏移量，
stack[2] 是要复制的字节数。

栈的情况如下：
stack[0]: 0
stack[1]: 0x24
stack[2]: 0xC1
stack[3]: 0xC1

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....
```

CODECOPY

从正在执行的代码中复制到内存。

stack[0] 是要写入的内存偏移量，
stack[1] 是要读取的代码偏移量，
stack[2] 是要复制的字节数。
栈现在的情况如下：
stack[3]: 0xC1

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....
```

将将在区块链上的合约代码复制到内存中。
代码的起始偏移量为 0x24，长度为 0xC1。
复制到内存偏移量 0x00。

PEGASYS

CONSENSYS

ByteCode

PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....

RETURN
结束执行，返回结果并指示成功执行。
stack[0] 是结果的起始偏移量（在内存中）
。
stack[1] 是结果的结束偏移量（在内存中）
。

栈的情况如下：
stack[0]: 0
stack[1]: 0xC1

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....
```

对于合约部署：

将返回结果存储到部署的合约地址中。

在示例中，位于程序计数器偏移量 0x24 处的 193 字节（0xc1）代码被复制到内存 [0x0...0xc1]，并且这段代码将在该合约地址上找到。

PEGASYS

CONSENSYS

ByteCode

PC: 0x10, opcode: JUMPDEST
PC: 0x11, opcode: POP
PC: 0x12, opcode: PUSH1 0x03
PC: 0x14, opcode: PUSH1 0x01
PC: 0x16, opcode: SSTORE
PC: 0x17, opcode: PUSH1 0xc1
PC: 0x19, opcode: DUP1
PC: 0x1a, opcode: PUSH2 0x0024
PC: 0x1d, opcode: PUSH1 0x00
PC: 0x1f, opcode: CODECOPY
PC: 0x20, opcode: PUSH1 0x00
PC: 0x22, opcode: RETURN
PC: 0x23, opcode: INVALID
PC: 0x24, opcode: PUSH1 0x80
....

INVALID

无效操作标志着初始化代码的结束。

PEGASYS

CONSENSYS

初始化代码总结

- 设置自由存储指针（未使用）。
- 如果交易中有以太币发送，则引发 REVERT。
- 设置存储位置，并赋予初始非零值。
- 返回合约代码的指针和要存储在区块链上的代码长度。

PEGASYS

CONSENSYS

Function Calls

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

PEGASYS

CONSENSYS

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

公共存储变量会自动创建“getter”视图调用。

PEGASYS

CONSENSYS

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

这意味着在这份合约中有三个函数，而不仅仅是一个名为set(uint256)的函数。

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH1 0x0f
PC: 0xa, opcode: JUMPI
PC: 0xb, opcode: PUSH1 0x00
PC: 0xd, opcode: DUP1
PC: 0xe, opcode: REVERT
PC: 0xf, opcode: JUMPDEST

....
```

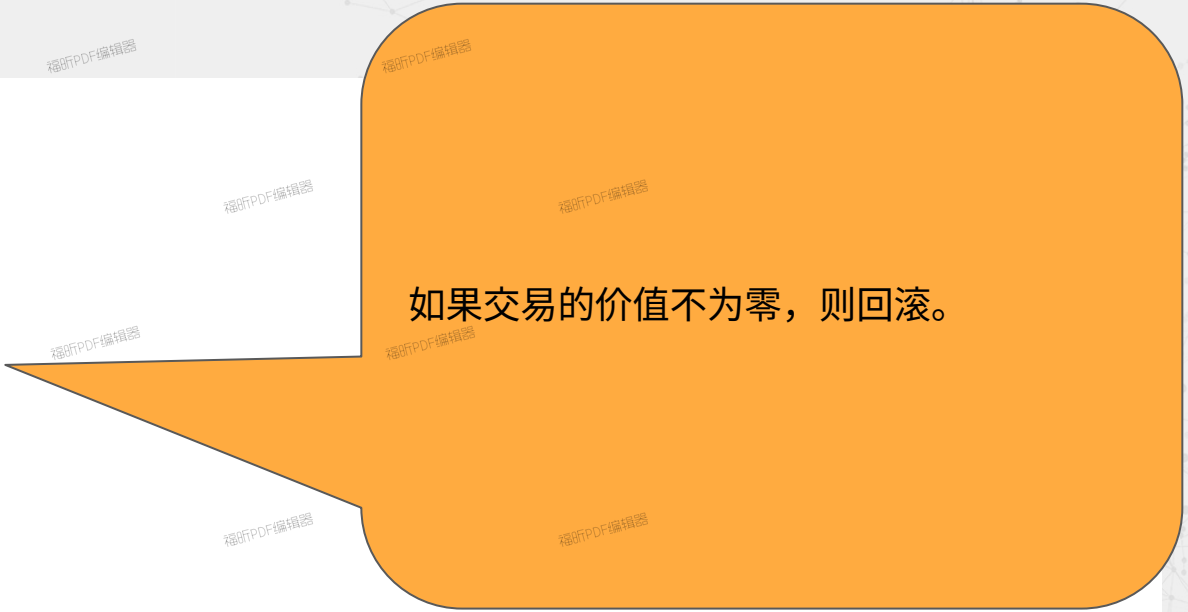
设置自由内存指针

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x0, opcode: PUSH1 0x80
PC: 0x2, opcode: PUSH1 0x40
PC: 0x4, opcode: MSTORE
PC: 0x5, opcode: CALLVALUE
PC: 0x6, opcode: DUP1
PC: 0x7, opcode: ISZERO
PC: 0x8, opcode: PUSH1 0x0f
PC: 0xa, opcode: JUMPI
PC: 0xb, opcode: PUSH1 0x00
PC: 0xd, opcode: DUP1
PC: 0xe, opcode: REVERT
PC: 0xf, opcode: JUMPDEST

....
```



PEGASYS

CONSENSYS

ByteCode

PC: 0xf, opcode: JUMPDEST
PC: 0x10, opcode: POP
PC: 0x11, opcode: PUSH1 0x04
PC: 0x13, opcode: CALLDATASIZE
PC: 0x14, opcode: LT
PC: 0x15, opcode: PUSH1 0x3c
PC: 0x17, opcode: JUMPI
PC: 0x18, opcode: PUSH1 0x00

....

从堆栈中移除VALUE。

PEGASYS

CONSENSYS

ByteCode

PC: 0xf, opcode: JUMPDEST
PC: 0x10, opcode: POP
PC: 0x11, opcode: PUSH1 0x04
PC: 0x13, opcode: CALLDATASIZE
PC: 0x14, opcode: LT
PC: 0x15, opcode: PUSH1 0x3c
PC: 0x17, opcode: JUMPI
PC: 0x18, opcode: PUSH1 0x00

....

CALLDATASIZE

将交易数据字段的大小推送到堆栈上。

PEGASYS

CONSENSYS

ByteCode

PC: 0xf, opcode: JUMPDEST
PC: 0x10, opcode: POP
PC: 0x11, opcode: PUSH1 0x04
PC: 0x13, opcode: CALLDATASIZE
PC: 0x14, opcode: LT
PC: 0x15, opcode: PUSH1 0x3c
PC: 0x17, opcode: JUMPI
PC: 0x18, opcode: PUSH1 0x00

....

LT: Less Than

If stack[0] < stack[1]:
  pop stack[0] and stack[1] off the stack
  push 1 onto the stack
otherwise
  push 0 onto the stack.

PEGASYS

CONSENSYS

ByteCode

PC: 0xf, opcode: JUMPDEST
PC: 0x10, opcode: POP
PC: 0x11, opcode: PUSH1 0x04
PC: 0x13, opcode: CALLDATASIZE
PC: 0x14, opcode: LT
PC: 0x15, opcode: PUSH1 0x3c
PC: 0x17, opcode: JUMPI
PC: 0x18, opcode: PUSH1 0x00
....
PC: 0x3c, opcode: JUMPDEST
PC: 0x3d, opcode: PUSH1 0x00
PC: 0x3f, opcode: DUP1
PC: 0x40, opcode: REVERT

如果交易数据字段长度小于4个字节，则回滚。

PEGASYS

CONSENSYS

Function Selector

“函数调用的调用数据的前四个字节指定要调用的函数。它是函数签名的Keccak-256哈希的前四个字节（从左到右，高位在前，大端序）。签名被定义为基本原型的规范表达，不包含数据位置说明符，即带有参数类型括在括号内的函数名称。参数类型由一个逗号分隔 - 不使用空格。”

https://solidity.readthedocs.io/en/v0.6.6/abi-spec.html?highlight=abi

ByteCode

```
PC: 0xf, opcode: JUMPDEST
PC: 0x10, opcode: POP
PC: 0x11, opcode: PUSH1 0x04
PC: 0x13, opcode: CALLDATASIZE
PC: 0x14, opcode: LT
PC: 0x15, opcode: PUSH1 0x3c
PC: 0x17, opcode: JUMPI
PC: 0x18, opcode: PUSH1 0x00
....
PC: 0x3c, opcode: JUMPDEST
PC: 0x3d, opcode: PUSH1 0x00
PC: 0x3f, opcode: DUP1
PC: 0x40, opcode: REVERT
```

如果交易的数据字段长度小于4个字节，
即函数选择器的大小，则回滚。

PEGASYS

CONSENSYS

"Jump table" for functions

PEGASYS

CONSENSYS

ByteCode

PC: 0x18, opcode: PUSH1 0x00
PC: 0x1a, opcode: CALLDATALOAD
PC: 0x1b, opcode: PUSH1 0xe0
PC: 0x1d, opcode: SHR
PC: 0x1e, opcode: DUP1
PC: 0x1f, opcode: PUSH4 0x60fe47b1
PC: 0x24, opcode: EQ
PC: 0x25, opcode: PUSH1 0x41
PC: 0x27, opcode: JUMPI
….

CALLDATALOAD

将从stack[0]偏移开始的32个字节（CALLDATA）推送到堆栈上。

PEGASYS

CONSENSYS

ByteCode

PC: 0x18, opcode: PUSH1 0x00
PC: 0x1a, opcode: CALLDATALOAD
PC: 0x1b, opcode: PUSH1 0xe0
PC: 0x1d, opcode: SHR
PC: 0x1e, opcode: DUP1
PC: 0x1f, opcode: PUSH4 0x60fe47b1
PC: 0x24, opcode: EQ
PC: 0x25, opcode: PUSH1 0x41
PC: 0x27, opcode: JUMPI
….

CALLDATALOAD

将从偏移量0x00开始的32个字节推送到堆栈上。

PEGASYS

CONSENSYS

ByteCode

PC: 0x18, opcode: PUSH1 0x00
PC: 0x1a, opcode: CALLDATALOAD
PC: 0x1b, opcode: PUSH1 0xe0
PC: 0x1d, opcode: SHR
PC: 0x1e, opcode: DUP1
PC: 0x1f, opcode: PUSH4 0x60fe47b1
PC: 0x24, opcode: EQ
PC: 0x25, opcode: PUSH1 0x41
PC: 0x27, opcode: JUMPI
….

SHR

Shift stack[1] to the right stack[0] times, pop stack[0] off the stack.

PEGASYS

CONSENSYS

ByteCode

PC: 0x18, opcode: PUSH1 0x00
PC: 0x1a, opcode: CALLDATALOAD
PC: 0x1b, opcode: PUSH1 0xe0
PC: 0x1d, opcode: SHR
PC: 0x1e, opcode: DUP1
PC: 0x1f, opcode: PUSH4 0x60fe47b1
PC: 0x24, opcode: EQ
PC: 0x25, opcode: PUSH1 0x41
PC: 0x27, opcode: JUMPI
….

Shift the top 32 bits to the bottom 32 bits of the word.

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x18, opcode: PUSH1 0x00
PC: 0x1a, opcode: CALLDATALOAD
PC: 0x1b, opcode: PUSH1 0xe0
PC: 0x1d, opcode: SHR
PC: 0x1e, opcode: DUP1
PC: 0x1f, opcode: PUSH4 0x60fe47b1
PC: 0x24, opcode: EQ
PC: 0x25, opcode: PUSH1 0x41
PC: 0x27, opcode: JUMPI
….
```

Load the Function Selector from the transaction data field.

PEGASYS

CONSENSYS

ByteCode

PC: 0x18, opcode: PUSH1 0x00
PC: 0x1a, opcode: CALLDATALOAD
PC: 0x1b, opcode: PUSH1 0xe0
PC: 0x1d, opcode: SHR
PC: 0x1e, opcode: DUP1
PC: 0x1f, opcode: PUSH4 0x60fe47b1
PC: 0x24, opcode: EQ
PC: 0x25, opcode: PUSH1 0x41
PC: 0x27, opcode: JUMPI
….

Make a copy of stack[0], the Function Selector from the transaction data field.

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x18, opcode: PUSH1 0x00
PC: 0x1a, opcode: CALLDATALOAD
PC: 0x1b, opcode: PUSH1 0xe0
PC: 0x1d, opcode: SHR
PC: 0x1e, opcode: DUP1
PC: 0x1f, opcode: PUSH4 0x60fe47b1
PC: 0x24, opcode: EQ
PC: 0x25, opcode: PUSH1 0x41
PC: 0x27, opcode: JUMPI
....
```

Jump to 0x41 if the Function Selector from the transaction data field == 0x60fe47b1

PEGASYS

CONSENSYS

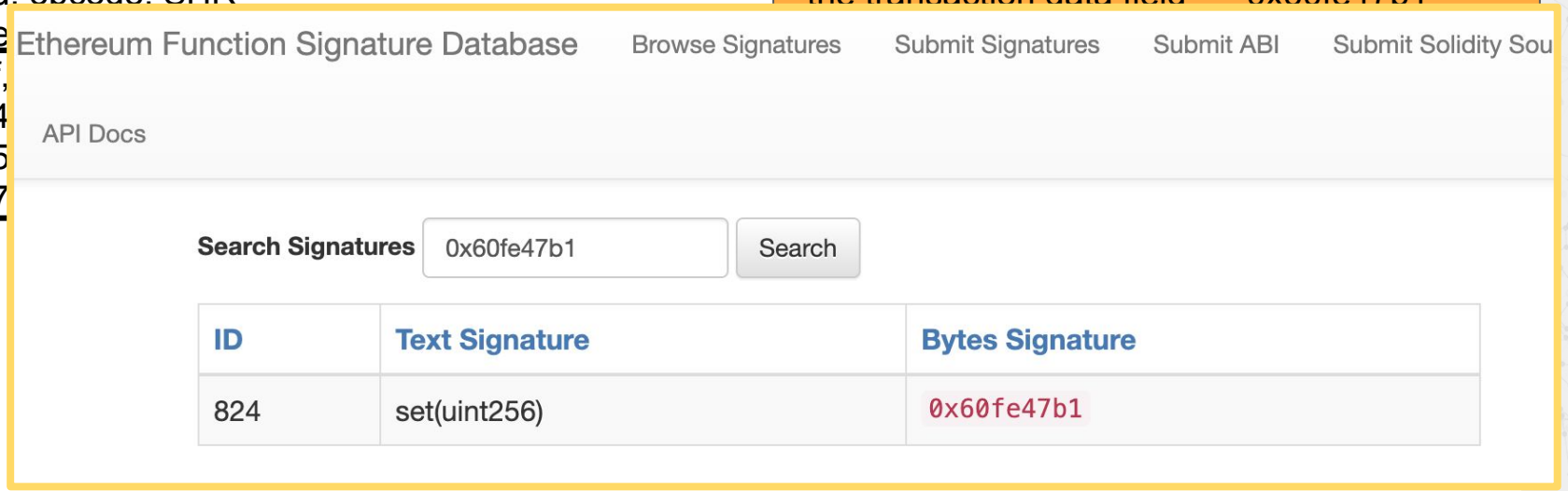
ByteCode

PC: 0x18, opcode: PUSH1 0x00

https://www.4byte.directory/signatures/?bytes4_signature=0x60fe47b1

nction Selector from
the transaction data field == 0x60fe47b1

PC: 0x1d, opcode: SHR
PC: 0x1e
PC: 0x1f,
PC: 0x24
PC: 0x25
PC: 0x27
....



PEGASYS

CONSENSYS

ByteCode

```
PC: 0x28, opcode: DUP1
PC: 0x29, opcode: PUSH4 0x95cacbe0
PC: 0x2e, opcode: EQ
PC: 0x2f, opcode: PUSH1 0x5d
PC: 0x31, opcode: JUMPI
PC: 0x32, opcode: DUP1
PC: 0x33, opcode: PUSH4 0xc82fdf36
PC: 0x38, opcode: EQ
PC: 0x39, opcode: PUSH1 0x75
PC: 0x3b, opcode: JUMPI
PC: 0x3c, opcode: JUMPDEST
PC: 0x3d, opcode: PUSH1 0x00
PC: 0x3f, opcode: DUP1
PC: 0x40, opcode: REVERT
....
```

Jump to 0x5d if the Function Selector from the transaction data field == 0x95cacbe0

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x28, opcode: DUP1
PC: 0x29, opcode: PUSH4 0x95cacbe0
PC: 0x2e, opcode: EQ
PC: 0x2f, opcode: PUSH1 0x5d
PC: 0x31, opcode: JUMPI
PC: 0x32, opcode: DUP1
PC: 0x33, opcode: PUSH4 0xc82fdf36
PC: 0x38, opcode: EQ
PC: 0x39, opcode: PUSH1 0x75
PC: 0x3b, opcode: JUMPI
PC: 0x3c, opcode: JUMPDEST
PC: 0x3d, opcode: PUSH1 0x00
PC: 0x3f, opcode: DUP1
PC: 0x40, opcode: REVERT
....
```

Jump to 0x75 if the Function Selector from the transaction data field == 0xc82fdf36

PEGASYS

CONSENSYS

ByteCode

PC: 0x28, opcode: DUP1
PC: 0x29, opcode: PUSH4 0x95cacbe0
PC: 0x2e, opcode: EQ
PC: 0x2f, opcode: PUSH1 0x5d
PC: 0x31, opcode: JUMPI
PC: 0x32, opcode: DUP1
PC: 0x33, opcode: PUSH4 0xc82fdf36
PC: 0x38, opcode: EQ
PC: 0x39, opcode: PUSH1 0x75
PC: 0x3b, opcode: JUMPI
PC: 0x3c, opcode: JUMPDEST
PC: 0x3d, opcode: PUSH1 0x00
PC: 0x3f, opcode: DUP1
PC: 0x40, opcode: REVERT
....

If the Function Selector from the transaction data field doesn't equal any of the functions then Revert.

PEGASYS

CONSENSYS

ByteCode

PC: 0x28, opcode: DUP1
PC: 0x29, opcode: PUSH4 0x95cacbe0
PC: 0x2e, opcode: EQ
PC: 0x2f, opcode: PUSH1 0x5d
PC: 0x31, opcode: JUMPI
PC: 0x32, opcode: DUP1
PC: 0x33, opcode: PUSH4 0xc82fdf36
PC: 0x38, opcode: EQ
PC: 0x39, opcode: PUSH1 0x75
PC: 0x3b, opcode: JUMPI
PC: 0x3c, opcode: JUMPDEST
PC: 0x3d, opcode: PUSH1 0x00
PC: 0x3f, opcode: DUP1
PC: 0x40, opcode: REVERT
....

From Ben Edgington:

有趣的事实是，Solidity按照数值顺序排列其函数选择器。如果您为经常使用的函数选择具有较低数值的名称，它们将首先被检查，并且您可以节省一些（微小的）燃气费用 :-)

然而...一旦您拥有超过四个函数，代码将执行“二分查找”函数选择器。

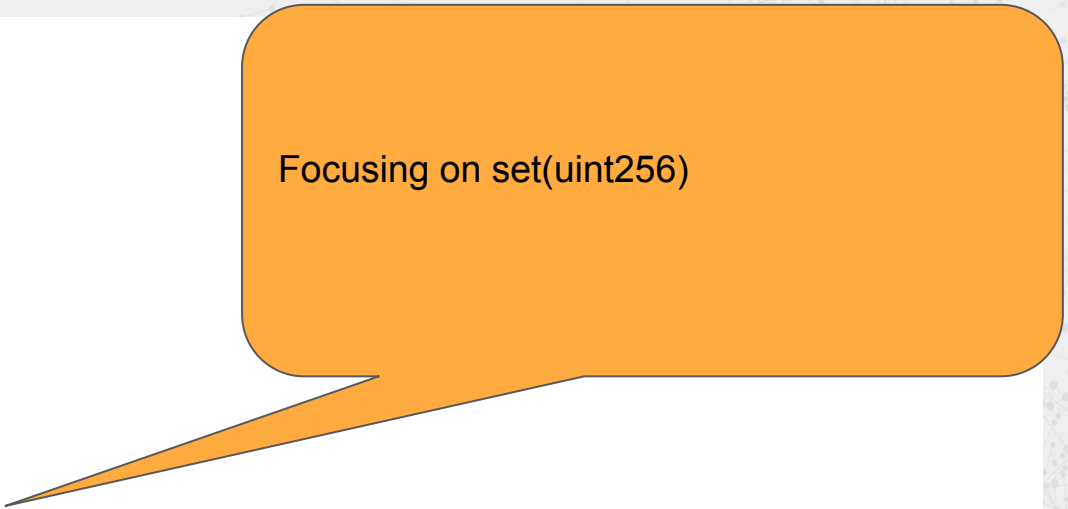
总结一下：如果您想进行这种类型的操作以确保结果符合预期，请查看生成的字节码。

PEGASYS

set(uint256)

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```



PEGASYS

CONSENSYS

ByteCode

```
PC: 0x41, opcode: JUMPDEST
PC: 0x42, opcode: PUSH1 0x5b
PC: 0x44, opcode: PUSH1 0x04
PC: 0x46, opcode: DUP1
PC: 0x47, opcode: CALLDATASIZE
PC: 0x48, opcode: SUB
PC: 0x49, opcode: PUSH1 0x20
PC: 0x4b, opcode: DUP2
PC: 0x4c, opcode: LT
PC: 0x4d, opcode: ISZERO
PC: 0x4e, opcode: PUSH1 0x55
PC: 0x50, opcode: JUMPI
PC: 0x51, opcode: PUSH1 0x00
PC: 0x53, opcode: DUP1
PC: 0x54, opcode: REVERT
...
```

Put a “return address” onto the stack.

This will be used later.

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x41, opcode: JUMPDEST
PC: 0x42, opcode: PUSH1 0x5b
PC: 0x44, opcode: PUSH1 0x04
PC: 0x46, opcode: DUP1
PC: 0x47, opcode: CALLDATASIZE
PC: 0x48, opcode: SUB
PC: 0x49, opcode: PUSH1 0x20
PC: 0x4b, opcode: DUP2
PC: 0x4c, opcode: LT
PC: 0x4d, opcode: ISZERO
PC: 0x4e, opcode: PUSH1 0x55
PC: 0x50, opcode: JUMPI
PC: 0x51, opcode: PUSH1 0x00
PC: 0x53, opcode: DUP1
PC: 0x54, opcode: REVERT
...
```

stack[0] = CALLDATASIZE - 4

That is, the call data size less the length of the Function Selector.

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x41, opcode: JUMPDEST
PC: 0x42, opcode: PUSH1 0x5b
PC: 0x44, opcode: PUSH1 0x04
PC: 0x46, opcode: DUP1
PC: 0x47, opcode: CALLDATASIZE
PC: 0x48, opcode: SUB
PC: 0x49, opcode: PUSH1 0x20
PC: 0x4b, opcode: DUP2
PC: 0x4c, opcode: LT
PC: 0x4d, opcode: ISZERO
PC: 0x4e, opcode: PUSH1 0x55
PC: 0x50, opcode: JUMPI
PC: 0x51, opcode: PUSH1 0x00
PC: 0x53, opcode: DUP1
PC: 0x54, opcode: REVERT
...
```

NOTE: LT followed by ISZERO means NOT(LT) which is the same as Greater Than Or Equal To.

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x41, opcode: JUMPDEST
PC: 0x42, opcode: PUSH1 0x5b
PC: 0x44, opcode: PUSH1 0x04
PC: 0x46, opcode: DUP1
PC: 0x47, opcode: CALLDATASIZE
PC: 0x48, opcode: SUB
PC: 0x49, opcode: PUSH1 0x20
PC: 0x4b, opcode: DUP2
PC: 0x4c, opcode: LT
PC: 0x4d, opcode: ISZERO
PC: 0x4e, opcode: PUSH1 0x55
PC: 0x50, opcode: JUMPI
PC: 0x51, opcode: PUSH1 0x00
PC: 0x53, opcode: DUP1
PC: 0x54, opcode: REVERT
...
```

Revert if the (CALLDATASIZE - 4) < the size of a uint256 (32 bytes = 0x20 bytes)

PEGASYS

CONSENSYS

ByteCode

PC: 0x55, opcode: JUMPDEST
PC: 0x56, opcode: POP
PC: 0x57, opcode: CALLDATALOAD
PC: 0x58, opcode: PUSH1 0x7b
PC: 0x5a, opcode: JUMP
PC: 0x5b, opcode: JUMPDEST
PC: 0x5c, opcode: STOP
…
PC: 0x7b, opcode: JUMPDEST
PC: 0x7c, opcode: PUSH1 0x00
PC: 0x7e, opcode: SSTORE
PC: 0x7f, opcode: JUMP

Stack after the POP is:
stack[0]: 0x04
stack[1]: 0x5b

PEGASYS

CONSENSYS

ByteCode

PC: 0x55, opcode: JUMPDEST
PC: 0x56, opcode: POP
PC: 0x57, opcode: CALLDATALOAD
PC: 0x58, opcode: PUSH1 0x7b
PC: 0x5a, opcode: JUMP
PC: 0x5b, opcode: JUMPDEST
PC: 0x5c, opcode: STOP
…
PC: 0x7b, opcode: JUMPDEST
PC: 0x7c, opcode: PUSH1 0x00
PC: 0x7e, opcode: SSTORE
PC: 0x7f, opcode: JUMP

Push the word at offset 0x04 in the transaction data field onto the stack and jump to 0x7b.

ByteCode

PC: 0x55, opcode: JUMPDEST
PC: 0x56, opcode: POP
PC: 0x57, opcode: CALLDATALOAD
PC: 0x58, opcode: PUSH1 0x7b
PC: 0x5a, opcode: JUMP
PC: 0x5b, opcode: JUMPDEST
PC: 0x5c, opcode: STOP
…
PC: 0x7b, opcode: JUMPDEST
PC: 0x7c, opcode: PUSH1 0x00
PC: 0x7e, opcode: SSTORE
PC: 0x7f, opcode: JUMP

Push **_param** onto the stack and jump to 0x7b.

PEGASYS

CONSENSYS

ByteCode

PC: 0x55, opcode: JUMPDEST
PC: 0x56, opcode: POP
PC: 0x57, opcode: CALLDATALOAD
PC: 0x58, opcode: PUSH1 0x7b
PC: 0x5a, opcode: JUMP
PC: 0x5b, opcode: JUMPDEST
PC: 0x5c, opcode: STOP
…
PC: 0x7b, opcode: JUMPDEST
PC: 0x7c, opcode: PUSH1 0x00
PC: 0x7e, opcode: SSTORE
PC: 0x7f, opcode: JUMP

Store **_param** to storage location 0x00.

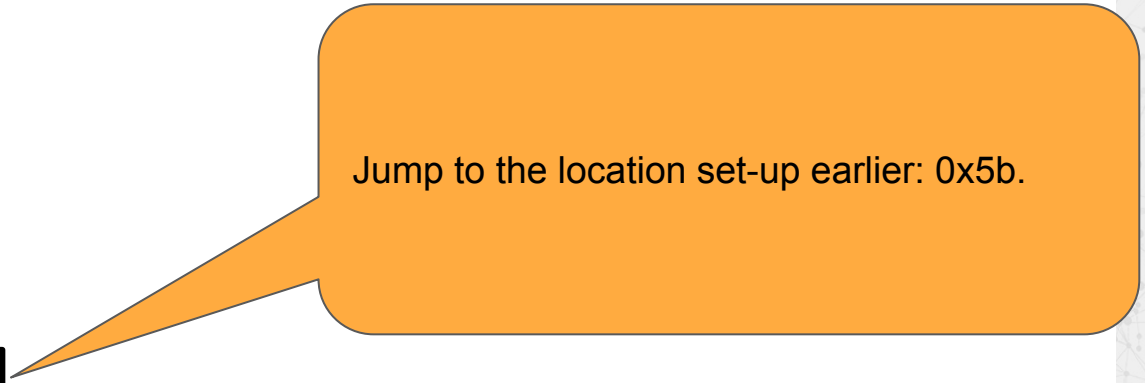
Stack now contains:
stack[0]: 0x5b

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x55, opcode: JUMPDEST
PC: 0x56, opcode: POP
PC: 0x57, opcode: CALLDATALOAD
PC: 0x58, opcode: PUSH1 0x7b
PC: 0x5a, opcode: JUMP
PC: 0x5b, opcode: JUMPDEST
PC: 0x5c, opcode: STOP
…
PC: 0x7b, opcode: JUMPDEST
PC: 0x7c, opcode: PUSH1 0x00
PC: 0x7e, opcode: SSTORE
PC: 0x7f, opcode: JUMP
```



PEGASYS

CONSENSYS

ByteCode

PC: 0x55, opcode: JUMPDEST
PC: 0x56, opcode: POP
PC: 0x57, opcode: CALLDATALOAD
PC: 0x58, opcode: PUSH1 0x7b
PC: 0x5a, opcode: JUMP
PC: 0x5b, opcode: JUMPDEST
PC: 0x5c, opcode: STOP
…
PC: 0x7b, opcode: JUMPDEST
PC: 0x7c, opcode: PUSH1 0x00
PC: 0x7e, opcode: SSTORE
PC: 0x7f, opcode: JUMP

STOP

End execution, indicate execution success, and don’t return any data.

PEGASYS

CONSENSYS

val2() view returns (uint256)

PEGASYS

CONSENSYS

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

Focusing on val2() view returns (uint256)

ByteCode

PC: 0x5d, opcode: JUMPDEST
PC: 0x5e, opcode: PUSH1 0x63
PC: 0x60, opcode: PUSH1 0x80
PC: 0x62, opcode: JUMP
…
PC: 0x80, opcode: JUMPDEST
PC: 0x81, opcode: PUSH1 0x01
PC: 0x83, opcode: SLOAD
PC: 0x84, opcode: DUP2
PC: 0x85, opcode: JUMP

Set-up “return address”

stack[0] = 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x5d, opcode: JUMPDEST
PC: 0x5e, opcode: PUSH1 0x63
PC: 0x60, opcode: PUSH1 0x80
PC: 0x62, opcode: JUMP
…
PC: 0x80, opcode: JUMPDEST
PC: 0x81, opcode: PUSH1 0x01
PC: 0x83, opcode: SLOAD
PC: 0x84, opcode: DUP2
PC: 0x85, opcode: JUMP
```

val2 is at storage location 0x01.

DUP2 is to duplicate 2nd stack item i.e stack[1]
stack[0] = 0x63
stack[1] = val2
stack[2] = 0x63

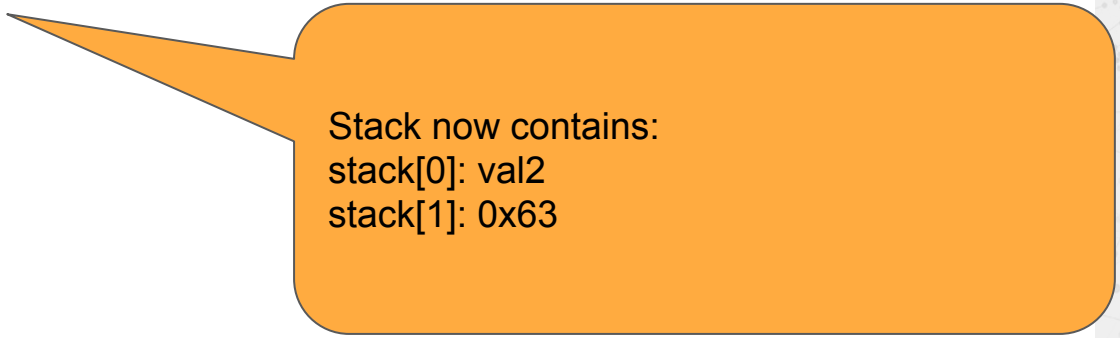
So at PC 0x85 we JUMP to 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```



PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```

Load the Free Memory Pointer from location 0x40.

Stack now contains:
stack[0]: 0x80
stack[1]: 0x40
stack[2]: val2
stack[3]: 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```

SWAP2: Swap stack[0] and stack[2]

Stack now contains:
stack[0]: val2
stack[1]: 0x40
stack[2]: 0x80
stack[3]: 0x63

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

DUP3: Push a copy of stack[2] onto the stack

Stack now contains:
stack[0]: 0x80
stack[1]: val2
stack[2]: 0x40
stack[3]: 0x80
stack[4]: 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```

MSTORE: Store stack[1] in memory location specified by stack[0].

memory[0x80] = val2

Stack now contains:
stack[0]: 0x40
stack[1]: 0x80
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```

MLOAD: Push onto the stack the value at memory location specified by stack[0].

stack[0] = memory[0x40]

Stack now contains:
stack[0]: 0x80
stack[1]: 0x80
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```

SWAP1: Swap stack[0] and stack[1].

Stack now contains:
stack[0]: 0x80
stack[1]: 0x80
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

DUP2: Push a copy of stack[1] onto the stack.

Stack now contains:
stack[0]: 0x80
stack[1]: 0x80
stack[2]: 0x80
stack[3]: 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```

SWAP1: Swap stack[0] and stack[1].

Stack now contains:
stack[0]: 0x80
stack[1]: 0x80
stack[2]: 0x80
stack[3]: 0x63

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

SUB: stack[0] = stack[0] - stack[1]

Stack now contains:
stack[0]: 0x00
stack[1]: 0x80
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

Push length of return value: uint256

Stack now contains:
stack[0]: 0x20
stack[1]: 0x00
stack[2]: 0x80
stack[3]: 0x63

PEGASYS

CONSENSYS

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

ADD: stack[0] = stack[0] + stack[1]

Stack now contains:
stack[0]: 0x20
stack[1]: 0x80
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

SWAP1: Swap stack[0] and stack[1]

Stack now contains:
stack[0]: 0x80
stack[1]: 0x20
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

RETURN

End execution, return a result and indicate successful execution.

stack[0] is the offset of the result.
stack[1] is the length of the result.

Stack ***contained***:
stack[0]: 0x80
stack[1]: 0x20
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN

RETURN

val2 was in memory 0x80 and has length 0x20. (see Slide 104)

stack[0] is the offset of the result.
stack[1] is the length of the result.

Stack ***contained***:
stack[0]: 0x80
stack[1]: 0x20
stack[2]: 0x63

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x63, opcode: JUMPDEST
PC: 0x64, opcode: PUSH1 0x40
PC: 0x66, opcode: DUP1
PC: 0x67, opcode: MLOAD
PC: 0x68, opcode: SWAP2
PC: 0x69, opcode: DUP3
PC: 0x6a, opcode: MSTORE
PC: 0x6b, opcode: MLOAD
PC: 0x6c, opcode: SWAP1
PC: 0x6d, opcode: DUP2
PC: 0x6e, opcode: SWAP1
PC: 0x6f, opcode: SUB
PC: 0x70, opcode: PUSH1 0x20
PC: 0x72, opcode: ADD
PC: 0x73, opcode: SWAP1
PC: 0x74, opcode: RETURN
```

Put together the return result, a uint256 which was put in memory, and return it.

val2 was loaded from storage (SLOAD), written to memory, and then returned.

PEGASYS

CONSENSYS

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

Focusing on val2() view returns (uint256)

val1() view returns (uint256)

PEGASYS

CONSENSYS

Example

```
pragma solidity >=0.4.23;

contract Simple {
  uint256 public val1;
  uint256 public val2;

  constructor() public {
    val2 = 3;
  }

  function set(uint256 _param) external {
    val1 = _param;
  }
}
```

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x75, opcode: JUMPDEST
PC: 0x76, opcode: PUSH1 0x63
PC: 0x78, opcode: PUSH1 0x86
PC: 0x7a, opcode: JUMP
…
PC: 0x86, opcode: JUMPDEST
PC: 0x87, opcode: PUSH1 0x00
PC: 0x89, opcode: SLOAD
PC: 0x8a, opcode: DUP2
PC: 0x8b, opcode: JUMP
PC: 0x8c, opcode: INVALID
```

Set-up “return address”

NOTE: This is the same program counter offset as was used for
val2() view returns(uint256)

That is: val1() and val2() use the same code fragment to return a uint256 value.

val1 is at storage location 0x00.

PEGASYS

CONSENSYS

Payable Fallback Functions

Example: SimpleFallBack

```
pragma solidity >=0.4.23;

contract SimpleFallBack {
  uint256 public val1;
  uint256 public val2;
  constructor() public {
    val2 = 3;
  }
  function set(uint256 _param) external {
    val1 = _param;
  }
  function () external payable {
  }
}
```

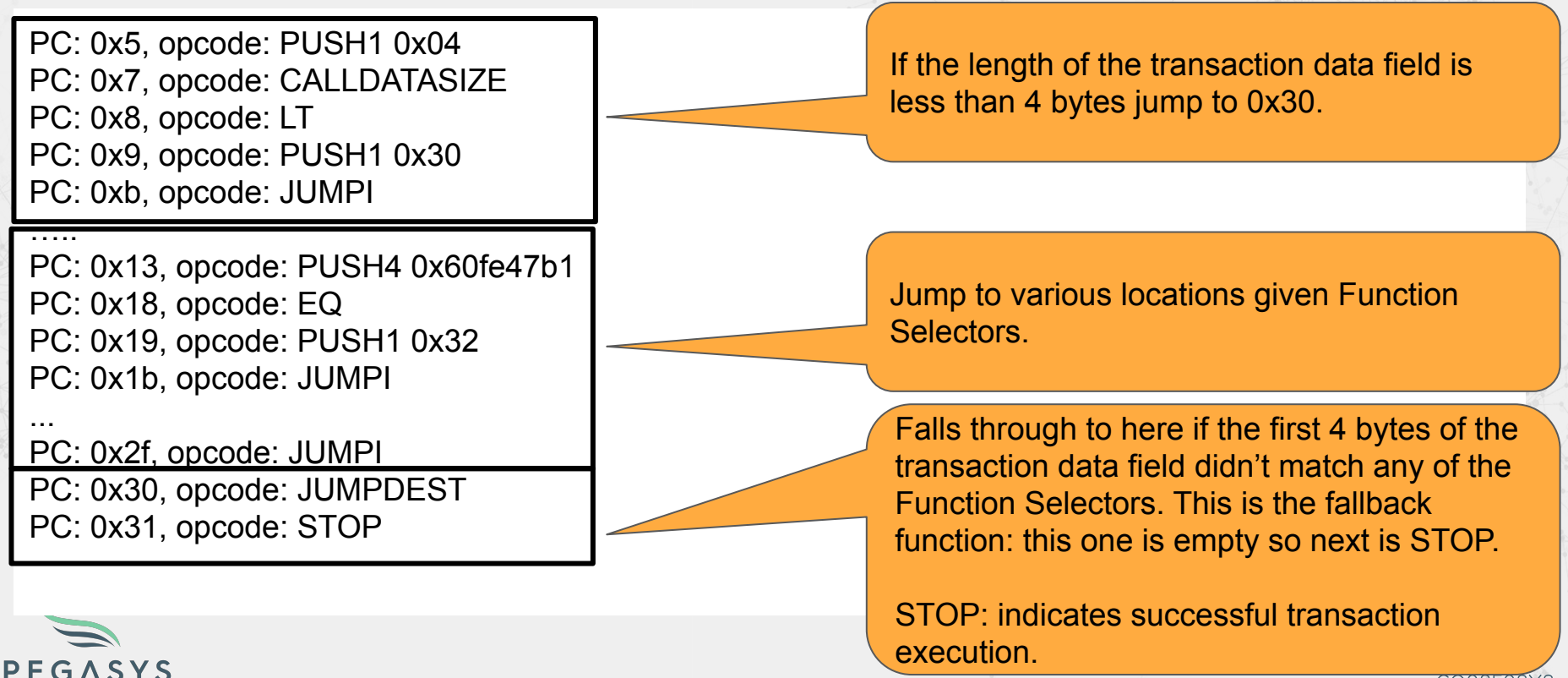
Fallback Functions:

- Allows Ether to be transferred to a contract as if it were an EOA.
- If someone is sending Ether to this contract and they muck up the transaction data field, it will still accept the transferred Wei, and ignore the transaction data field.

PEGASYS

CONSENSYS

ByteCode



PEGASYS

CONSENSYS

ByteCode

```
PC: 0x32, opcode: JUMPDEST
PC: 0x33, opcode: CALLVALUE
PC: 0x34, opcode: DUP1
PC: 0x35, opcode: ISZERO
PC: 0x36, opcode: PUSH1 0x3d
PC: 0x38, opcode: JUMPI
PC: 0x39, opcode: PUSH1 0x00
PC: 0x3b, opcode: DUP1
PC: 0x3c, opcode: REVERT
```

Now each function needs to check the transaction value field if the function is not payable.

PEGASYS

CONSENSYS

Storage

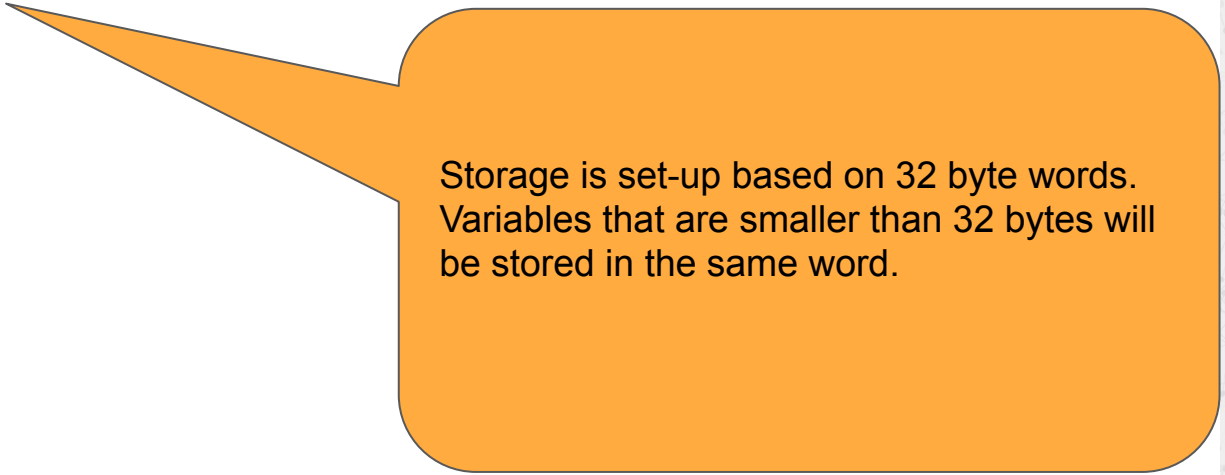
Storage Layout

PEGASYS

CONSENSYS

Example: Storage Layout

```
contract StorageLayout {
  byte  private valByte;
  uint256 private valUint256a;
  uint32 private valUint32;
  uint64 private valUint64;
  address private valAddress;
  uint256 private valUint256b;

  function set() external {
     valByte = 0x10;
     valUint256a = 0x11;
     valUint32 = 0x12;
     valUint64 = 0x13;
     valAddress = address(0x14);
     valUint256b = 0x15;
   }
}
```



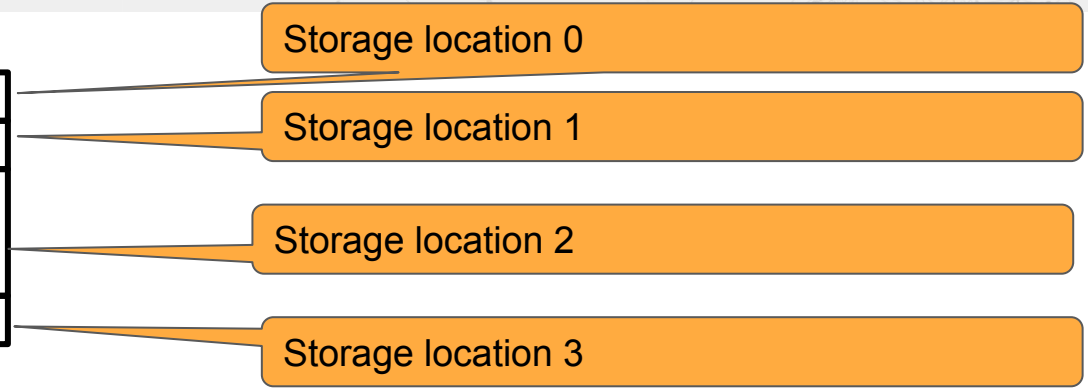
Example: Storage Layout

```
contract StorageLayout {
  byte  private valByte;
  uint256 private valUint256a;
  uint32 private valUint32;
  uint64 private valUint64;
  address private valAddress;
  uint256 private valUint256b;

  function set() external {
    valByte = 0x10;
    valUint256a = 0x11;
    valUint32 = 0x12;
    valUint64 = 0x13;
    valAddress = address(0x14);
    valUint256b = 0x15;
  }
}
```

Storage is set-up based on 32 byte words. Variables that are smaller than 32 bytes will be stored in the same word.

Example: Storage Layout

```
contract StorageLayout {
  byte  private valByte;
  uint256 private valUint256a;
  uint32 private valUint32;
  uint64 private valUint64;
  address private valAddress;
  uint256 private valUint256b;

  function set() external {
    valByte = 0x10;
    valUint256a = 0x11;
    valUint32 = 0x12;
    valUint64 = 0x13;
    valAddress = address(0x14);
    valUint256b = 0x15;
  }
}
```



ByteCode

```
PC: 0x35, opcode: JUMPDEST
PC: 0x36, opcode: PUSH1 0x00
PC: 0x38, opcode: DUP1
PC: 0x39, opcode: SLOAD
PC: 0x3a, opcode: PUSH1 0x10
PC: 0x3c, opcode: PUSH1 0xff
PC: 0x3e, opcode: NOT
PC: 0x3f, opcode: SWAP1
PC: 0x40, opcode: SWAP2
PC: 0x41, opcode: AND
PC: 0x42, opcode: OR
PC: 0x43, opcode: SWAP1
PC: 0x44, opcode: SSTORE
...
```

valByte = 0x10;

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x35, opcode: JUMPDEST
PC: 0x36, opcode: PUSH1 0x00
PC: 0x38, opcode: DUP1
PC: 0x39, opcode: SLOAD
PC: 0x3a, opcode: PUSH1 0x10
PC: 0x3c, opcode: PUSH1 0xff
PC: 0x3e, opcode: NOT
PC: 0x3f, opcode: SWAP1
PC: 0x40, opcode: SWAP2
PC: 0x41, opcode: AND
PC: 0x42, opcode: OR
PC: 0x43, opcode: SWAP1
PC: 0x44, opcode: SSTORE
```

Load storage location 0x00

Push the constant we want to set the storage value to.

Create a bit mask 0xFFFFFF….FF00

Mask off the bottom byte in the word loaded from the storage location

OR in the value being stored.

Store the result to storage location 0x00

PEGASYS

ByteCode

```
PC: 0x45, opcode: PUSH1 0x11
PC: 0x47, opcode: PUSH1 0x01
PC: 0x49, opcode: SSTORE
....
```

```
valUint256a = 0x11;
```

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x4a, opcode: PUSH1 0x02
PC: 0x4c, opcode: DUP1
PC: 0x4d, opcode: SLOAD
PC: 0x4e, opcode: PUSH1 0x12
PC: 0x50, opcode: PUSH4 0xffffffff
PC: 0x55, opcode: NOT
PC: 0x56, opcode: SWAP1
PC: 0x57, opcode: SWAP2
PC: 0x58, opcode: AND
PC: 0x59, opcode: OR
PC: 0x5a, opcode: PUSH12 0xffffffffffffffffffffffff00000000
PC: 0x67, opcode: NOT
PC: 0x68, opcode: AND
PC: 0x69, opcode: PUSH5 0x1300000000
PC: 0x6f, opcode: OR
PC: 0x70, opcode: PUSH12 0xffffffffffffffffffffffff
PC: 0x7d, opcode: AND
PC: 0x7e, opcode: PUSH1 0x05
PC: 0x80, opcode: PUSH1 0x62
PC: 0x82, opcode: SHL
PC: 0x83, opcode: OR
PC: 0x84, opcode: SWAP1
PC: 0x85, opcode: SSTORE
```

For storage location 2:

```
valUint32 = 0x12;
valUint64 = 0x13;
valAddress = address(0x14);
```

ByteCode

```
PC: 0x4a, opcode: PUSH1 0x02
PC: 0x4c, opcode: DUP1
PC: 0x4d, opcode: SLOAD
PC: 0x4e, opcode: PUSH1 0x12
PC: 0x50, opcode: PUSH4 0xffffffff
PC: 0x55, opcode: NOT
PC: 0x56, opcode: SWAP1
PC: 0x57, opcode: SWAP2
PC: 0x58, opcode: AND
PC: 0x59, opcode: OR
PC: 0x5a, opcode: PUSH12 0xffffffffffffffff00000000
PC: 0x67, opcode: NOT
PC: 0x68, opcode: AND
PC: 0x69, opcode: PUSH5 0x1300000000
PC: 0x6f, opcode: OR
PC: 0x70, opcode: PUSH12 0xffffffffffffffffffffffff
PC: 0x7d, opcode: AND
PC: 0x7e, opcode: PUSH1 0x05
PC: 0x80, opcode: PUSH1 0x62
PC: 0x82, opcode: SHL
PC: 0x83, opcode: OR
PC: 0x84, opcode: SWAP1
PC: 0x85, opcode: SSTORE
```

Load storage location 0x02

Mask in: valUint32 = 0x12;

Mask in: valUint64 = 0x13;

Mask in: valAddress = **address**(0x14);

Store to storage location 0x02

ByteCode

```
PC: 0x86, opcode: PUSH1 0x15
PC: 0x88, opcode: PUSH1 0x03
PC: 0x8a, opcode: SSTORE
PC: 0x8b, opcode: JUMP
```

```
valUint256b = 0x15;
```

PEGASYS

CONSENSYS

Summary

- Well laid out Solidity storage variables will result in fewer storage locations.
- Fewer storage locations will save you $$$
- In this example, there was no way to optimize storing valByte and it would have been cheaper to just use a uint256. (The code in Slide 129 is an example of https://ethereum.stackexchange.com/questions/3067/why-does-uint8-cost-more-gas-than-uint256)

PEGASYS

CONSENSYS

Bytes and String

Bytes and String

字节（bytes）和字符串（string）都是可变长度的字节数组，它们的行为方式相同。

PEGASYS

CONSENSYS

Example: Storage Bytes

```
contract StorageBytes {
  bytes private valBytes;

  function setBytes(byte _val) external {
      valBytes.push(_val);
  }
}
```

Bytes and String

- 字节（bytes）被分配一个存储槽。
- 对于bytes类型的私有变量valBytes，它位于存储槽[0]。
- 对于0到31字节：

Marker bit indicating Bytes with 31 or fewer bytes

PEGASYS

CONSENSYS

Bytes and String

- 字节（bytes）被分配一个存储槽。
- 对于bytes类型的私有变量valBytes，它位于存储槽[0]。
- 对于32个字节或更多：

Marker bit indicating Bytes with 32 or more bytes

Byte 31	Byte 30	Byte 29	...	Byte 2	Byte 1	Byte 0
Length						1

- At storage location: storage[keccak256(storage slot)]

byte[0]	byte[1]	byte[2]	...	byte[29]	byte[30]	byte[31]

- At storage location: storage[keccak256(storage slot) + 1]

byte[32]	byte[33]	byte[34]	...

ByteCode

```
PC: 0x2d, opcode: JUMPDEST
PC: 0x2e, opcode: PUSH1 0x51
PC: 0x30, opcode: PUSH1 0x04
PC: 0x32, opcode: DUP1
PC: 0x33, opcode: CALLDATASIZE
PC: 0x34, opcode: SUB
PC: 0x35, opcode: PUSH1 0x20
PC: 0x37, opcode: DUP2
PC: 0x38, opcode: LT
PC: 0x39, opcode: ISZERO
PC: 0x3a, opcode: PUSH1 0x41
PC: 0x3c, opcode: JUMPI
PC: 0x3d, opcode: PUSH1 0x00
PC: 0x3f, opcode: DUP1
PC: 0x40, opcode: REVERT
```

Program Counter of end code.

Length of transaction data field less size of Function Selector

Revert if the size of **byte** _val isn't at least 0x20 (32) bytes

All parameters are passed as 32 byte words

PEGASYS

CONSENSYS

ByteCode

```
PC: 0x41, opcode: JUMPDEST
PC: 0x42, opcode: POP
PC: 0x43, opcode: CALLDATALOAD
PC: 0x44, opcode: PUSH1 0x01
PC: 0x46, opcode: PUSH1 0x01
PC: 0x48, opcode: PUSH1 0xf8
PC: 0x4a, opcode: SHL
PC: 0x4b, opcode: SUB
PC: 0x4c, opcode: NOT
PC: 0x4d, opcode: AND
PC: 0x4e, opcode: PUSH1 0x53
PC: 0x50, opcode: JUMP
```

Stack is:
stack[0]: CALLDATASIZE - 0x04
stack[1]: 0x04
stack[2]: 0x51

Load one word of call data from offset 0x04

Create a bit mask 0x00FFFFFF..FF

Mask off **byte** _val

ByteCode

```
PC: 0x53, opcode: JUMPDEST
PC: 0x54, opcode: PUSH1 0x00
PC: 0x56, opcode: DUP2
PC: 0x57, opcode: SWAP1
PC: 0x58, opcode: DUP1
PC: 0x59, opcode: DUP1
PC: 0x5a, opcode: SLOAD
PC: 0x5b, opcode: PUSH1 0x3f
PC: 0x5d, opcode: DUP2
PC: 0x5e, opcode: AND
PC: 0x5f, opcode: DUP1
PC: 0x60, opcode: PUSH1 0x3e
PC: 0x62, opcode: DUP2
PC: 0x63, opcode: EQ
PC: 0x64, opcode: PUSH1 0x84
PC: 0x66, opcode: JUMPI
```

Stack is:
stack[0]: 0x??00000000..00
stack[1]: 0x51

Load storage location 0x00

Mask off what would be the length if the bytes contains fewer than 31 bytes.

Does **bytes private** valBytes currently contain exactly 31 bytes?

Recall the length is shifted left by 1.

PEGASYS

ByteCode

```
PC: 0x84, opcode: JUMPDEST
  PC: 0x85, opcode: PUSH1 0x00
  PC: 0x87, opcode: DUP5
  PC: 0x88, opcode: DUP2
  PC: 0x89, opcode: MSTORE
  PC: 0x8a, opcode: PUSH1 0x20
  PC: 0x8c, opcode: SWAP1
  PC: 0x8d, opcode: DUP2
  PC: 0x8e, opcode: SWAP1
  PC: 0x8f, opcode: SHA3
  PC: 0x90, opcode: PUSH1 0xff
  PC: 0x92, opcode: NOT
  PC: 0x93, opcode: DUP6
  PC: 0x94, opcode: AND
  PC: 0x95, opcode: SWAP1
  PC: 0x96, opcode: SSTORE
```

Stack is:
stack[0]: 0x??00000000..00
stack[1]: 0x??00000000..00
stack[2]: storage location 0x00
stack[3]: 0x00
stack[4]: 0x00
stack[5]: 0x??00000000..00
stack[6]: 0x??00000000..00
stack[7]: 0x51

Memory[0x00] = 0

stack[0] = KECCAK256 from
memory(stack[0]) stack[1] number of bytes
stack[0] is 0, stack[1] is 0x20

Mask off the top of the existing storage value, removing the length byte

Store the existing bytes into the storage location KECCAK256(the storage offset, 0x00).

PEGASYS

CONSENSYS

ByteCode

PC: 0x97, opcode: PUSH1 0x41
PC: 0x99, opcode: SWAP1
PC: 0x9a, opcode: SWAP5
PC: 0x9b, opcode: SSTORE
PC: 0x9c, opcode: JUMPDEST

Store the new length & indicator to storage location 0x00.

Length (0x20) shift left by 1 + 1 to indicate longer than one word.

ByteCode

```
PC: 0x9c, opcode: JUMPDEST
PC: 0x9d, opcode: POP
PC: 0x9e, opcode: POP
PC: 0x9f, opcode: POP
PC: 0xa0, opcode: SWAP1
PC: 0xa1, opcode: PUSH1 0x01
PC: 0xa3, opcode: DUP3
PC: 0xa4, opcode: SUB
PC: 0xa5, opcode: DUP2
PC: 0xa6, opcode: SLOAD
PC: 0xa7, opcode: PUSH1 0x01
PC: 0xa9, opcode: AND
PC: 0xaa, opcode: ISZERO
PC: 0xab, opcode: PUSH1 0xc2
PC: 0xad, opcode: JUMPI
PC: 0xae, opcode: SWAP1
PC: 0xaf, opcode: PUSH1 0x00
PC: 0xb1, opcode: MSTORE
PC: 0xb2, opcode: PUSH1 0x20
PC: 0xb4, opcode: PUSH1 0x00
PC: 0xb6, opcode: SHA3
PC: 0xb7, opcode: SWAP1
PC: 0xb8, opcode: PUSH1 0x20
PC: 0xba, opcode: SWAP2
PC: 0xbb, opcode: DUP3
PC: 0xbc, opcode: DUP3
PC: 0xbd, opcode: DIV
PC: 0xbe, opcode: ADD
PC: 0xbf, opcode: SWAP2
PC: 0xc0, opcode: SWAP1
PC: 0xc1, opcode: MOD
PC: 0xc2, opcode: JUMPDEST
PC: 0xc3, opcode: SWAP1
PC: 0xc4, opcode: SWAP2
PC: 0xc5, opcode: SWAP3
PC: 0xc6, opcode: SWAP1
PC: 0xc7, opcode: SWAP2
PC: 0xc8, opcode: SWAP1
PC: 0xc9, opcode: SWAP2
PC: 0xca, opcode: PUSH1 0x1f
PC: 0xcc, opcode: SUB
PC: 0xcd, opcode: PUSH2 0x0100
PC: 0xd0, opcode: EXP
PC: 0xd1, opcode: DUP2
PC: 0xd2, opcode: SLOAD
PC: 0xd3, opcode: DUP2
PC: 0xd4, opcode: PUSH1 0xff
PC: 0xd6, opcode: MUL
PC: 0xd7, opcode: NOT
PC: 0xd8, opcode: AND
PC: 0xd9, opcode: SWAP1
PC: 0xda, opcode: PUSH1 0x01
PC: 0xdc, opcode: PUSH1 0xf8
PC: 0xde, opcode: SHL
PC: 0xdf, opcode: DUP5
PC: 0xe0, opcode: DIV
PC: 0xe1, opcode: MUL
PC: 0xe2, opcode: OR
PC: 0xe3, opcode: SWAP1
PC: 0xe4, opcode: SSTORE
PC: 0xe5, opcode: POP
PC: 0xe6, opcode: POP
PC: 0xe7, opcode: POP
PC: 0xe8, opcode: JUMP
```

将byte_val插入到字节数组的正确位置。

PEGASYS

CONSENSYS

Arrays

Example: Storage Arrays

```
contract StorageArrays {
   uint256[] private arrayUint256;
   byte[] private arrayByte;

   function setUint256ArrayVal(uint256 _ofs, uint256 _val) external {
      arrayUint256[_ofs] = _val;
   }

   function setByteArrayVal(uint256 _ofs, byte _val) external {
      arrayByte[_ofs] = _val;
   }
}
```

Dynamic Arrays

Values for (`uint256[] private arrayUint256;`) are stored at locations:

storage[keccak256(storage slot number)+key] = value

Note:

- The number of elements in the dynamic array is stored at storage[storage slot number]

PEGASYS

CONSENSYS

Mappings

Example: Storage Mappings

```
contract StorageMappings {
  mapping(uint256 => uint256) private map;

  function setMapVal(uint256 _key, uint256 _val) external {
      map[_key] = _val;
  }
}
```

Mappings

Values for (`mapping(uint256 => uint256) private map;`) are stored at locations:

storage[keccak256(key . storage slot number)] = value

Note:

- “.” is concatenation.
- Nothing is stored at storage[storage slot number]

PEGASYS

CONSENSYS

Mapping of Mapping, Array of Arrays, Arrays of Mappings, Mappings of Arrays, etc

- 用于确定存储位置的规则在更复杂的结构（如映射的映射）中递归应用。
- 请参阅 https://solidity.readthedocs.io/en/v0.6.9/internals/layout_in_storage.html以获取更多详细信息。

PEGASYS

CONSENSYS

Memory

Memory Expansion

Could you write some code that:

Push32 0xFFFFFFF….FFFFF A BIG NUMBER

Push8 0x00

MSTORE

& hope to crash the EVM by doing out of memory.

PEGASYS

CONSENSYS

Memory Expansion

But:

- Each word that memory expands costs Gas.
- Requesting a LOT of memory will cause an out of gas error.

PEGASYS

CONSENSYS

Memory

0x00 - 0x3f	(64 bytes): scratch space for hashing methods
0x40 - 0x5f	(32 bytes): currently allocated memory size (aka. free memory pointer)
0x60 - 0x7f	(32 bytes): zero slot
0X80	Allocated memory starts here. For example: `uint[] memory a = new uint[](7);`

PEGASYS

CONSENSYS

CODECOPY and EXTCODECOPY

CODE COPY

- Code Copy: 将字节从此合约复制到内存中。
- Ext Code Copy: 将字节从另一个合约复制到内存中。

Usage:

- Revert Reason error messages.
- Deploy a contract from this contract.
- Any other static data.

PEGASYS

CONSENSYS

AuxData / MetaData

Aux Data / Solidity Metadata

".auxdata" : "a265627a7a723058209832b60d1aa283d2a1ca724b09775bd9232213e63c1e8eaf01c08741fd1b016864736f6c634300050a0032",

- Bytes included at the end of modern Solidity contracts.
- CBOR encoded: https://tools.ietf.org/html/rfc7049#page-7
- Indicates:
  - Swarm / IPFS message digest, which is location of the source code.
    - NOTE: You need to upload the code yourself.
  - Compiler name
  - Compiler version
- Allows tools like EtherScan to automatically verify that the deployed contract was compiled from specific source code.

PEGASYS

CONSENSYS

Aux Data / Solidity Metadata

".auxdata" : "a265627a7a723058209832b60d1aa283d2a1ca724b09775bd9232213e63c1e8eaf01c08741fd1b016864736f6c634300050a0032",

- A2 Map
- 65 6X mean array of characters. 5 is the length
- 627a7a7230 “bzzr0” which means Swarm. This could be IPFS.
- 58
- 20 length of message digest
- 9832b60d1aa283d2a1ca724b09775bd9232213e63c1e8eaf01c08741fd1b0168 message digest
- 64 6X mean array of characters. 4 is the length
- 736f6c63 "solc"
- 43 4X means array. 3 is length.
- 00050a 00,05,0A = version 0.5.10
- 0032 The length of the aux data: 0x32 = 50 bytes

PEGASYS

CONSENSYS

Aux Data / Solidity Metadata

For more information see:
https://solidity.readthedocs.io/en/v0.6.9/metadata.html

PEGASYS

CONSENSYS

Summary

Summary

EVM（以太坊虚拟机）是一种基于堆栈的处理器，具有以下访问权限：

- CallData：交易参数。
- Memory：事务内的临时数据存储。
- Storage：作为世界状态的一部分的持久存储。
- Code：存储代码和静态数据，如字符串。
- Logs：只写的事件日志输出。

PEGASYS

CONSENSYS

So much more to cover

- 跨合约调用
- 错误处理：断言（Assert）、要求（Require）、回滚（Revert）和异常（Exceptions）
- 事件/日志
- 关于存储的更多内容：bytes32和uint256之间的区别
- 关于内存变量的更多内容
- 早期以太坊中的奇怪合约
- 深入研究ERC20合约
- 深入研究以太坊主网上的一些重要合约

What else would you like covered in the next talk on this topic?

PEGASYS

CONSENSYS

Future Talks

- Jun 24: Code Merklization
  - Peter Robinson
- Jul 08: Polynomial Commitments
  - Raghavendra Ramesh
- July 13: Supply Chain on Blockchain Conference
  - Details: https://scobc.net/
  - Registrations: https://www.eventbrite.com.au/e/supply-chain-on-blockchain-conference-2020-tickets-103758121334
  - Early Bird Registrations (AU$20): Ends Friday June 12
  - Standard Registrations (AU$40)
- August 5: EIP-1559 and Escalators: Fixing the Ethereum Fee Market
  - Tim Beiko
  - **NOTE: 9:30 am Brisbane time zone**

PEGASYS

CONSENSYS