

---

# **pyevmasm Documentation**

**Trail of Bits**

**Jul 18, 2018**



---

## Contents:

---

<b>1</b>	<b>API Reference</b>	<b>1</b>
1.1	evmasm . . . . .	1
<b>2</b>	<b>Indices and tables</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>



### 1.1 evmasm

**class** `pyevmasm.evmasm.Instruction` (*opcode, name, operand\_size, pops, pushes, fee, description, operand=None, pc=0*)

**bytes**

Encoded instruction

**description**

Colloquial description of the instruction

**fee**

The basic gas fee of the instruction

**group**

Instruction classification as per the yellow paper

**has\_operand**

True if the instruction uses an immediate operand

**is\_arithmetic**

True if the instruction is an arithmetic operation

**is\_branch**

True if the instruction is a jump

**is\_endtx**

True if the instruction is a transaction terminator

**is\_environmental**

True if the instruction access enviromental data

**is\_starttx**

True if the instruction is a transaction initiator

**is\_system**

True if the instruction is a system operation

**is\_terminator**

True if the instruction is a basic block terminator

**mnemonic**

Alias for name

**name**

The instruction name/mnemonic

**opcode**

The opcode as an integer

**operand\_size**

The immediate operand size

**parse\_operand** (*buf*)

Parses an operand from buf

**Parameters** *buf* (*iterator/generator/string*) – a buffer

**pops**

Number words popped from the stack

**pushes**

Number words pushed to the stack

**reads\_from\_memory**

True if the instruction reads from memory

**reads\_from\_stack**

True if the instruction reads from stack

**reads\_from\_storage**

True if the instruction reads from the storage

**semantics**

Canonical semantics

**size**

Size of the encoded instruction

**uses\_block\_info**

True if the instruction access block information

**uses\_stack**

True if the instruction reads/writes from/to the stack

**writes\_to\_memory**

True if the instruction writes to memory

**writes\_to\_stack**

True if the instruction writes to the stack

**writes\_to\_storage**

True if the instruction writes to the storage

**class** `pyevmasm.evmasm.InstructionTable` (*\*args*, *\*\*kwargs*)

**EVM Instruction factory** Implements an immutable, iterable instruction LUT that can be indexed by both mnemonic or opcode.

Example:

```

>>> from pyevmasm import instruction_table
>>> instruction_table[0]
Instruction(0x0, 'STOP', 0, 0, 0, 0, 'Halts execution.', None, 0)
>>> instruction_table['STOP']
Instruction(0x0, 'STOP', 0, 0, 0, 0, 'Halts execution.', None, 0)
>>> i = instruction_table.__iter__()
>>> i.__next__()
Instruction(0x0, 'STOP', 0, 0, 0, 0, 'Halts execution.', None, 0)
>>> i.__next__()
Instruction(0x1, 'ADD', 0, 2, 1, 3, 'Addition operation.', None, 0)
>>> i.__next__()
Instruction(0x2, 'MUL', 0, 2, 1, 5, 'Multiplication operation.', None, 0)
>>> i.__next__()
Instruction(0x3, 'SUB', 0, 2, 1, 3, 'Subtraction operation.', None, 0)

```

**exception** pyevmasm.evmasm.**UnknownMnemonicError**

**exception** pyevmasm.evmasm.**UnknownOpcodeError**

pyevmasm.evmasm.**assemble** (*asmcode*, *pc=0*)

Assemble an EVM program

#### Parameters

- **asmcode** (*str*) – an evm assembler program
- **pc** (*int*) – program counter of the first instruction(optional)

**Returns** the hex representation of the bytecode

**Return type** str

Example use:

```

>>> assemble('''PUSH1 0x60
                BLOCKHASH
                MSTORE
                PUSH1 0x2
                PUSH2 0x100
                ''')
...
b"````@R` }a{\ "

```

pyevmasm.evmasm.**assemble\_all** (*asmcode*, *pc=0*)

Assemble a sequence of textual representation of EVM instructions

#### Parameters

- **asmcode** (*str*) – assembly code for any number of instructions
- **pc** (*int*) – program counter of the first instruction(optional)

**Returns** An generator of Instruction objects

**Return type** generator[Instructions]

Example use:

```

>>> assemble_one('''PUSH1 0x60
                    PUSH1 0x40
                    MSTORE
                    PUSH1 0x2

```

(continues on next page)

(continued from previous page)

```

        PUSH2 0x108
        PUSH1 0x0
        POP
        SSTORE
        PUSH1 0x40
        MLOAD
        '''

```

`pyevmasm.evmasm.assemble_hex(asmcode, pc=0)`

Assemble an EVM program

#### Parameters

- **asmcode** (*str* | *iterator*[*Instruction*]) – an evm assembler program
- **pc** (*int*) – program counter of the first instruction(optional)

**Returns** the hex representation of the bytecode

**Return type** *str*

Example use:

```

>>> assemble_hex(''PUSH1 0x60
                    BLOCKHASH
                    MSTORE
                    PUSH1 0x2
                    PUSH2 0x100
                    ''')
...
"0x6060604052600261010"

```

`pyevmasm.evmasm.assemble_one(asmcode, pc=0)`

Assemble one EVM instruction from its textual representation.

#### Parameters

- **asmcode** (*str*) – assembly code for one instruction
- **pc** (*int*) – program counter of the instruction(optional)

**Returns** An *Instruction* object

**Return type** *Instruction*

Example use:

```

>>> print assemble_one('LT')

```

`pyevmasm.evmasm.disassemble(bytecode, pc=0)`

Disassemble an EVM bytecode

#### Parameters

- **bytecode** (*str* | *bytes* | *bytearray*) – binary representation of an evm bytecode
- **pc** (*int*) – program counter of the first instruction(optional)

**Returns** the text representation of the assembler code

Example use:



```
>>> disassemble("`@R` }a{`")
...
PUSH1 0x60
BLOCKHASH
MSTORE
PUSH1 0x2
PUSH2 0x100
```

`pyevmasm.evmasm.disassemble_all(bytecode, pc=0)`  
 Disassemble all instructions in bytecode

#### Parameters

- **bytecode** (*str* | *bytes* | *bytearray* | *iterator*) – an evm bytecode (binary)
- **pc** (*int*) – program counter of the first instruction(optional)

**Returns** An generator of Instruction objects

**Return type** list[*Instruction*]

Example use:

```
>>> for inst in disassemble_all(bytecode):
...     print(instr)

...
PUSH1 0x60
PUSH1 0x40
MSTORE
PUSH1 0x2
PUSH2 0x108
PUSH1 0x0
POP
SSTORE
PUSH1 0x40
MLOAD
```

`pyevmasm.evmasm.disassemble_hex(bytecode, pc=0)`  
 Disassemble an EVM bytecode

#### Parameters

- **bytecode** (*str*) – canonical representation of an evm bytecode (hexadecimal)
- **pc** (*int*) – program counter of the first instruction(optional)

**Returns** the text representation of the assembler code

**Return type** *str*

Example use:

```
>>> disassemble_hex("0x6060604052600261010")
...
PUSH1 0x60
BLOCKHASH
MSTORE
PUSH1 0x2
PUSH2 0x100
```

`pyevmasm.evmasm.disassemble_one` (*bytecode*, *pc=0*)

Disassemble a single instruction from a bytecode

**Parameters**

- **bytecode** (*str* | *bytes* | *bytearray* | *iterator*) – the bytecode stream
- **pc** (*int*) – program counter of the instruction(optional)

**Returns** an Instruction object

**Return type** *Instruction*

Example use:

```
>>> print disassemble_one('`')
```

`pyevmasm.evmasm.instruction`

Instance of InstructionTable for EVM. (see; InstructionTable)

## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### p

`pyevmasm.evmasm`, [1](#)



## A

assemble() (in module pyevmasm.evmasm), 3  
assemble\_all() (in module pyevmasm.evmasm), 3  
assemble\_hex() (in module pyevmasm.evmasm), 4  
assemble\_one() (in module pyevmasm.evmasm), 4

## B

bytes (pyevmasm.evmasm.Instruction attribute), 1

## D

description (pyevmasm.evmasm.Instruction attribute), 1  
disassemble() (in module pyevmasm.evmasm), 4  
disassemble\_all() (in module pyevmasm.evmasm), 5  
disassemble\_hex() (in module pyevmasm.evmasm), 5  
disassemble\_one() (in module pyevmasm.evmasm), 5

## F

fee (pyevmasm.evmasm.Instruction attribute), 1

## G

group (pyevmasm.evmasm.Instruction attribute), 1

## H

has\_operand (pyevmasm.evmasm.Instruction attribute), 1

## I

Instruction (class in pyevmasm.evmasm), 1  
instruction (in module pyevmasm.evmasm), 6  
InstructionTable (class in pyevmasm.evmasm), 2  
is\_arithmetic (pyevmasm.evmasm.Instruction attribute), 1  
is\_branch (pyevmasm.evmasm.Instruction attribute), 1  
is\_endtx (pyevmasm.evmasm.Instruction attribute), 1  
is\_environmental (pyevmasm.evmasm.Instruction attribute), 1  
is\_starttx (pyevmasm.evmasm.Instruction attribute), 1  
is\_system (pyevmasm.evmasm.Instruction attribute), 1  
is\_terminator (pyevmasm.evmasm.Instruction attribute), 2

## M

mnemonic (pyevmasm.evmasm.Instruction attribute), 2

## N

name (pyevmasm.evmasm.Instruction attribute), 2

## O

opcode (pyevmasm.evmasm.Instruction attribute), 2  
operand\_size (pyevmasm.evmasm.Instruction attribute), 2

## P

parse\_operand() (pyevmasm.evmasm.Instruction method), 2  
pops (pyevmasm.evmasm.Instruction attribute), 2  
pushes (pyevmasm.evmasm.Instruction attribute), 2  
pyevmasm.evmasm (module), 1

## R

reads\_from\_memory (pyevmasm.evmasm.Instruction attribute), 2  
reads\_from\_stack (pyevmasm.evmasm.Instruction attribute), 2  
reads\_from\_storage (pyevmasm.evmasm.Instruction attribute), 2

## S

semantics (pyevmasm.evmasm.Instruction attribute), 2  
size (pyevmasm.evmasm.Instruction attribute), 2

## U

UnknownMnemonicError, 3  
UnknownOpcodeError, 3  
uses\_block\_info (pyevmasm.evmasm.Instruction attribute), 2  
uses\_stack (pyevmasm.evmasm.Instruction attribute), 2

## W

writes\_to\_memory (pyevmasm.evmasm.Instruction attribute), 2

`writes_to_stack` (pyevmasm.evmasm.Instruction attribute), [2](#)  
`writes_to_storage` (pyevmasm.evmasm.Instruction attribute), [2](#)