Solidity yul opcodes cheat sheet		
Instruction		**Explanation**
stop()	F	stop execution, identical to return(0, 0)
add(x, y)	F	x + y
sub(x, y)	F	x - y
mul(x, y)	F	x * y
div(x, y)	F	x / y or 0 if y == 0
sdiv(x, y)	F	x / y, for signed numbers in two’s complement, 0 if y == 0
mod(x, y)	F	x % y, 0 if y == 0
smod(x, y)	F	x % y, for signed numbers in two’s complement, 0 if y == 0
exp(x, y)	F	x to the power of y
not(x)	F	bitwise « not » of x (every bit of x is negated)
lt(x, y)	F	1 if x < y, 0 otherwise
gt(x, y)	F	1 if x > y, 0 otherwise
slt(x, y)	F	1 if x < y, 0 otherwise, for signed numbers in two’s complement
sgt(x, y)	F	1 if x > y, 0 otherwise, for signed numbers in two’s complement
eq(x, y)	F	1 if x == y, 0 otherwise
iszero(x)	F	1 if x == 0, 0 otherwise
and(x, y)	F	bitwise « and » of x and y
or(x, y)	F	bitwise « or » of x and y
xor(x, y)	F	bitwise « xor » of x and y
byte(n, x)	F	nth byte of x, where the most significant byte is the 0th byte
shl(x, y)	C	logical shift left y by x bits
shr(x, y)	C	logical shift right y by x bits
sar(x, y)	C	signed arithmetic shift right y by x bits
addmod(x, y, m)	F	(x + y) % m with arbitrary precision arithmetic, 0 if m == 0
mulmod(x, y, m)	F	(x * y) % m with arbitrary precision arithmetic, 0 if m == 0

signextend(i, x)	F	sign extend from (i*8+7)th bit counting from least significant
keccak256(p, n)	F	keccak(mem[p…(p+n)))
pc()	F	current position in code
pop(x)	F	discard value x
mload(p)	F	mem[p…(p+32))
mstore(p, v)	F	mem[p…(p+32)) := v
mstore8(p, v)	F	mem[p] := v & 0xff (only modifies a single byte)
sload(p)	F	storage[p]
sstore(p, v)	F	storage[p] := v
msize()	F	size of memory, i.e. largest accessed memory index
gas()	F	gas still available to execution
address()	F	address of the current contract / execution context
balance(a)	F	wei balance at address a
selfbalance()	I	equivalent to balance(address()), but cheaper
caller()	F	call sender (excluding delegatecall)
callvalue()	F	wei sent together with the current call
calldataload(p)	F	call data starting from position p (32 bytes)
calldatasize()	F	size of call data in bytes
calldatacopy(t, f, s)	F	copy s bytes from calldata at position f to mem at position t
codesize()	F	size of the code of the current contract / execution context
codecopy(t, f, s)	F	copy s bytes from code at position f to mem at position t
extcodesize(a)	F	size of the code at address a
extcodecopy(a, t, f, s)	F	like codecopy(t, f, s) but take code at address a
returndatasize()	B	size of the last returndata
returndatacopy(t, f, s)	B	copy s bytes from returndata at position f to mem at position t
extcodehash(a)	C	code hash of address a

create(v, p, n)	F	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)	C	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)	F	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)	F	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)	H	identical to callcode but also keep caller and callvalue See more
staticcall (g, a, in, insize, out, outsize)	B	identical to call(g, a, 0, in, insize, out, outsize) but do not allow state modifications See more
return(p, s)	F	end execution, return data mem[p…(p+s))
revert(p, s)	B	end execution, revert state changes, return data mem[p…(p+s))
selfdestruct(a)	F	end execution, destroy current contract and send funds to a
invalid()	F	end execution with invalid instruction
log0(p, s)	F	log without topics and data mem[p…(p+s))
log1(p, s, t1)	F	log with topic t1 and data mem[p…(p+s))
log2(p, s, t1, t2)	F	log with topics t1, t2 and data mem[p…(p+s))
log3(p, s, t1, t2, t3)	F	log with topics t1, t2, t3 and data mem[p…(p+s))
log4(p, s, t1, t2, t3, t4)	F	log with topics t1, t2, t3, t4 and data mem[p…(p+s))
chainid()	I	ID of the executing chain (EIP-1344)
basefee()	L	current block's base fee (EIP-3198 and EIP-1559)
origin()	F	transaction sender
gasprice()	F	gas price of the transaction
blockhash(b)	F	hash of block nr b - only for last 256 blocks excluding current
coinbase()	F	current mining beneficiary
timestamp()	F	timestamp of the current block in seconds since the epoch
number()	F	current block number
difficulty()	F	difficulty of the current block
gaslimit()	F	block gas limit of the current block

swapX		Exchange 1st and Xth stack items, X entre 1 et 16
dupX		Duplicate Xth stack item, X entre 1 et 16
JUMPDEST		Valid destination for jumps
JUMP(x)		Jump to the byte offset specified on the stack
JUMPI(x, b)		Jump to the byte offset specified on the stack if b different of 0
example de stack : JUMPI(10, 1)	PUSH1 0 // Offest 0 PUSH1 10 // Offset 2 (previous instruction occupies 2 bytes) JUMPI  ne saute pas	