title: Opcodes for the EVM description: A list of all available opcodes for the Ethereum virtual machine. lang: en

Overview {#overview}

This is an updated version of the EVM reference page at wolflo/evm-opcodes. Also drawn from the Yellow Paper, the Jello Paper, and the geth implementation. This is intended to be an accessible reference, but it is not particularly rigorous. If you want to be certain of correctness and aware of every edge case, using the Jello Paper or a client implementation is advisable.

Looking for an interactive reference? Check out evm.codes.

For operations with dynamic gas costs, see gas.md.

Quick tip: To view entire lines, use `[shift] + scroll` to scroll horizontally on desktop.

Stack	Name	Gas	Initial Stack	Resulting Stack	Mem / Storage	Notes
00	STOP	0				halt execution
01	ADD	3	`a, b`	`a + b`		(u)int256 addition modulo 2**256
02	MUL	5	`a, b`	`a * b`		(u)int256 multiplication modulo 2**256
03	SUB	3	`a, b`	`a - b`		(u)int256 addition modulo 2**256
04	DIV	5	`a, b`	`a // b`		uint256 division
05	SDIV	5	`a, b`	`a // b`		int256 division
06	MOD	5	`a, b`	`a % b`		uint256 modulus
07	SMOD	5	`a, b`	`a % b`		int256 modulus
08	ADDMOD	8	`a, b, N`	`(a + b) % N`		(u)int256 addition modulo N
09	MULMOD	8	`a, b, N`	`(a * b) % N`		(u)int256 multiplication modulo N
0A	EXP	A1	`a, b`	`a ** b`		uint256 exponentiation modulo 2**256
0B	SIGNEXTEND	5	`b, x`	`SIGNEXTEND(x, b)`		sign extend `x` from `(b+1)` bytes to 32 bytes
0C-0F	*invalid*					
10	LT	3	`a, b`	`a < b`		uint256 less-than
11	GT	3	`a, b`	`a > b`		uint256 greater-than
12	SLT	3	`a, b`	`a < b`		int256 less-than
13	SGT	3	`a, b`	`a > b`		int256 greater-than
14	EQ	3	`a, b`	`a == b`		(u)int256 equality
15	ISZERO	3	`a`	`a == 0`		(u)int256 iszero
16	AND	3	`a, b`	`a && b`		bitwise AND
17	OR	3	`a b`	`a || b`		bitwise OR
18	XOR	3	`a, b`	`a ^ b`		bitwise XOR
19	NOT	3	`a`	`~a`		bitwise NOT
1A	BYTE	3	`i, x`	`(x >> (248 - i * 8)) && 0xFF`		`i`th byte of (u)int256 `x`, from the left
1B	SHL	3	`shift, val`	`val << shift`		shift left
1C	SHR	3	`shift, val`	`val >> shift`		logical shift right
1D	SAR	3	`shift, val`	`val >> shift`		arithmetic shift right
1E-1F	*invalid*					
20	KECCAK256	A2	`ost, len`	`keccak256(mem[ost:ost+len-1])`		keccak256
21-2F	*invalid*					
30	ADDRESS	2	.	`address(this)`		address of executing contract
31	BALANCE	A5	`addr`	`addr.balance`		balance, in wei
32	ORIGIN	2	.	`tx.origin`		address that originated the tx
33	CALLER	2	.	`msg.sender`		address of msg sender
34	CALLVALUE	2	.	`msg.value`		msg value, in wei
35	CALLDATALOAD	3	`idx`	`msg.data[idx:idx+32]`		read word from msg data at index `idx`
36	CALLDATASIZE	2	.	`len(msg.data)`		length of msg data, in bytes
37	CALLDATACOPY	A3	`dstOst, ost, len`	.	mem[dstOst:dstOst+len-1] := msg.data[ost:ost+len-1]	copy msg data
38	CODESIZE	2	.	`len(this.code)`		length of executing contract's code, in bytes
39	CODECOPY	A3	`dstOst, ost, len`	.		mem[dstOst:dstOst+len-1] := this.code[ost:ost+len-1]	copy executing contract's bytecode
3A	GASPRICE	2	.	`tx.gasprice`		gas price of tx, in wei per unit gas **
3B	EXTCODESIZE	A5	`addr`	`len(addr.code)`		size of code at addr, in bytes
3C	EXTCODECOPY	A4	`addr, dstOst, ost, len`	.	mem[dstOst:dstOst+len-1] := addr.code[ost:ost+len-1]	copy code from `addr`
3D	RETURNDATASIZE	2	.	`size`		size of returned data from last external call, in bytes
3E	RETURNDATACOPY	A3	`dstOst, ost, len`	.	mem[dstOst:dstOst+len-1] := returndata[ost:ost+len-1]	copy returned data from last external call
3F	EXTCODEHASH	A5	`addr`	`hash`		hash = addr.exists ? keccak256(addr.code) : 0
40	BLOCKHASH	20	`blockNum`	`blockHash(blockNum)`		
41	COINBASE	2	.	`block.coinbase`		address of miner of current block
42	TIMESTAMP	2	.	`block.timestamp`		timestamp of current block
43	NUMBER	2	.	`block.number`		number of current block
44	PREVRANDAO	2	.	`randomness beacon`		randomness beacon
45	GASLIMIT	2	.	`block.gaslimit`		gas limit of current block
46	CHAINID	2	.	`chain_id`		push current chain id onto stack
47	SELFBALANCE	5	.	`address(this).balance`		balance of executing contract, in wei
48	BASEFEE	2	.	`block.basefee`		base fee of current block
49-4F	*invalid*					
50	POP	2	`_anon`	.		remove item from top of stack and discard it
51	MLOAD	3*	`ost`	`mem[ost:ost+32]`		read word from memory at offset `ost`
52	MSTORE	3*	`ost, val`	.	mem[ost:ost+32] := val	write a word to memory
53	MSTORE8	3*	`ost, val`	.	mem[ost] := val && 0xFF	write a single byte to memory
54	SLOAD	A6	`key`	`storage[key]`		read word from storage
55	SSTORE	A7	`key, val`	.	storage[key] := val	write word to storage
56	JUMP	8	`dst`	.		`$pc := dst` mark that `pc` is only assigned if `dst` is a valid jumpdest
57	JUMPI	10	`dst, condition`	.		`$pc := condition ? dst : $pc + 1`
58	PC	2	.	`$pc`		program counter
59	MSIZE	2	.	`len(mem)`		size of memory in current execution context, in bytes
5A	GAS	2	.	`gasRemaining`		
5B	JUMPDEST	1			mark valid jump destination	a valid jump destination for example a jump destination not inside the push data
5C-5E	*invalid*					
5F	PUSH0	2	.	`uint8`		push the constant value 0 onto stack
60	PUSH1	3	.	`uint8`		push 1-byte value onto stack
61	PUSH2	3	.	`uint16`		push 2-byte value onto stack
62	PUSH3	3	.	`uint24`		push 3-byte value

onto stack | | 63 | PUSH4 | 3 | . | `uint32` | | push 4-byte value onto stack | | 64 | PUSH5 | 3 | . | `uint40` | | push 5-byte value onto stack | | 65 | PUSH6 | 3 | . | `uint48` | | push 6-byte value onto stack | | 66 | PUSH7 | 3 | . | `uint56` | | push 7-byte value onto stack | | 67 | PUSH8 | 3 | . | `uint64` | | push 8-byte value onto stack | | 68 | PUSH9 | 3 | . | `uint72` | | push 9-byte value onto stack | | 69 | PUSH10 | 3 | . | `uint80` | | push 10-byte value onto stack | | 6A | PUSH11 | 3 | . | `uint88` | | push 11-byte value onto stack | | 6B | PUSH12 | 3 | . | `uint96` | | push 12-byte value onto stack | | 6C | PUSH13 | 3 | . | `uint104` | | push 13-byte value onto stack | | 6D | PUSH14 | 3 | . | `uint112` | | push 14-byte value onto stack | | 6E | PUSH15 | 3 | . | `uint120` | | push 15-byte value onto stack | | 6F | PUSH16 | 3 | . | `uint128` | | push 16-byte value onto stack | | 70 | PUSH17 | 3 | . | `uint136` | | push 17-byte value onto stack | | 71 | PUSH18 | 3 | . | `uint144` | | push 18-byte value onto stack | | 72 | PUSH19 | 3 | . | `uint152` | | push 19-byte value onto stack | | 73 | PUSH20 | 3 | . | `uint160` | | push 20-byte value onto stack | | 74 | PUSH21 | 3 | . | `uint168` | | push 21-byte value onto stack | | 75 | PUSH22 | 3 | . | `uint176` | | push 22-byte value onto stack | | 76 | PUSH23 | 3 | . | `uint184` | | push 23-byte value onto stack | | 77 | PUSH24 | 3 | . | `uint192` | | push 24-byte value onto stack | | 78 | PUSH25 | 3 | . | `uint200` | | push 25-byte value onto stack | | 79 | PUSH26 | 3 | . | `uint208` | | push 26-byte value onto stack | | 7A | PUSH27 | 3 | . | `uint216` | | push 27-byte value onto stack | | 7B | PUSH28 | 3 | . | `uint224` | | push 28-byte value onto stack | | 7C | PUSH29 | 3 | . | `uint232` | | push 29-byte value onto stack | | 7D | PUSH30 | 3 | . | `uint240` | | push 30-byte value onto stack | | 7E | PUSH31 | 3 | . | `uint248` | | push 31-byte value onto stack | | 7F | PUSH32 | 3 | . | `uint256` | | push 32-byte value onto stack | | 80 | DUP1 | 3 | `a` | `a, a` | | clone 1st value on stack | | 81 | DUP2 | 3 | `_, a` | `a, _, a` | | clone 2nd value on stack | | 82 | DUP3 | 3 | `_, _, a` | `a, _, _, a` | | clone 3rd value on stack | | 83 | DUP4 | 3 | `_, _, _, a` | `a, _, _, _, a` | | clone 4th value on stack | | 84 | DUP5 | 3 | `..., a` | `a, ..., a` | | clone 5th value on stack | | 85 | DUP6 | 3 | `..., a` | `a, ..., a` | | clone 6th value on stack | | 86 | DUP7 | 3 | `..., a` | `a, ..., a` | | clone 7th value on stack | | 87 | DUP8 | 3 | `..., a` | `a, ..., a` | | clone 8th value on stack | | 88 | DUP9 | 3 | `..., a` | `a, ..., a` | | clone 9th value on stack | | 89 | DUP10 | 3 | `..., a` | `a, ..., a` | | clone 10th value on stack | | 8A | DUP11 | 3 | `..., a` | `a, ..., a` | | clone 11th value on stack | | 8B | DUP12 | 3 | `..., a` | `a, ..., a` | | clone 12th value on stack | | 8C | DUP13 | 3 | `..., a` | `a, ..., a` | | clone 13th value on stack | | 8D | DUP14 | 3 | `..., a` | `a, ..., a` | | clone 14th value on stack | | 8E | DUP15 | 3 | `..., a` | `a, ..., a` | | clone 15th value on stack | | 8F | DUP16 | 3 | `..., a` | `a, ..., a` | | clone 16th value on stack | | 90 | SWAP1 | 3 | `a, b` | `b, a` | | | 91 | SWAP2 | 3 | `a, _, b` | `b, _, a` | | | 92 | SWAP3 | 3 | `a, _, _, b` | `b, _, _, a` | | | 93 | SWAP4 | 3 | `a, _, _, _, b` | `b, _, _, _, a` | | | 94 | SWAP5 | 3 | `a, ..., b` `b, ..., a` | | | 95 | SWAP6 | 3 | `a, ..., b` | `b, ..., a` | | | 96 | SWAP7 | 3 | `a, ..., b` | `b, ..., a` | | | 97 | SWAP8 | 3 | `a, ..., b` | `b, ..., a` | | | 98 | SWAP9 | 3 | `a, ..., b` | `b, ..., a` | | | 99 | SWAP10 | 3 | `a, ..., b` | `b, ..., a` | | | 9A | SWAP11 | 3 | `a, ..., b` | `b, ..., a` | | | 9B | SWAP12 | 3 | `a, ..., b` | `b, ..., a` | | | 9C | SWAP13 | 3 | `a, ..., b` | `b, ..., a` | | | 9D | SWAP14 | 3 | `a, ..., b` | `b, ..., a` | | | 9E | SWAP15 | 3 | `a, ..., b` | `b, ..., a` | | | 9F | SWAP16 | 3 | `a, ..., b` | `b, ..., a` | | | A0 | LOG0 | A8 | `ost, len` | . | | LOG0(memory[ost:ost+len-1]) | | A1 | LOG1 | A8 | `ost, len, topic0` | . | | LOG1(memory[ost:ost+len-1], topic0) | | A2 | LOG2 | A8 | `ost, len, topic0, topic1` | . | | LOG1(memory[ost:ost+len-1], topic0, topic1) | | A3 | LOG3 | A8 | `ost, len, topic0, topic1, topic2` | . | | LOG1(memory[ost:ost+len-1], topic0, topic1, topic2) | | A4 | LOG4 | A8 | `ost, len, topic0, topic1, topic2, topic3` | . | | LOG1(memory[ost:ost+len-1], topic0, topic1, topic2, topic3) | | A5-EF | *invalid* | | F0 | CREATE | A9 | `val, ost, len` | `addr` | | addr = keccak256(rlp([address(this), this.nonce])) | | F1 | CALL | AA | `gas, addr, val, argOst, argLen, retOst, retLen` | `success` | mem[retOst:retOst+retLen-1] := returndata | | F2 | CALLCODE | AA | `gas, addr, val, argOst, argLen, retOst, retLen` | `success` | mem[retOst:retOst+retLen-1] = returndata | same as DELEGATECALL, but does not propagate original msg.sender and msg.value | | F3 | RETURN | 0* | `ost, len` | . | | return mem[ost:ost+len-1] | | F4 | DELEGATECALL | AA | `gas, addr, argOst, argLen, retOst, retLen` | `success` | mem[retOst:retOst+retLen-1] := returndata | | F5 | CREATE2 | A9 | `val, ost, len, salt` | `addr` | | addr = keccak256(0xff ++ address(this) ++ salt ++ keccak256(mem[ost:ost+len-1]))[12:] | | F6-F9 | *invalid* | | FA | STATICCALL | AA | `gas, addr, argOst, argLen, retOst, retLen` | `success` | mem[retOst:retOst+retLen-1] := returndata | | FB-FC | *invalid* | | FD | REVERT | 0* | `ost, len` | . | | revert(mem[ost:ost+len-1]) | | FE | INVALID | AF | | | designated invalid opcode - EIP-141 | | FF | SELFDESTRUCT | AB | `addr` | . | | | destroy contract and sends all funds to `addr` |