Opcodes

Notation : `opcode`

Description : The EVM has several opcodes which function as an assembly language:

0.0.1 EVM Code

The bytecode that the EVM can natively execute. Used to explicitly specify the meaning of a message to an account.

Notation : `contract`

Description : A piece of EVM Code that may be associated with an Account or an Autonomous Object.

0.0.2 Opcodes/EVM Assembly

The human readable version of EVM code. But what exactly are these computer instructions that can be executed with the same level of veracity and certainty as Bitcoin transactions? How do they come about, what makes them up, how are they kept in order, and what makes them execute? The first part of answering this question is understanding opcodes. In traditional machine architectures, you may not be introduced to working with processor-level assembly instructions for some time. In Ethereum however, they are essential to understanding the protocol because they are the most minute and subtle (yet HUGELY important) things going on in the Ethereum Blockchain at any moment, and they are the real "currency," that Ethereum trades in. I'll explain what I mean by that in a minute. First, let's go over a few Opcodes:[1]

The STOP Opcode is used in order to stop a computation once it has completed, or to halt a computation if it has run out of gas. The ADD, MUL, SUB, and DIV operations are addition, multiplication, subtraction and division operations. The In/Out columns refer to inputs (to `potential_state`), the state which decides every new `actual_state`.

[1] A full list of Opcodes is in Appendix A

1 Opcodes

1.1 0x10's: Comparisons and Bitwise Logic Operations

Data	Opcode	Gas	Input	Output	Description
0x00	STOP	0	0	0	Halts execution.
0x01	ADD	3	2	1	Addition operation.
0x02	MUL	5	2	1	Multiplication operation.
0x03	SUB	3	2	1	Subtraction operation.
0x04	DIV	5	2	1	Integer division operation.
0x05	SDIV	5	2	1	Signed integer division operation (truncated.)
0x06	MOD	5	2	1	Modulo remainder operation.
0x07	SMOD	5	2	1	Signed modulo remainder operation.
0x08	ADDMOD	8	3	1	Modulo addition operation.
0x09	MULMOD	8	3	1	Modulo multiplication operation.
0x0a	EXP	10	2	1	Exponential operation.
0x0b	SIGNEXTEND	5	2	1	Extend the length of two's complementary signed integer.
0x10	LT	3	2	1	Less-than comparison.
0x11	GT	3	2	1	Greater-than comparison.
0x12	SLT	3	2	1	Signed less-than comparison.
0x13	SGT	3	2	1	Signed greater-than comparison.
0x14	EQ	3	2	1	Equality comparison.
0x15	ISZERO	3	1	1	Simple not operator.
0x16	AND	3	2	1	Bitwise AND operation.
0x17	OR	3	2	1	Bitwise OR operation.
0x18	XOR	3	2	1	Bitwise XOR operation.
0x19	NOT	3	1	1	Bitwise NOT operation.
0x1a	BYTE	3	2	1	Retrieve single byte from word.

1.2 0x20's: SHA3

Data	Opcode	Gas	Input	Output	Description
0x20	SHA3	30	2	1	Compute a Keccak-256 hash.

1.3 0x30's: Environmental Information

Data	Opcode	Gas	Input	Output	Description
0x30	ADDRESS	2	0	1	Get the address of the currently executing account.
0x31	BALANCE	400	1	1	Get the balance of the given account.

0x32	ORIGIN	2	0	1	Get execution origination address. This is always the original sender of a transaction, never a contract account.
0x33	CALLER	2	0	1	Get caller address. This is the address of the account that is directly responsible for this execution.
0x34	CALLVALUE	2	0	1	Get deposited value by the instruction/transaction responsible for this execution.
0x35	CALLDATALOAD	3	1	1	Get input data of the current environment.
0x36	CALLDATASIZE	2	0	1	Get size of input data in current environment. This refers to the optional data field that can be passed with a message call instruction or transaction.
0x37	CALLDATACOPY	3	3	0	Copy input data in the current environment to memory. This refers to the optional data field passed with the message call instruction or transaction.
0x38	CODESIZE	2	0	1	Get size of code running in the current environment.
0x39	CODECOPY	3	3	0	Copy the code running in the current environment to memory.
0x3a	GASPRICE	2	0	1	Get the price of gas in the current environment. This is the gas price specified by the originating transaction.
0x3b	EXTCODESIZE	700	1	1	Get the size of an account's code.
0x3c	EXTCODECOPY	700	4	0	Copy an account's code to memory.
0x3d	RETURNDATASIZE	2	0	1	
0x3e	RETURNDATACOPY	3	3	0	

1.4 0x40's: Block Data

Data	Opcode	Gas	Input	Output	Description
0x40	BLOCKHASH	20	1	1	Get the hash of one of the 256 most recent blocks. [2]
0x41	COINBASE	2	0	1	Look up a block's beneficiary address by its hash.
0x42	TIMESTAMP	2	0	1	Look up a block's timestamp by its hash.
0x43	NUMBER	2	0	1	Look up a block's number by its hash.
0x44	DIFFICULTY	2	0	1	Look up a block's difficulty by its hash.

[2] A value of 0 is left on the stack if the block number is more than 256 in number behind the current one, or if it is a number greater than the current one.

0x45	GASLIMIT	2	0	1	Look up a block's gas limit by its hash.

1.5 0x50's: Stack, memory, storage, and flow operations.

Data	Opcode	Gas	Input	Output	Description
0x50	POP	2	1	0	Removes an item from the stack.
0x51	MLOAD	3	1	1	Load a word from memory.
0x52	MSTORE	3	2	0	Save a word to memory.
0x53	MSTORE8	3	2	0	Save a byte to memory.
0x54	SLOAD	200	1	1	Load a word from storage.
0x55	SSTORE	0	2	0	Save a word to storage.
0x56	JUMP	8	1	0	Alter the program counter.
0x57	JUMPI	10	2	0	Conditionally alter the program counter.
0x58	PC	2	0	1	Look up the value of the program counter prior to the increment resulting from this instruction.
0x59	MSIZE	2	0	1	Get the size of active memory in bytes.
0x5a	GAS	2	0	1	Get the amount of available gas, including the corresponding reduction for the cost of this instruction.
0x5b	JUMPDEST	1	0	0	Mark a valid destination for jumps. [3]

1.6 0x60-70's: Push Operations

Data	Opcode	Gas	Input	Output	Description
0x60	PUSH1	-	0	1	Place a 1-byte item on the stack.
0x61	PUSH2	-	0	1	Place a 2-byte item on the stack.
0x62	PUSH3	-	0	1	Place a 3-byte item on the stack.
0x63	PUSH4	-	0	1	Place a 4-byte item on the stack.
0x64	PUSH5	-	0	1	Place a 5-byte item on the stack.
0x65	PUSH6	-	0	1	Place a 6-byte item on the stack.
0x66	PUSH7	-	0	1	Place a 7-byte item on the stack.
0x67	PUSH8	-	0	1	Place a 8-byte item on the stack.
0x68	PUSH9	-	0	1	Place a 9-byte item on the stack.
0x69	PUSH10	-	0	1	Place a 10-byte item on the stack.
0x6a	PUSH11	-	0	1	Place a 11-byte item on the stack.
0x6b	PUSH12	-	0	1	Place a 12-byte item on the stack.
0x6c	PUSH13	-	0	1	Place a 13-byte item on the stack.
0x6d	PUSH14	-	0	1	Place a 14-byte item on the stack.
0x6e	PUSH15	-	0	1	Place a 15-byte item on the stack.
0x6f	PUSH16	-	0	1	Place a 16-byte item on the stack.

[3] This operation has no effect on the `machine_state during execution.`

0x70	PUSH17	-	0	1	Place a 17-byte item on the stack.
0x71	PUSH18	-	0	1	Place a 18-byte item on the stack.
0x72	PUSH19	-	0	1	Place a 19-byte item on the stack.
0x73	PUSH20	-	0	1	Place a 20-byte item on the stack.
0x74	PUSH21	-	0	1	Place a 21-byte item on the stack.
0x75	PUSH22	-	0	1	Place a 22-byte item on the stack.
0x76	PUSH23	-	0	1	Place a 23-byte item on the stack.
0x77	PUSH24	-	0	1	Place a 24-byte item on the stack.
0x78	PUSH25	-	0	1	Place a 25-byte item on the stack.
0x79	PUSH26	-	0	1	Place a 26-byte item on the stack.
0x7a	PUSH27	-	0	1	Place a 27-byte item on the stack.
0x7b	PUSH28	-	0	1	Place a 28-byte item on the stack.
0x7c	PUSH29	-	0	1	Place a 29-byte item on the stack.
0x7d	PUSH30	-	0	1	Place a 30-byte item on the stack.
0x7e	PUSH31	-	0	1	Place a 31-byte item on the stack.
0x7f	PUSH32	-	0	1	Place a 32-byte item on the stack.

1.7 0x80's: Duplication Operations

Data	Opcode	Gas	Input	Output	Description
0x80	DUP1	-	1	2	Duplicate the 1st item in the stack.
0x81	DUP2	-	2	3	Duplicate the 2nd item in the stack.
0x82	DUP3	-	3	4	Duplicate the 3rd item in the stack.
0x83	DUP4	-	4	5	Duplicate the 4th item in the stack.
0x84	DUP5	-	5	6	Duplicate the 5th item in the stack.
0x85	DUP6	-	6	7	Duplicate the 6th item in the stack.
0x86	DUP7	-	7	8	Duplicate the 7th item in the stack.
0x87	DUP8	-	8	9	Duplicate the 8th item in the stack.
0x88	DUP9	-	9	10	Duplicate the 9th item in the stack.
0x89	DUP10	-	10	11	Duplicate the 10th item in the stack.
0x8a	DUP11	-	11	12	Duplicate the 11th item in the stack.
0x8b	DUP12	-	12	13	Duplicate the 12th item in the stack.
0x8c	DUP13	-	13	14	Duplicate the 13th item in the stack.
0x8d	DUP14	-	14	15	Duplicate the 14th item in the stack.
0x8e	DUP15	-	15	16	Duplicate the 15th item in the stack.
0x8f	DUP16	-	16	17	Duplicate the 16th item in the stack.

1.8 0x90's: Swap Operations

Data	Opcode	Gas	Input	Output	Description
0x90	SWAP1	-	2	2	Exchange the 1st and 2nd stack items.
0x91	SWAP2	-	3	3	Exchange the 1st and 3rd stack items.
0x92	SWAP3	-	4	4	Exchange the 1st and 4th stack items.

0x93	SWAP4	-	5	5	Exchange the 1st and 5th stack items.
0x94	SWAP5	-	6	6	Exchange the 1st and 6th stack items.
0x95	SWAP6	-	7	7	Exchange the 1st and 7th stack items.
0x96	SWAP7	-	8	8	Exchange the 1st and 8th stack items.
0x97	SWAP8	-	9	9	Exchange the 1st and 9th stack items.
0x98	SWAP9	-	10	10	Exchange the 1st and 10th stack items.
0x99	SWAP10	-	11	11	Exchange the 1st and 11th stack items.
0x9a	SWAP11	-	12	12	Exchange the 1st and 12th stack items.
0x9b	SWAP12	-	13	13	Exchange the 1st and 13th stack items.
0x9c	SWAP13	-	14	14	Exchange the 1st and 14th stack items.
0x9d	SWAP14	-	15	15	Exchange the 1st and 15th stack items.
0x9e	SWAP15	-	16	16	Exchange the 1st and 16th stack items.
0x9f	SWAP16	-	17	17	Exchange the 1st and 17th stack items.

1.9 0xa0's: Logging Operations

Data	Opcode	Gas	Input	Output	Description
0xa0	LOG0	375	2	0	Append log record with 0 topics.
0xa1	LOG1	750	3	0	Append log record with 1 topic.
0xa2	LOG2	1125	4	0	Append log record with 2 topic.
0xa3	LOG3	1500	5	0	Append log record with 3 topic.
0xa4	LOG4	1875	6	0	Append log record with 4 topic.

1.10 0xf0's: System Operations

Data	Opcode	Gas	Input	Output	Description
0xf0	CREATE	32000	3	1	Create a new contract account. Operand order is: value, input offset, input size.
0xf1	CALL	700	7	1	Message-call into an account. The operand order is: gas, to, value, in offset, in size, out offset, out size.
0xf2	CALLCODE	700	7	1	Message-call into this account with an alternative account's code. Exactly equivalent to CALL, except the recipient is the same account as at present, but the code is overwritten.
0xf3	RETURN	0	2	0	Halt execution, then return output data. This defines the output at the moment of the halt.

0xf4	DELEGATECALL	700	6	1	Message-call into this account with an alternative account's code, but with persisting values for `sender` and `value`. DELEGATECALL takes one less argument than CALL. This means that the recipient is in fact the same account as at present, but that the code is overwritten *and* the context is almost entirely identical.
0xf5	CALLBLACKBOX	40	7	1	-
0xfa	STATICCALL	40	6	1	-
0xfd	REVERT	0	2	0	-
0xfe	INVALID	-	1	0	Designated invalid instruction.
0xff	SELFDESTRUCT	5000	1	0	Halt execution and register the account for later deletion.

References

[1] V. Buterin, *Pyethereum source*, `https://www.github.com/ethereum/pyethereum/ethereum/opcodes.py`, 2017.

[2] D. G. Wood, *Ethereum: A secure decentralised generalised transaction ledger*, `https://github.com/ethereum/yellowpaper`, 2017.