Instruction		Explanation
stop()	-	stop execution, identical to return(0, 0)
add(x, y)		x + y
sub(x, y)		x - y
mul(x, y)		x * y
div(x, y)		x / y or 0 if y == 0
sdiv(x, y)		x / y, for signed numbers in two’s complement, 0 if y == 0
mod(x, y)		x % y, 0 if y == 0
smod(x, y)		x % y, for signed numbers in two’s complement, 0 if y == 0
exp(x, y)		x to the power of y
not(x)		bitwise “not” of x (every bit of x is negated)
lt(x, y)		1 if x < y, 0 otherwise
gt(x, y)		1 if x > y, 0 otherwise
slt(x, y)		1 if x < y, 0 otherwise, for signed numbers in two’s complement
sgt(x, y)		1 if x > y, 0 otherwise, for signed numbers in two’s complement
eq(x, y)		1 if x == y, 0 otherwise
iszero(x)		1 if x == 0, 0 otherwise
and(x, y)		bitwise “and” of x and y
or(x, y)		bitwise “or” of x and y
xor(x, y)		bitwise “xor” of x and y
byte(n, x)		nth byte of x, where the most significant byte is the 0th byte
shl(x, y)		logical shift left y by x bits
shr(x, y)		logical shift right y by x bits

sar(x, y)		signed arithmetic shift right y by x bits
addmod(x, y, m)		(x + y) % m with arbitrary precision arithmetic, 0 if m == 0
mulmod(x, y, m)		(x * y) % m with arbitrary precision arithmetic, 0 if m == 0
signextend(i, x)		sign extend from (i*8+7)th bit counting from least significant
keccak256(p, n)		keccak(mem[p…(p+n)))
pc()		current position in code
pop(x)	-	discard value x
mload(p)		mem[p…(p+32))
mstore(p, v)	-	mem[p…(p+32)) := v
mstore8(p, v)	-	mem[p] := v & 0xff (only modifies a single byte)
sload(p)		storage[p]
sstore(p, v)	-	storage[p] := v
msize()		size of memory, i.e. largest accessed memory index
gas()		gas still available to execution
address()		address of the current contract / execution context
balance(a)		wei balance at address a
selfbalance()		equivalent to balance(address()), but cheaper
caller()		call sender (excluding `delegatecall`)
callvalue()		wei sent together with the current call
calldataload(p)		call data starting from position p (32 bytes)
calldatasize()		size of call data in bytes
calldatacopy(t, f, s)	-	copy s bytes from calldata at position f to mem at position t

codesize()		size of the code of the current contract / execution context
codecopy(t, f, s)	-	copy s bytes from code at position f to mem at position t
extcodesize(a)		size of the code at address a
extcodecopy(a, t, f, s)	-	like codecopy(t, f, s) but take code at address a
returndatasize()		size of the last returndata
returndatacopy(t, f, s)	-	copy s bytes from returndata at position f to mem at position t
extcodehash(a)		code hash of address a
create(v, p, n)		create new contract with code mem[p…(p+n)) and send v wei and return the new address; returns 0 on error
create2(v, p, n, s)		create new contract with code mem[p…(p+n)) at address keccak256(0xff . this . s . keccak256(mem[p…(p+n))) and send v wei and return the new address, where `0xff` is a 1 byte value, `this` is the current contract's address as a 20 byte value and `s` is a big-endian 256-bit value; returns 0 on error
call(g, a, v, in, insize, out, outsize)		call contract at address a with input mem[in…(in+insize)) providing g gas and v wei and output area mem[out…(out+outsize)) returning 0 on error (eg. out of gas) and 1 on success See more
callcode(g, a, v, in, insize, out, outsize)		identical to `call` but only use the code from a and stay in the context of the current contract otherwise See more
delegatecall(g, a, in, insize, out, outsize)		identical to `callcode` but also keep `caller` and `callvalue` See more
staticcall(g, a, in, insize, out, outsize)		identical to `call(g, a, 0, in, insize, out, outsize)` but do not allow state modifications See more
return(p, s)	-	end execution, return data mem[p…(p+s))
revert(p, s)	-	end execution, revert state changes, return data mem[p…(p+s))
selfdestruct(a)	-	end execution, destroy current contract and send funds to a
invalid()	-	end execution with invalid instruction

log0(p, s)	-	log without topics and data mem[p…(p+s))
log1(p, s, t1)	-	log with topic t1 and data mem[p…(p+s))
log2(p, s, t1, t2)	-	log with topics t1, t2 and data mem[p…(p+s))
log3(p, s, t1, t2, t3)	-	log with topics t1, t2, t3 and data mem[p…(p+s))
log4(p, s, t1, t2, t3, t4)	-	log with topics t1, t2, t3, t4 and data mem[p…(p+s))
chainid()		ID of the executing chain (EIP-1344)
basefee()		current block’s base fee (EIP-3198 and EIP-1559)
origin()		transaction sender
gasprice()		gas price of the transaction
blockhash(b)		hash of block nr b - only for last 256 blocks excluding current
coinbase()		current mining beneficiary
timestamp()		timestamp of the current block in seconds since the epoch
number()		current block number
difficulty()		difficulty of the current block
gaslimit()		block gas limit of the current block