# Synthesizer: Opcode Reference

This document describes how the Tokamak [Synthesizer](synthesizer-terminology.md#synthesizer) handles each EVM opcode, comparing standard EVM behavior with circuit generation.

## On This Page

* [Overview](#overview)
* [All Opcodes](#all-opcodes)
* [Detailed Opcode Reference](#Xd6456d0ccb61a027748130d5ee04b13478b9225)
* [Circuit Complexity Summary](#circuit-complexity-summary)
* [Related Resources](#related-resources)
* [Appendix: Subcircuit Mapping Table](#appendix-subcircuit-mapping-table)

## Overview

### Standard EVM vs Synthesizer

// Standard EVM (Value Processing)  
0x01 ADD: pop(a, b) → push(a + b) // Black box  
  
// Synthesizer (Symbol Processing + Circuit Generation)  
0x01 ADD: pop(a, b) → place(ALU1, [selector, a, b], [result]) → push(result) // Transparent

### Key Differences

| Aspect | Standard EVM | Synthesizer |
| --- | --- | --- |
| **Processing** | Value-based computation | Symbol-based circuit generation |
| **Traceability** | Black box (input → output) | Transparent (input → placements → output) |
| **Output** | Final computation result | Circuit representation + result |
| **Purpose** | Execute transaction | Generate zk-SNARK proof |

### Subcircuit Types

The Synthesizer uses pre-compiled [subcircuits](synthesizer-terminology.md#subcircuit) from the [QAP Compiler](https://github.com/tokamak-network/Tokamak-zk-EVM/tree/main/packages/frontend/qap-compiler):

* **ALU1**: Basic arithmetic (ADD, MUL, SUB, EQ, ISZERO, NOT)
* **ALU2**: Modular arithmetic (DIV, SDIV, MOD, SMOD, ADDMOD, MULMOD)
* **ALU3**: Shift operations (SHL, SHR, SAR)
* **ALU4**: Comparisons (LT, GT, SLT, SGT)
* **ALU5**: Specialized operations (SIGNEXTEND, BYTE)
* **AND/OR/XOR**: Bitwise operations
* **DecToBit**: Decimal to bit decomposition
* **Accumulator**: Multi-input accumulation

## All Opcodes

| Opcode | Name | Stack In | Stack Out | Subcircuit | Details |
| --- | --- | --- | --- | --- | --- |
| 0x01 | ADD | a, b | a + b | ALU1 | [View Details →](#Xd6456d0ccb61a027748130d5ee04b13478b9225) |
| 0x02 | MUL | a, b | a × b | ALU1 | [View Details →](#X141e297fc6330a004b01b69d8edb2b3e4e6427e) |
| 0x03 | SUB | a, b | a - b | ALU1 | [View Details →](#X558aa0a86100aec78c50f24637d76e4cc823ae9) |
| 0x04 | DIV | a, b | a / b | ALU2 | [View Details →](#X285fdc4cb5eb17cf3b34587592dad066d3a64ed) |
| 0x0a | EXP | base, exp | base ^ exp | ALU1+DecToBit | [View Details →](#X1c45e952ecb6c96956cabbdd9af549b27620455) |
| 0x10 | LT | a, b | a < b | ALU4 | [View Details →](#Xc2bbb74a4009af0d71687728b57051ce0073066) |
| 0x11 | GT | a, b | a > b | ALU4 | [View Details →](#Xa32a4847ccf9a33fec9f4b600cfede023a3a9f8) |
| 0x12 | SLT | a, b | a < b (signed) | ALU4 | [View Details →](#Xec63758c833279d9a9a867823fce756ce79a5c6) |
| 0x13 | SGT | a, b | a > b (signed) | ALU4 | [View Details →](#X648605ada1b02ddd84fd2892ade7ba8cdee36a0) |
| 0x14 | EQ | a, b | a == b | ALU1 | [View Details →](#X716d275ecfe68f1804f6443b4f32aa94f66eeea) |
| 0x15 | ISZERO | a | a == 0 | ALU1 | [View Details →](#X25af60593f1d8a00ba4b27509368eda434003ad) |
| 0x16 | AND | a, b | a & b | AND | [View Details →](#X952694e27719ca1633cdc37a43243c83fcf4c57) |
| 0x17 | OR | a, b | a | b | OR | [View Details →](#X22765bd9edfc7501af0f06b9d1bd6df224f9d6c) |
| 0x18 | XOR | a, b | a ^ b | XOR | [View Details →](#X3232bf2802d29eb0be232c9a307323c49870f5c) |
| 0x19 | NOT | a | ~a | ALU1 | [View Details →](#Xd05fa7029ce0a7581bd318c015be55235272598) |
| 0x1a | BYTE | i, x | x[i] | ALU5 | [View Details →](#Xcb98345a1b8238c55b042b33d13e5f5178d5970) |
| 0x1b | SHL | shift, value | value << shift | ALU3 | [View Details →](#Xa8118b36c1394687471d4f9986b2f1b6c06d750) |
| 0x1c | SHR | shift, value | value >> shift | ALU3 | [View Details →](#X9d29d74d3ba1049b591c945725ccba0b082e630) |
| 0x1d | SAR | shift, value | value >> shift (signed) | ALU3 | [View Details →](#Xb5a7de3660cd7b9f47e914a87f062b84aa702a9) |
| 0x20 | KECCAK256 | offset, size | hash | External | [View Details →](#X9e3e4b40434e9783e2e14a94b794636b109c99b) |
| 0x30 | ADDRESS | - | address(this) | PUB\_IN | [View Details →](#X2e3123c41932fd6a2fe0b1686d1a8ae5b369e3b) |
| 0x35 | CALLDATALOAD | i | calldata[i] | PUB\_IN | [View Details →](#Xc57e6104a8216c2cb42ec40fbb153a674df7144) |
| 0x37 | CALLDATACOPY | memOff, dataOff, len | - | PUB\_IN | [View Details →](#X5689e52d73df5efe8014d3d24e1a271f0e6b69d) |
| 0x50 | POP | value | - | Stack | [View Details →](#X3f332bb10fdc8f786339b28e66914e3e850d3fd) |
| 0x51 | MLOAD | offset | memory[offset] | Memory | [View Details →](#X2f3c8eaf43fa57799f372cdd525392593ab7acc) |
| 0x52 | MSTORE | offset, value | - | Memory | [View Details →](#Xf7645b403dda8fcc68796ddeb084de818d2c94e) |
| 0x54 | SLOAD | key | storage[key] | PRV\_IN | [View Details →](#Xe448d2815a83ea7d23c26ebc3dfde9d77355f2d) |
| 0x55 | SSTORE | key, value | - | PRV\_OUT | [View Details →](#Xfce900b38a49dbbe1c7ea70a53258b551f440cc) |
| 0x60-0x7f | PUSH1-32 | - | value | Constant | [View Details →](#X608a45ea3bf8ec653f7e8550c6e210c01957f34) |
| 0x80-0x8f | DUP1-16 | … | value, … | Stack | [View Details →](#Xa5c9858a11c3c53aeaaa0fe1ece71f9dac5c0ac) |
| 0x90-0x9f | SWAP1-16 | a, …, b | b, …, a | Stack | [View Details →](#Xd9f0a324aac113a0ebd4c099d1ce042e7e25bc3) |

### 0x01: ADD

**Constraints**: 803 (entire ALU1 subcircuit)

**Stack**: a, b → a + b mod 2^256

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
const result = mod(a.value + b.value, TWO\_POW256);  
synthesizerArith('ADD', [a.value, b.value], result, runState);

#### Circuit Generation

* [**Subcircuit**](synthesizer-terminology.md#subcircuit): ALU1 | [**Selector**](synthesizer-terminology.md#selector): 1n << 1n
* **Constraints**: 803 (entire ALU1 subcircuit, 630 non-linear + 173 linear)

**Source**:

* Synthesizer: [functions.ts:97-103](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L97-L103) | [handlers.ts:18-26](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L18-L26)
* Circuit: [ALU1 (alu\_safe.circom:43-50)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L43-L50) | [Add256\_unsafe (arithmetic\_unsafe\_type1.circom:10-24)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/arithmetic_unsafe_type1.circom#L10-L24)

### 0x02: MUL

**Constraints**: 803 (entire ALU1 subcircuit)

**Stack**: a, b → a \* b mod 2^256

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
const result = mod(a.value \* b.value, TWO\_POW256);  
synthesizerArith('MUL', [a.value, b.value], result, runState);

#### Circuit Generation

* **Subcircuit**: ALU1 | **Selector**: 1n << 2n
* **Constraints**: 803 (entire ALU1 subcircuit, shared with ADD/SUB/EQ/ISZERO/NOT)

**Source**:

* Synthesizer: [functions.ts:105-111](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L105-L111) | [handlers.ts:28-36](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L28-L36)
* Circuit: [ALU1 (alu\_safe.circom:52-59)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L52-L59) | [Mul256\_unsafe (arithmetic\_unsafe\_type1.circom:37-66)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/arithmetic_unsafe_type1.circom#L37-L66)

### 0x03: SUB

**Constraints**: 803 (entire ALU1 subcircuit)

**Stack**: a, b → a - b mod 2^256

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
const result = mod(a.value - b.value, TWO\_POW256);  
synthesizerArith('SUB', [a.value, b.value], result, runState);

#### Circuit Generation

* **Subcircuit**: ALU1 | **Selector**: 1n << 3n
* **Constraints**: 803 (entire ALU1 subcircuit, shared with ADD/MUL/EQ/ISZERO/NOT)

**Source**:

* Synthesizer: [functions.ts:113-119](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L113-L119) | [handlers.ts:38-46](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L38-L46)
* Circuit: [ALU1 (alu\_safe.circom:61-68)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L61-L68) | [Sub256\_unsafe (arithmetic\_unsafe\_type1.circom:26-35)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/arithmetic_unsafe_type1.circom#L26-L35)

### 0x04: DIV

**Constraints**: 993 (entire ALU2 subcircuit)

**Stack**: a, b → a / b (integer division, 0 if b == 0)

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
let result = b.value === BIGINT\_0 ? BIGINT\_0 : mod(a.value / b.value, TWO\_POW256);  
synthesizerArith('DIV', [a.value, b.value], result, runState);

#### Circuit Generation

* **Subcircuit**: ALU2 (handles division/modulo) | **Selector**: 1n << 4n
* **Constraints**: 993 (entire ALU2 subcircuit, shared with SDIV/MOD/SMOD/ADDMOD/MULMOD)

**Special Cases**: Division by zero returns 0 (EVM convention)

**Source**:

* Synthesizer: [functions.ts:121-131](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L121-L131) | [handlers.ts:48-61](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L48-L61)
* Circuit: [ALU2 (alu\_safe.circom:154-162)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L154-L162) | [Div256\_unsafe (arithmetic\_unsafe\_type2.circom:16-46)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/arithmetic_unsafe_type2.circom#L16-L46)

### 0x0a: EXP

**Stack Input**: base, exponent  
**Stack Output**: base ^ exponent mod 2^256

#### Synthesizer Behavior

const [base, exponent] = stackPt.popN(2);  
let result;  
if (exponent.value === BIGINT\_0) {  
 result = BIGINT\_1;  
} else if (base.value === BIGINT\_0) {  
 result = base.value;  
} else {  
 result = (base.value \*\* exponent.value) % TWO\_POW256;  
}  
synthesizerArith('EXP', [base.value, exponent.value], result, runState);

#### Circuit Generation

EXP uses a two-phase approach implemented in [placeExp()](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/tokamak/operations/exp.ts#L12-L44):

**Phase 1: Binary Decomposition**

* **DecToBit**: Converts exponent to binary representation
* Line 28: synthesizer.placeArith('DecToBit', [bPt]).reverse()
* Constraints: 258 (256 non-linear + 2 linear)

**Phase 2: Square-and-Multiply Loop**

* **SubEXP**: Repeated squaring and conditional multiplication
* Lines 36-41: Loop through each bit of exponent
* Each iteration: synthesizer.placeArith('SubEXP', \_inPts)
* SubEXP per iteration: 803 constraints (entire ALU1 [subcircuit](synthesizer-terminology.md#subcircuit))
* Number of iterations: bit length of exponent (max 256)

**Why Two Subcircuits?**

// From exp.ts:26-41  
const k = Math.floor(Math.log2(bNum)) + 1; // Bit length  
  
// Step 1: Convert exponent to bits  
const bitifyOutPts = synthesizer.placeArith('DecToBit', [bPt]).reverse();  
  
// Step 2: Square-and-multiply using each bit  
for (let i = 1; i <= k; i++) {  
 const \_inPts = [chPts[i - 1], ahPts[i - 1], bitifyOutPts[i - 1]];  
 const \_outPts = synthesizer.placeArith('SubEXP', \_inPts);  
 chPts.push(\_outPts[0]); // Cumulative result  
 ahPts.push(\_outPts[1]); // Current power  
}

**Total Constraints**: 258 (DecToBit) + 803 × k (SubEXP iterations), where k = bit length of exponent

#### Example: Computing 3^13

13 = 0b1101 (binary, LSB first: [1, 0, 1, 1])  
  
Step 1: DecToBit(13) → [1, 0, 1, 1] (258 constraints)  
  
Step 2: Square-and-Multiply Loop (4 iterations, 803 constraints each)  
 Initial: ch[0] = 1, ah[0] = 3  
  
 i=1, bit[0]=1: SubEXP(ch=1, ah=3, bit=1)  
 → ch[1] = 3 (1 \* 3^1)  
 → ah[1] = 9 (3^2)  
  
 i=2, bit[1]=0: SubEXP(ch=3, ah=9, bit=0)  
 → ch[2] = 3 (3 \* 1, bit=0 means no multiply)  
 → ah[2] = 81 (9^2)  
  
 i=3, bit[2]=1: SubEXP(ch=3, ah=81, bit=1)  
 → ch[3] = 243 (3 \* 81^1)  
 → ah[3] = 6561 (81^2)  
  
 i=4, bit[3]=1: SubEXP(ch=243, ah=6561, bit=1)  
 → ch[4] = 1594323 (243 \* 6561^1)  
 → ah[4] = 43046721 (6561^2)  
  
Result: 3^13 = 1594323  
  
Total Constraints: 258 + (803 × 4) = 3,470 constraints

**Source**:

* Synthesizer: [functions.ts:177-188](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L177-L188) | [handlers.ts:141-156](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L141-L156) | [placeExp() (exp.ts:12-44)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/tokamak/operations/exp.ts#L12-L44)
* Circuit: [ALU1 SubEXP (alu\_safe.circom:70-78)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L70-L78) | [DecToBit\_circuit.circom](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/subcircuits/circom/DecToBit_circuit.circom)

### 0x10: LT

**Constraints**: 629 (entire ALU4 subcircuit)

**Stack**: a, b → a < b (1 if true, 0 if false)

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
const result = a.value < b.value ? BIGINT\_1 : BIGINT\_0;  
synthesizerArith('LT', [a.value, b.value], result, runState);

#### Circuit Generation

* **Subcircuit**: ALU4 (comparison operations) | **Selector**: 1n << 16n
* **Constraints**: 629 (entire ALU4 subcircuit, shared with GT/SLT/SGT)

**Source**:

* Synthesizer: [functions.ts:240-246](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L240-L246) | [handlers.ts:167-175](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L167-L175)
* Circuit: [ALU4 (alu\_safe.circom:350-354)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L350-L354) | [LessThan256 (compare\_safe.circom:6-19)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/compare_safe.circom#L6-L19)

### 0x11: GT

**Constraints**: 629 (entire ALU4 subcircuit)

**Stack**: a, b → a > b (1 if true, 0 if false)

#### Circuit Generation

* **Subcircuit**: ALU4 | **Selector**: 1n << 17n
* **Constraints**: 629 (entire ALU4 subcircuit, shared with LT/SLT/SGT)

**Source**:

* Synthesizer: [functions.ts:248-254](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L248-L254) | [handlers.ts:177-185](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L177-L185)
* Circuit: [ALU4 (alu\_safe.circom:356-359)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L356-L359) | [GreaterThan256 (compare\_safe.circom:35-39)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/compare_safe.circom#L35-L39)

### 0x12: SLT

**Constraints**: 629 (entire ALU4 subcircuit)

**Stack**: a, b → a < b (signed comparison, 1 if true, 0 if false)

#### Circuit Generation

* **Subcircuit**: ALU4 | **Selector**: 1n << 18n
* **Constraints**: 629 (entire ALU4 subcircuit, shared with LT/GT/SGT)

**Source**:

* Synthesizer: [functions.ts:256-262](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L256-L262) | [handlers.ts:187-195](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L187-L195)
* Circuit: [ALU4 (alu\_safe.circom:361-395)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L361-L395) | [SignedLessThan256 (compare\_safe.circom:41-74)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/compare_safe.circom#L41-L74)

### 0x13: SGT

**Constraints**: 629 (entire ALU4 subcircuit)

**Stack**: a, b → a > b (signed comparison, 1 if true, 0 if false)

#### Circuit Generation

* **Subcircuit**: ALU4 | **Selector**: 1n << 19n
* **Constraints**: 629 (entire ALU4 subcircuit, shared with LT/GT/SLT)

**Source**:

* Synthesizer: [functions.ts:264-270](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L264-L270) | [handlers.ts:197-205](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L197-L205)
* Circuit: [ALU4 (alu\_safe.circom:397-400)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L397-L400) | [SignedGreaterThan256 (compare\_safe.circom:76-80)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/compare_safe.circom#L76-L80)

### 0x14: EQ

**Constraints**: 803 (entire ALU1 subcircuit)

**Stack**: a, b → a == b (1 if true, 0 if false)

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
const result = a.value === b.value ? BIGINT\_1 : BIGINT\_0;  
await synthesizerArith('EQ', [a.value, b.value], result, runState);

#### Circuit Generation

* **Subcircuit**: ALU1
* **Selector**: 1n << 20n
* **Inputs**: [selector, a, b]
* **Outputs**: [result] (0 or 1)
* **Constraints**: 803 (same ALU1 subcircuit)

**Source**:

* Synthesizer: [functions.ts:272-278](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L272-L278) | [handlers.ts:207-215](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L207-L215)
* Circuit: [ALU1 (alu\_safe.circom:80-87)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L80-L87) | [IsEqual256 (compare\_safe.circom:92-99)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/compare_safe.circom#L92-L99)

### 0x15: ISZERO

**Constraints**: 803 (entire ALU1 subcircuit)

**Stack**: a → a == 0 (1 if true, 0 if false)

#### Circuit Generation

* **Subcircuit**: ALU1 | **Selector**: 1n << 21n
* **Constraints**: 803 (entire ALU1 subcircuit, shared with ADD/MUL/SUB/EQ/NOT)

**Source**:

* Synthesizer: [functions.ts:280-286](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L280-L286) | [handlers.ts:217-225](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L217-L225)
* Circuit: [ALU1 (alu\_safe.circom:89-95)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L89-L95) | [IsZero256 (compare\_safe.circom:82-90)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/compare_safe.circom#L82-L90)

### 0x16: AND

**Constraints**: 774

**Stack**: a, b → a & b (bitwise AND)

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
const result = a.value & b.value;  
await synthesizerArith('AND', [a.value, b.value], result, runState);

#### Circuit Generation

* **Subcircuit**: AND (dedicated bitwise circuit)
* **Constraints**: 774 (768 non-linear + 6 linear)

**Source**:

* Synthesizer: [functions.ts:288-294](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L288-L294) | [handlers.ts:227-234](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L227-L234)
* Circuit: [ALU\_bitwise (alu\_safe.circom:870-880)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L870-L880) | [And256 (bitwise\_safe.circom:21-26)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/bitwise_safe.circom#L21-L26)

### 0x17: OR

**Constraints**: 774

**Stack**: a, b → a | b (bitwise OR)

#### Circuit Generation

* **Subcircuit**: OR (dedicated bitwise circuit)
* **Constraints**: 774 (768 non-linear + 6 linear)

**Source**:

* Synthesizer: [functions.ts:296-302](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L296-L302) | [handlers.ts:237-244](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L237-L244)
* Circuit: [ALU\_bitwise (alu\_safe.circom:882-892)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L882-L892) | [Or256 (bitwise\_safe.circom:14-19)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/bitwise_safe.circom#L14-L19)

### 0x18: XOR

**Constraints**: 774

**Stack**: a, b → a ^ b (bitwise XOR)

#### Circuit Generation

* **Subcircuit**: XOR (dedicated bitwise circuit)
* **Constraints**: 774 (768 non-linear + 6 linear)

**Source**:

* Synthesizer: [functions.ts:304-310](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L304-L310) | [handlers.ts:247-254](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L247-L254)
* Circuit: [ALU\_bitwise (alu\_safe.circom:894-904)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L894-L904) | [Xor256 (bitwise\_safe.circom:7-12)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/bitwise_safe.circom#L7-L12)

### 0x19: NOT

**Constraints**: 803 (entire ALU1 subcircuit)

**Stack**: a → ~a (bitwise NOT)

#### Circuit Generation

* **Subcircuit**: ALU1 | **Selector**: 1n << 25n
* **Constraints**: 803 (entire ALU1 subcircuit, shared with ADD/MUL/SUB/EQ/ISZERO)

**Source**:

* Synthesizer: [functions.ts:312-318](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L312-L318) | [handlers.ts:257-265](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L257-L265)
* Circuit: [ALU1 (alu\_safe.circom:98-104)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L98-L104) | [Not256\_unsafe (arithmetic\_unsafe\_type1.circom:95-100)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/arithmetic_unsafe_type1.circom#L95-L100)

### 0x1a: BYTE

**Constraints**: 819 (entire ALU5 subcircuit)

**Stack**: i, x → x[i] (i-th byte of x, 0-indexed from left)

#### Circuit Generation

* **Subcircuit**: ALU5 | **Selector**: 1n << 26n
* **Constraints**: 819 (entire ALU5 subcircuit, shared with SIGNEXTEND)

**Source**:

* Synthesizer: [functions.ts:320-327](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L320-L327) | [handlers.ts:268-276](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L268-L276)
* Circuit: [ALU5 (alu\_safe.circom:444-453)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L444-L453)

### 0x1b: SHL

**Constraints**: 816 (entire ALU3 subcircuit)

**Stack**: shift, value → value << shift

#### Synthesizer Behavior

const [a, b] = stackPt.popN(2);  
const result = (b.value << a.value) & ((BigInt(1) << BigInt(256)) - BigInt(1));  
await synthesizerArith('SHL', [a.value, b.value], result, runState);

#### Circuit Generation

* **Subcircuit**: ALU3 (shift operations) | **Selector**: 1n << 27n
* **Constraints**: 816 (entire ALU3 subcircuit, shared with SHR/SAR)

**Source**:

* Synthesizer: [functions.ts:329-335](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L329-L335) | [handlers.ts:278-286](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L278-L286)
* Circuit: [ALU3 (alu\_safe.circom:266-276)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L266-L276)

### 0x1c: SHR

**Constraints**: 816 (entire ALU3 subcircuit)

**Stack**: shift, value → value >> shift (logical shift right)

#### Circuit Generation

* **Subcircuit**: ALU3 | **Selector**: 1n << 28n
* **Constraints**: 816 (entire ALU3 subcircuit, shared with SHL/SAR)

**Source**:

* Synthesizer: [functions.ts:337-343](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L337-L343) | [handlers.ts:288-296](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L288-L296)
* Circuit: [ALU3 (alu\_safe.circom:278-287)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L278-L287)

### 0x1d: SAR

**Constraints**: 816 (entire ALU3 subcircuit)

**Stack**: shift, value → value >> shift (arithmetic shift right, sign-extended)

#### Circuit Generation

* **Subcircuit**: ALU3 | **Selector**: 1n << 29n
* **Constraints**: 816 (entire ALU3 subcircuit, shared with SHL/SHR)

**Source**:

* Synthesizer: [functions.ts:345-351](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L345-L351) | [handlers.ts:298-306](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L298-L306)
* Circuit: [ALU3 (alu\_safe.circom:289-301)](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/qap-compiler/templates/256bit/alu_safe.circom#L289-L301)

### 0x20: KECCAK256

**Constraints**: ~5000

**Stack**: offset, size → keccak256(memory[offset:offset+size])

#### Synthesizer Behavior

const [offsetPt, lengthPt] = stackPt.popN(2);  
const offset = offsetPt.value;  
const length = lengthPt.value;  
  
if (length !== BIGINT\_0) {  
 // Load memory data as symbols (DataPts)  
 const nChunks = Math.ceil(lengthNum / 32);  
 const chunkDataPts = [];  
  
 for (let i = 0; i < nChunks; i++) {  
 const dataAliasInfos = memoryPt.getDataAlias(\_offset, \_length);  
 chunkDataPts[i] = synthesizer.placeMemoryToStack(dataAliasInfos);  
 }  
  
 // Hash is computed externally, but circuit tracks inputs  
 const result = stack.peek(1)[0]; // Get result from EVM execution  
 stackPt.push(synthesizer.loadAndStoreKeccak(chunkDataPts, result, length));  
}

#### Circuit Generation

* **Processing**: **External** (hash computed outside circuit)
* **Tracking**: Input data symbols recorded in circuit
* **Reason**: Keccak256 is too expensive to compute in-circuit (~100,000 constraints per hash)
* **Approach**:
  1. Track input symbols ([DataPts](synthesizer-terminology.md#datapt-data-point) from memory)
  2. Compute hash externally (standard Keccak256)
  3. Load result as [auxiliary input](synthesizer-terminology.md#auxiliary-input-auxin)
  4. Circuit verifies correct inputs were hashed (not the hash itself)

#### Why External?

In-circuit Keccak256: ~100,000 constraints per hash  
Large contract with 10 hashes: 1,000,000 constraints just for hashing  
Solution: Compute hash externally, verify inputs/outputs

**Source**:

* Synthesizer: [functions.ts:190-197](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L190-L197) | [handlers.ts:322-375](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L322-L375)
* Note: Keccak256 is computed externally, not in-circuit

### 0x30: ADDRESS

**Constraints**: ~100

**Stack**: - → address(this) (current contract address)

#### Synthesizer Behavior

await synthesizerEnvInf('ADDRESS', runState);

#### Circuit Generation

* [**Buffer**](synthesizer-terminology.md#buffer-placements): [PUB\_IN](synthesizer-terminology.md#pub-in-and-pub-out) (Placement 0)
* **Flow**:
  1. External value (contract address) → PUB\_IN buffer
  2. Buffer creates [DataPt](synthesizer-terminology.md#datapt-data-point) symbol
  3. Symbol pushed to [StackPt](synthesizer-terminology.md#stackpt)

**Source**:

* Synthesizer: [functions.ts:355-358](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L355-L358) | [handlers.ts:377-382](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L377-L382)
* Circuit: Buffer operation (PUB\_IN placement)

### 0x35: CALLDATALOAD

**Constraints**: ~100

**Stack**: i → calldata[i:i+32] (32 bytes, zero-padded if needed)

#### Synthesizer Behavior

const pos = stackPt.pop().value;  
await synthesizerEnvInf('CALLDATALOAD', runState, undefined, pos);

#### Circuit Generation

* **Buffer**: PUB\_IN (Placement 0) - Calldata is public
* **Processing**:
  1. Load calldata value at position
  2. Create DataPt via PUB\_IN buffer
  3. Push symbol to StackPt
* **Constraints**: ~100 (buffer operation)

**Source**:

* Synthesizer: [functions.ts:383-387](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L383-L387) | [handlers.ts:418-425](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L418-L425)
* Circuit: Buffer operation (PUB\_IN placement)

### 0x37: CALLDATACOPY

**Constraints**: ~100

**Stack**: destOffset, offset, size → - (copies calldata to memory)

#### Synthesizer Behavior

const [memOffset, dataOffset, dataLength] = stackPt.popN(3);  
  
if (dataLength.value !== BIGINT\_0) {  
 // Load calldata as DataPt symbols  
 const dataPt = synthesizer.loadEnvInf(  
 env.address.toString(),  
 'Calldata',  
 bytesToBigInt(data),  
 Number(dataOffset.value),  
 Number(dataLength.value)  
 );  
  
 // Write symbols to MemoryPt  
 memoryPt.write(  
 Number(memOffset.value),  
 Number(dataLength.value),  
 dataPt  
 );  
}

#### Circuit Generation

* **Type**: Memory Operation + Environmental Information
* **Processing**:
  1. Load calldata from PUB\_IN buffer → DataPt
  2. Write DataPt to [MemoryPt](synthesizer-terminology.md#memorypt) (tracking memory state)
  3. MemoryPt handles [data aliasing](synthesizer-terminology.md#data-aliasing) if memory overlaps
* **Constraints**: ~100 (buffer) + ~5,000 per memory word (if memory circuits needed)

#### Memory Aliasing Handling

If CALLDATACOPY overlaps with existing memory:

Example:  
1. CALLDATACOPY(0x00, 0x00, 64) // Write calldata to memory 0x00-0x40  
2. MSTORE(0x20, value) // Overwrite memory 0x20-0x40  
3. MLOAD(0x00) // Load memory 0x00-0x20  
  
MemoryPt tracks:  
- Calldata symbol at 0x00-0x20 (first 32 bytes)  
- MSTORE value at 0x20-0x40 (overwrites last 32 bytes of calldata)  
  
MLOAD generates circuit to reconstruct first 32 bytes from calldata symbol.

**Source**:

* Synthesizer: [functions.ts:389-405](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L389-L405) | [handlers.ts:435-496](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L435-L496)
* Circuit: Buffer operation + Memory management

### 0x40: BLOCKHASH

**Stack**: blockNumber → blockhash(blockNumber) (or 0 if invalid)

#### Circuit Generation

* **Type**: Block Information
* **Buffer**: PUB\_IN (Placement 0)
* **Processing**: Similar to environmental info

**Source**:

* Synthesizer: [handlers.ts:611-616](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L611-L616)
* Circuit: Buffer operation (PUB\_IN placement)

### 0x41: COINBASE

**Stack**: - → block.coinbase (miner address)

**Source**:

* Synthesizer: [handlers.ts:619-624](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L619-L624)
* Circuit: Buffer operation (PUB\_IN placement)

### 0x50: POP

**Constraints**: 0

**Stack**: value → - (removes top value)

#### Synthesizer Behavior

stackPt.pop(); // Remove top DataPt from symbol stack

#### Circuit Generation

* **Type**: Stack Operation
* **Processing**: No circuit placement (pure stack manipulation)
* **Constraints**: 0

### 0x51: MLOAD

**Constraints**: ~5000

**Stack**: offset → memory[offset:offset+32]

#### Synthesizer Behavior

const offset = stackPt.pop().value;  
  
// Query MemoryPt for data aliasing info  
const dataAliasInfos = memoryPt.getDataAlias(Number(offset), 32);  
  
if (dataAliasInfos.length > 0) {  
 // Generate circuit to reconstruct memory from symbols  
 const resultPt = synthesizer.placeMemoryToStack(dataAliasInfos);  
 stackPt.push(resultPt);  
} else {  
 // Memory uninitialized, load zero  
 stackPt.push(synthesizer.loadAuxin(BIGINT\_0));  
}

#### Circuit Generation

* **Type**: Memory Operation with data aliasing resolution
* **Subcircuits**: DecToBit, Accumulator, Bitwise (AND, OR)
* **Constraints**: ~5,000 per overlapping DataPt

#### Data Aliasing Example

// Overlapping memory writes:  
1. MSTORE(0x00, 0xAAAA...AAAA) // Write to 0x00-0x20  
2. MSTORE(0x10, 0xBBBB...BBBB) // Overlapping write to 0x10-0x30  
3. MLOAD(0x00) // Load 0x00-0x20  
  
// MLOAD must reconstruct:  
// Result = dataPt1[0x00-0x10] | dataPt2[0x10-0x20]  
  
// Circuit:  
result = (dataPt1 & mask1) | ((dataPt2 & mask2) << 128)

**Source**:

* Synthesizer: [functions.ts:476-481](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L476-L481) | [handlers.ts:632-648](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L632-L648)
* Circuit: Memory management with data aliasing (DecToBit + Accumulator + Bitwise)

### 0x52: MSTORE

**Constraints**: ~5000

**Stack**: offset, value → - (writes 32 bytes to memory)

#### Synthesizer Behavior

const [offset, value] = stackPt.popN(2);  
  
// Write DataPt symbol to MemoryPt with timestamp  
memoryPt.write(Number(offset.value), 32, value);

#### Circuit Generation

* **Type**: Memory Operation
* **Processing**: Record symbol in MemoryPt with timestamp
* **Constraints**: 0 (lazy - circuits generated on MLOAD)

#### How MemoryPt Tracks State

MemoryPt: Map<timestamp, { memOffset, containerSize, dataPt }>  
  
// Each MSTORE increments timestamp:  
MSTORE(0x00, dataPt1) → timestamp 0  
MSTORE(0x10, dataPt2) → timestamp 1  
// Later MLOAD queries for overlaps

**Source**:

* Synthesizer: [functions.ts:483-489](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L483-L489) | [handlers.ts:650-656](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L650-L656)
* Circuit: Memory tracking (lazy evaluation, circuits generated on MLOAD)

### 0x54: SLOAD

**Constraints**: ~100

**Stack**: key → storage[key]

#### Synthesizer Behavior

const keyPt = stackPt.pop();  
const value = await stateManager.getContractStorage(address, key);  
  
// Load storage value as private input  
const valuePt = synthesizer.loadPrvInf(  
 address.toString(),  
 'Storage',  
 bytesToBigInt(value),  
 bytesToBigInt(key)  
);  
  
stackPt.push(valuePt);

#### Circuit Generation

* **Buffer**: [PRV\_IN](synthesizer-terminology.md#prv-in-and-prv-out) (Placement 2) - Storage is private by default
* **Processing**:
  1. Read storage value from stateManager (external)
  2. Load value as DataPt via PRV\_IN buffer
  3. Push symbol to StackPt
* **Constraints**: ~100 (buffer operation)

#### Why Private?

Storage values are often sensitive (user balances, private state). The Synthesizer treats storage as private input, allowing users to prove execution without revealing storage contents.

**Source**:

* Synthesizer: [functions.ts:511-520](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L511-L520) | [handlers.ts:672-689](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L672-L689)
* Circuit: Buffer operation (PRV\_IN placement for private storage)

### 0x55: SSTORE

**Constraints**: ~100

**Stack**: key, value → - (writes to storage)

#### Synthesizer Behavior

const [keyPt, valuePt] = stackPt.popN(2);  
  
// Store to state manager (external)  
await stateManager.putContractStorage(address, key, valuePt.value);  
  
// Record in PRV\_OUT buffer  
synthesizer.storePrvOut(address.toString(), 'Storage', valuePt, key);

#### Circuit Generation

* **Buffer**: [PRV\_OUT](synthesizer-terminology.md#prv-in-and-prv-out) (Placement 3) - Private outputs
* **Processing**:
  1. Write DataPt symbol to PRV\_OUT buffer
  2. Actual storage update happens externally
  3. Circuit tracks which symbols were stored
* **Constraints**: ~100 (buffer operation)

**Source**:

* Synthesizer: [functions.ts:522-548](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L522-L548) | [handlers.ts:691-710](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L691-L710)
* Circuit: Buffer operation (PRV\_OUT placement for private storage updates)

### 0x60-0x7f: PUSH1-PUSH32

**Constraints**: 0

**Stack**: - → value (pushes 1-32 bytes from bytecode)

#### Synthesizer Behavior

const numToPush = opcode - 0x5f;  
const value = bytesToBigInt(code.subarray(pc + 1, pc + 1 + numToPush));  
  
// Load hardcoded value as auxiliary input  
const valuePt = synthesizer.loadAuxin(value, numToPush);  
stackPt.push(valuePt);

#### Why loadAuxin?

PUSH values are hardcoded in bytecode (not from environment/storage), so they’re treated as [auxiliary inputs](synthesizer-terminology.md#auxiliary-input-auxin):

PUSH1 0x05 → synthesizer.loadAuxin(5, 1)  
PUSH32 0xFFFF... → synthesizer.loadAuxin(0xFFFF..., 32)

#### Circuit Generation

* **Type**: Hardcoded constant
* **Constraints**: 0 (constants don’t need circuits)

**Source**:

* Synthesizer: [functions.ts:572-579](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L572-L579) | [handlers.ts:726-733](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L726-L733)
* Circuit: Auxiliary input (hardcoded constants, no circuit needed)

### 0x80-0x8f: DUP1-DUP16

**Constraints**: 0

**Stack**: ..., value\_n, ... → ..., value\_n, ..., value\_n (duplicates n-th value from top)

#### Synthesizer Behavior

const n = opcode - 0x7f; // 0x80 → 1, 0x8f → 16  
stackPt.dup(n); // Duplicate DataPt reference

#### Circuit Generation

* **Type**: Stack manipulation
* **Constraints**: 0 (no circuit needed)

### 0x90-0x9f: SWAP1-SWAP16

**Constraints**: 0

**Stack**: ..., value\_n+1, ..., value\_0 → ..., value\_0, ..., value\_n+1 (swaps top with n+1-th value)

#### Synthesizer Behavior

const n = opcode - 0x8f; // 0x90 → 1, 0x9f → 16  
stackPt.swap(n); // Swap DataPt references

#### Circuit Generation

* **Type**: Stack manipulation
* **Constraints**: 0 (no circuit needed)

### 0x56: JUMP

**Stack**: counter → - (jumps to destination PC)

#### Synthesizer Behavior

const destPt = stackPt.pop();  
// PC change handled by interpreter  
// No circuit generation (control flow)

#### Circuit Generation

* **Type**: Control Flow
* **Processing**: No circuit (PC manipulation)
* **Constraints**: 0

**Source**:

* Synthesizer: [functions.ts:550-555](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/functions.ts#L550-L555) | [handlers.ts:713-723](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L713-L723)
* Circuit: Control flow (no circuit needed)

### 0xf0: CREATE

**Stack**: value, offset, size → address (new contract address, or 0 if failed)

#### Status

❌ **Not Supported**

#### Reason

The Tokamak zk-EVM is designed specifically for Layer 2 state channel applications, where contract creation is not required. Contract creation requires:

* Context switching circuits (~10,000 constraints)
* Deployment code execution tracking
* Address computation verification

**Note**: There are no plans to support this opcode, as it is outside the scope of state channel use cases.

### 0xf3: RETURN

**Stack**: offset, size → - (halts execution, returns memory[offset:offset+size])

#### Synthesizer Behavior

const [offsetPt, lengthPt] = stackPt.popN(2);  
  
if (lengthPt.value !== BIGINT\_0) {  
 // Load return data from memory as symbols  
 const offset = Number(offsetPt.value);  
 const length = Number(lengthPt.value);  
  
 const dataAliasInfos = memoryPt.getDataAlias(offset, length);  
  
 // Generate circuits to reconstruct return data  
 for (const info of dataAliasInfos) {  
 const dataPt = synthesizer.placeMemoryToStack([info]);  
 synthesizer.storePubOut(dataPt); // Write to PUB\_OUT buffer  
 }  
}

#### Circuit Generation

* **Type**: System Operation
* **Buffer**: [PUB\_OUT](synthesizer-terminology.md#pub-in-and-pub-out) (Placement 1) - Return data is public
* **Processing**:
  1. Load return data from MemoryPt (with aliasing resolution)
  2. Generate circuits to reconstruct data
  3. Write symbols to PUB\_OUT buffer
* **Constraints**: ~5,000 per memory segment

**Source**:

* Synthesizer: [handlers.ts:845-868](https://github.com/tokamak-network/Tokamak-zk-EVM/blob/main/packages/frontend/synthesizer/src/opcodes/synthesizer/handlers.ts#L845-L868)
* Circuit: Buffer operation (PUB\_OUT placement) + Memory reconstruction

### 0xfd: REVERT

**Opcode**: 0xfd  
**Mnemonic**: REVERT

#### Status

❌ **Not Supported**

#### Reason

The Tokamak zk-EVM is designed specifically for Layer 2 state channel applications, where revert handling is not required. Revert handling requires:

* State rollback tracking
* Error data propagation
* Gas refund calculations

**Note**: There are no plans to support this opcode, as it is outside the scope of state channel use cases.

### 0xff: SELFDESTRUCT

**Opcode**: 0xff  
**Mnemonic**: SELFDESTRUCT

#### Status

❌ **Not Supported**

#### Reason

The Tokamak zk-EVM is designed specifically for Layer 2 state channel applications, where self-destruct is not required. Self-destruct requires:

* Balance transfer verification
* Contract deletion tracking
* Beneficiary updates

**Note**: There are no plans to support this opcode, as it is outside the scope of state channel use cases.

## Circuit Complexity Summary

### Constraint Counts by Operation Type

| Operation Type | Constraints (from QAP Compiler) | Example Opcodes |
| --- | --- | --- |
| **ALU1 Operations** | 803 | ADD, SUB, MUL, EQ, ISZERO |
| **ALU2 Operations** | 993 | DIV, MOD, SDIV, SMOD, ADDMOD |
| **ALU3 Operations** | 816 | SHL, SHR, SAR |
| **ALU4 Operations** | 629 | LT, GT, SLT, SGT |
| **ALU5 Operations** | 819 | SIGNEXTEND, BYTE |
| **Bitwise Operations** | 774 | AND, OR, XOR |
| **Bit Decomposition** | 258 | DecToBit (used in EXP) |
| **Exponentiation** | 258 + (803 × bit\_length) | EXP (e.g., 3^13 = 3,470) |
| **Memory Operations** | Variable | MLOAD, MSTORE (with aliasing) |
| **External Operations** | 0 (processed outside) | KECCAK256 |

## Appendix: Subcircuit Mapping Table

| Opcode | Operation | Subcircuit | Selector | Constraints | Non-linear | Linear |
| --- | --- | --- | --- | --- | --- | --- |
| 0x01 | ADD | ALU1 | 1 << 1 | 803 | 630 | 173 |
| 0x02 | MUL | ALU1 | 1 << 2 | 803 | 630 | 173 |
| 0x03 | SUB | ALU1 | 1 << 3 | 803 | 630 | 173 |
| 0x04 | DIV | ALU2 | 1 << 4 | 993 | 566 | 427 |
| 0x05 | SDIV | ALU2 | 1 << 5 | 993 | 566 | 427 |
| 0x06 | MOD | ALU2 | 1 << 6 | 993 | 566 | 427 |
| 0x07 | SMOD | ALU2 | 1 << 7 | 993 | 566 | 427 |
| 0x08 | ADDMOD | ALU2 | 1 << 8 | 993 | 566 | 427 |
| 0x09 | MULMOD | ALU2 | 1 << 9 | 993 | 566 | 427 |
| 0x0a | EXP | ALU1 (SubEXP) | 1 << 10 | ~206,000 | - | - |
| 0x0b | SIGNEXTEND | ALU5 | 1 << 11 | 819 | 640 | 179 |
| 0x10 | LT | ALU4 | 1 << 16 | 629 | 594 | 35 |
| 0x11 | GT | ALU4 | 1 << 17 | 629 | 594 | 35 |
| 0x12 | SLT | ALU4 | 1 << 18 | 629 | 594 | 35 |
| 0x13 | SGT | ALU4 | 1 << 19 | 629 | 594 | 35 |
| 0x14 | EQ | ALU1 | 1 << 20 | 803 | 630 | 173 |
| 0x15 | ISZERO | ALU1 | 1 << 21 | 803 | 630 | 173 |
| 0x16 | AND | AND | - | 774 | 768 | 6 |
| 0x17 | OR | OR | - | 774 | 768 | 6 |
| 0x18 | XOR | XOR | - | 774 | 768 | 6 |
| 0x19 | NOT | ALU1 | 1 << 25 | 803 | 630 | 173 |
| 0x1a | BYTE | ALU5 | 1 << 26 | 819 | 640 | 179 |
| 0x1b | SHL | ALU3 | 1 << 27 | 816 | 638 | 178 |
| 0x1c | SHR | ALU3 | 1 << 28 | 816 | 638 | 178 |
| 0x1d | SAR | ALU3 | 1 << 29 | 816 | 638 | 178 |

*Source:* [*packages/frontend/qap-compiler/subcircuits/library/info/*](https://github.com/tokamak-network/Tokamak-zk-EVM/tree/main/packages/frontend/qap-compiler/subcircuits/library/info) *- Compiled subcircuit constraint information from Circom circuits*