Calldata:

0x6356ce4f

0000000000000000000000000000000000000000000000000000000000000040 0x00

00000000000000000000000000000000000000000000000000000000000002c0 0x20

0000000000000000000000000000000000000000000000000000000000000244 0x40 <- this could go up to 0x260

6e6ecf9600000000000000000000000000000000000000000000000000000000 0x60

0000002000000000000000000000000000000000000000000000000000000000 0x80

0000001100000000000000000000000000000000000000000000000000028e44 0xa0

ea45085600000000000000000000000000000000000000000000000000000000 0xc0

0000058000000000000000000000000000000000000000000000000000028e44 0xe0

ea450d4c00000000000000000000000000000000000000000000000000000000 0x100

0000144e00000000000000000000000000000000000000000000000000028e44 0x120

ea45598500000000000000000000000000000000000000000000000000000000 0x140

00000b0800000000000000000000000000000000000000000000000000028e44 0x160

ea45589400000000000000000000000000000000000000000000000000000000 0x180

0000095200000000000000000000000000000000000000000000000000028e44 0x1a0

ea45d43600000000000000000000000000000000000000000000000000000000 0x1c0

000036a600000000000000000000000000000000000000000000000000028e44 0x1e0

ea45c69f00000000000000000000000000000000000000000000000000000000 0x200

00003ee600000000000000000000000000000000000000000000000000028e44 0x220

ea450a9600000000000000000000000000000000000000000000000000000000 0x240

00007a6600000000000000000000000000000000000000000000000000028e44 0x260

ea450de400000000000000000000000000000000000000000000000000028e44 0x280

ea45087700000000000000000000000000000000000000000000000000000000 0x2a0

000000000000000000000000000000000000000000000000000000000000000d 0x2c0

616c6d6f73742074686572652100000000000000000000000000000000000000 0x2e0

0x6356ce4f000000000000000000000000000000000000000000000000000000000000004000000000000000000000000000000000000000000000000000000000000002c000000000000000000000000000000000000000000000000000000000000002446e6ecf960000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000001100000000000000000000000000000000000000000000000000028e44ea450856000000000000000000000000000000000000000000000000000000000000058000000000000000000000000000000000000000000000000000028e44ea450d4c000000000000000000000000000000000000000000000000000000000000144e00000000000000000000000000000000000000000000000000028e44ea4559850000000000000000000000000000000000000000000000000000000000000b0800000000000000000000000000000000000000000000000000028e44ea455894000000000000000000000000000000000000000000000000000000000000095200000000000000000000000000000000000000000000000000028e44ea45d43600000000000000000000000000000000000000000000000000000000000036a600000000000000000000000000000000000000000000000000028e44ea45c69f0000000000000000000000000000000000000000000000000000000000003ee600000000000000000000000000000000000000000000000000028e44ea450a960000000000000000000000000000000000000000000000000000000000007a6600000000000000000000000000000000000000000000000000028e44ea450de400000000000000000000000000000000000000000000000000028e44ea45087700000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000d616c6d6f73742074686572652100000000000000000000000000000000000000



object "contract" {

code { }

object "runtime" {

code {

mstore(0x40, 0x80) // initialization of the fmp stored at 0x40, 0x80

let \_0 := iszero(callvalue())

require(not(\_0))

let \_1 := lt(calldatasize(), 0x4)

if not(\_1){

let \_2 := shr(0xe0, calldataload(0x0))

let \_3 := eq(0x6356ce4f, \_2)

switch \_2

case 0x6356ce4f{

func\_0x6356ce4f()

}

default { }

}

func\_ROOT4146650865()

function func\_0x6356ce4f() {

let \_4 := add(0x4, sub(calldatasize(), 0x4))

let \_5 := iszero(slt(sub(\_4, 0x4), 0x40))

require(not(\_5))

let \_6 := calldataload(add(0x4, 0x0)) // first pointer calldata[4]. Eg: 0x40

let \_7 := iszero(gt(\_6, 0xffffffffffffffff))

require(not(\_7))

let \_8 := add(0x4, \_6) // location of the length of the first bytes calldata[4] + 4. Eg: 0x44

let \_9 := slt(add(\_8, 0x1f), \_4)

require(not(\_9))

let \_10 := calldataload(\_8) // length of the first bytes Eg: 5

let \_11 := add(\_8, 0x20)

let \_12 := iszero(gt(\_10, 0xffffffffffffffff))

if \_12{

let \_13 := mload(0x40) // 0x80

let \_14 := add(\_13, and(add(add(and(add(\_10, 0x1f), not(0x1f)), 0x20), 0x1f), not(0x1f)))

let \_15 := iszero(or(gt(\_14, 0xffffffffffffffff), lt(\_14, \_13)))

if \_15{

mstore(0x40, \_14) // updated the fmp with the first bytes saved Eg: stored at 0x40 -> 0xc0

mstore(\_13, \_10) // stored at 0x80 th length of the first bytes Eg: stored at 0x80 -> 5

let \_16 := add(\_13, 0x20)

let \_17 := iszero(gt(add(\_11, \_10), \_4))

require(not(\_17))

calldatacopy(\_16, \_11, \_10) // stored the first bytes at position 0xa0

mstore(add(\_16, \_10), 0x0)

let \_18 := calldataload(add(0x4, 0x20))

let \_19 := iszero(gt(\_18, 0xffffffffffffffff))

require(not(\_19))

let \_20 := add(0x4, \_18)

let \_21 := slt(add(\_20, 0x1f), \_4)

require(not(\_21))

let \_22 := calldataload(\_20)

let \_23 := add(\_20, 0x20)

let \_24 := iszero(gt(\_22, 0xffffffffffffffff))

if \_24{

let \_25 := mload(0x40) // 0xc0

let \_26 := add(\_25, and(add(add(and(add(\_22, 0x1f), not(0x1f)), 0x20), 0x1f), not(0x1f)))

let \_27 := iszero(or(gt(\_26, 0xffffffffffffffff), lt(\_26, \_25)))

if \_27{

mstore(0x40, \_26) // updated the fmp with the second bytes saved Eg: stored at 0x40 -> 0x100

mstore(\_25, \_22) // stored at 0xc0 the length of the second bytes Eg: stored at 0xc0 -> 0xd

let \_28 := add(\_25, 0x20)

let \_29 := iszero(gt(add(\_23, \_22), \_4))

require(not(\_29))

calldatacopy(\_28, \_23, \_22) // stored the first bytes at position 0xe0

mstore(add(\_28, \_22), 0x0)

// Here starts the actual implementation of the function, all other code are just checks for the calldata parameters (bytes memory var1, bytes memory var2)

// 0000000000000000000000000000000000000000000000000000000 0x00

// 0000000000000000000000000000000000000000000000000000000 0x20

// 0000000000000000000000000000000000000000000000000000100 0x40 <- free memory pointer

// 0000000000000000000000000000000000000000000000000000000 0x60

// 0000000000000000000000000000000000000000000000000000005 0x80 <- length bytes1

// 0000000000000000000000000000000000000000000000102030405 0xa0 <- bytes1

// 0000000000000000000000000000000000000000000000000000005 0xc0 <- length bytes2

// 000000000000000000000000000000000000000000000060708090a 0xe0 <- bytes2

// 0000000000000000000000000000000000000000000000000000000 0x100

let \_30 := mload(0x40) // 0x100

mstore(0x40, add(0x40, \_30)) // update the fmp to allocate 2 more words Eg: stored at 0x40 -> 0x140

mstore(\_30, 0xd) // stored the length of "almost there!" 0xd at 0x100 Eg: stored at 0x100 -> 0xd

mstore(add(0x20, \_30), 0x616c6d6f73742074686572652100000000000000000000000000000000000000) // Eg: stored at 0x120 -> "almost there!"

// 0000000000000000000000000000000000000000000000000000000 0x00

// 0000000000000000000000000000000000000000000000000000000 0x20

// 0000000000000000000000000000000000000000000000000000140 0x40 <- free memory pointer

// 0000000000000000000000000000000000000000000000000000000 0x60

// 0000000000000000000000000000000000000000000000000000005 0x80 <- length bytes1

// 0000000000000000000000000000000000000000000000102030405 0xa0 <- bytes1

// 0000000000000000000000000000000000000000000000000000005 0xc0 <- length bytes2

// 000000000000000000000000000000000000000000000060708090a 0xe0 <- bytes2

// 000000000000000000000000000000000000000000000000000000d 0x100 <- length of "almost there!"

// 616c6d6f73742074686572652100000000000000000000000000000 0x120 <- "almost there!"

// 0000000000000000000000000000000000000000000000000000000 0x140

let \_31 := mload(add(\_25, 0x20)) // second bytes data -> calldata[0xe0]

let \_32 := mload(add(\_30, 0x20)) // "almost there!" -> calldata[0x120]

let \_33 := eq(\_32, \_31) // second bytes needs to be "almost there!"

require(not(\_33))

let \_34 := mload(0x40) // 0x140

mstore(0x40, add(0x720, \_34)) // update the fmp for more 0x720 bytes Eg: stored at 0x40 -> 0x860

mstore(\_34, 0x6ed) // store at 0x140 the length -> 0x6ed Eg: stored at 0x140-> 0x6ed

codecopy(add(0x20, \_34), 0x475, 0x6ed) // store at 0x160 the copied code -> 

// 0000000000000000000000000000000000000000000000000000000 0x00

// 0000000000000000000000000000000000000000000000000000000 0x20

// 0000000000000000000000000000000000000000000000000000860 0x40 <- free memory pointer

// 0000000000000000000000000000000000000000000000000000000 0x60

// 0000000000000000000000000000000000000000000000000000005 0x80 <- length bytes1

// 0000000000000000000000000000000000000000000000102030405 0xa0 <- bytes1

// 0000000000000000000000000000000000000000000000000000005 0xc0 <- length bytes2

// 000000000000000000000000000000000000000000000060708090a 0xe0 <- bytes2

// 000000000000000000000000000000000000000000000000000000d 0x100 <- length of "almost there!"

// 616c6d6f73742074686572652100000000000000000000000000000 0x120 <- "almost there!"

// 00000000000000000000000000000000000000000000000000006ed 0x140 <- length of copied code

// 6106c7610011610000396106c7610000f360003560e01c6002600c8 0x160 <- copied code

// ...

// 80399001a0637841906c781181800a16576797065728300030a0015 0x840

// 00000000000000000000000000000000000000000000000000006ed 0x860

let \_35 := mload(\_34) // 0x6ed

let \_36 := create(0x0, add(\_34, 0x20), \_35) // address of the contract created

let \_37 := mload(0x40) // 0x860

let \_38 := mload(\_13) // length bytes1

let \_39 := 0x0 // num1 = 0

let \_40 := 0x0 // num2 = 0

for {

let \_41 := lt(\_39, \_38) // condition1 num1 < 0x80 initial value true

let \_42 := 0x20 // num3 = 0x20

let \_43 := add(\_40, \_42) // num4 = num2 + num3 initial value 0x20

}

not(iszero(\_41)) // num1 >= 0x80 to leave the loop

{ } // No incremental actions

{

\_39 := \_43 // num1 = num4

\_41 := lt(\_39, \_38) //

let \_44 := not(iszero(\_41)) // leave the loop with num1 >= 0x80

\_40 := \_34 // num2 = 0x140

let \_45 := mload(add(add(\_13, 0x20), \_40)) // num5 = 0x80 + 0x20 + num2 = 0x1e0

mstore(add(\_37, \_40), \_45) // store at memory position 0x860 + 0x140 = 0x9a0 -> 0x1e0

\_42 := 0x20 // num3 = 0x20

\_43 := add(\_40, \_42) // num4 = num2 + num3

\_39 := \_43 // num1 = num4

\_40 := \_43 // num2 = num4

}

mstore(add(\_37, \_38), 0x0) // store at 0x860 + 0x80 = 0x8e0 the 0 value

let \_46 := mload(0x40) // 0x860

let \_47 := call(gas(), and(0xffffffffffffffffffffffffffffffffffffffff, \_36), 0x0, \_46, sub(add(\_37, \_38), \_46), \_46, 0x0)

// The calldata starts at the current fmp and loads (\_37 + \_38 - \_46)

// Current fmp -> 0x860

// Size -> 0x860 + length bytes1 - 0x860 = 0x100 should be 580 or 0x244 it is already okay

let \_48 := returndatasize()

let \_49 := eq(\_48, 0x0)

switch \_48

case 0x0{ }

default {

let \_50 := mload(0x40)

mstore(0x40, add(\_50, and(add(returndatasize(), 0x3f), not(0x1f))))

mstore(\_50, returndatasize())

returndatacopy(add(\_50, 0x20), 0x0, returndatasize())

}

if \_47{

stop()

}

if not(\_47){

mstore(0x0, 0x4e487b7100000000000000000000000000000000000000000000000000000000)

mstore(0x4, 0x1)

revert(0x0, 0x24)

}

// End of the implementation

}

if not(\_27){

mstore(0x0, 0x4e487b7100000000000000000000000000000000000000000000000000000000)

mstore(0x4, 0x41)

revert(0x0, 0x24)

}

}

if not(\_24){

mstore(0x0, 0x4e487b7100000000000000000000000000000000000000000000000000000000)

mstore(0x4, 0x41)

revert(0x0, 0x24)

}

}

if not(\_15){

mstore(0x0, 0x4e487b7100000000000000000000000000000000000000000000000000000000)

mstore(0x4, 0x41)

revert(0x0, 0x24)

}

}

if not(\_12){

mstore(0x0, 0x4e487b7100000000000000000000000000000000000000000000000000000000)

mstore(0x4, 0x41)

revert(0x0, 0x24)

}

}

function func\_ROOT4146650865() {

revert(0x0, 0x0)

}

function require(condition) {

if iszero(condition){

revert(0x0, 0x0)

}

}

}

}

}

[00] PUSH1 80 // [0x80]

[02] PUSH1 40 // [0x40, 0x80]

[04] MSTORE // Stored 0x80 at position 0x40

[05] CALLVALUE // [msg.value]

[06] DUP1 // [msg.value, msg.value]

[07] ISZERO // [msg.value == 0, msg.value]

[08] PUSH2 0010 // [0x10, msg.value == 0, msg.value]

[0b] JUMPI // [msg.value]

// msg.value needs to be zero

[10] JUMPDEST

[11] POP // []

[12] PUSH1 04 // [0x4]

[14] CALLDATASIZE // [calldata.length, 0x4]

[15] LT // [calldata.length < 0x4]

[16] PUSH2 002b // [0x2b, calldata.length < 0x4]

[19] JUMPI // []

// calldata.length needs to be greater or equal to 4 bytes

[1a] PUSH1 00 // [0]

[1c] CALLDATALOAD // [calldata[0:32]]

[1d] PUSH1 e0 // [0xe0, calldata[0:32]]

[1f] SHR // [selector]

[20] DUP1 // [selector, selector]

[21] PUSH4 6356ce4f // [6356ce4f, selector, selector]

[26] EQ // [6356ce4f == selector, selector]

[27] PUSH2 0030 // [0x30, 6356ce4f == selector, selector]

[2a] JUMPI // [selector]

// selector needs to be 6356ce4f

[30] JUMPDEST

[31] PUSH2 004a // [0x4a, selector]

[34] PUSH1 04 // [4, 0x4a, selector]

[36] DUP1 // [4, 4, 0x4a, selector]

[37] CALLDATASIZE // [calldata.length, 4, 4, 0x4a, selector]

[38] SUB // [calldata.length - 4, 4, 0x4a, selector]

[39] DUP2 // [4, calldata.length - 4, 4, 0x4a, selector]

[3a] ADD // [calldata.length, 4, 0x4a, selector]

[3b] SWAP1 // [4, calldata.length, 0x4a, selector]

[3c] PUSH2 0045 // [0x45, 4, calldata.length, 0x4a, selector]

[3f] SWAP2 // [calldata.length, 4, 0x45, 0x4a, selector]

[40] SWAP1 // [4, calldata.length, 0x45, 0x4a, selector]

[41] PUSH2 0345 // [0x345, 4, calldata.length, 0x45, 0x4a, selector]

[44] JUMP // [4, calldata.length, 0x45, 0x4a, selector]

[345] JUMPDEST

[346] PUSH1 00 // [0, 4, calldata.length, 0x45, 0x4a, selector]

[348] DUP1 // [0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[349] PUSH1 40 // [0x40, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[34b] DUP4 // [4, 0x40, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[34c] DUP6 // [calldata.length, 4, 0x40, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[34d] SUB // [calldata.length - 4, 0x40, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[34e] SLT // [calldata.length - 4 < 0x40, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[34f] ISZERO // [calldata.length - 4 < 0x40 == 0, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[350] PUSH2 035c // [0x35c, calldata.length - 4 < 0x40 == 0, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[353] JUMPI

// calldata.length - 4 >= 0x40 there needs to be at least 2 full words without counting the selector

[35c] JUMPDEST // [0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[35d] PUSH1 00 // [0, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[35f] DUP4 // [4, 0, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[360] ADD // [4 + 0, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[361] CALLDATALOAD // [firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[362] PUSH8 ffffffffffffffff// [0xffffffffffffffff, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[36b] DUP2 // [firstWord, 0xffffffffffffffff, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[36c] GT // [firstWord > 0xffffffffffffffff, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[36d] ISZERO // [firstWord > 0xffffffffffffffff == 0, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[36e] PUSH2 037a // [0x37a, firstWord > 0xffffffffffffffff == 0, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[371] JUMPI

// firstWord <= 0xffffffffffffffff

// Checked

[37a] JUMPDEST // [firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[37b] PUSH2 0386 // [0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[37e] DUP6 // [calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[37f] DUP3 // [firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[380] DUP7 // [4, firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[381] ADD // [4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[382] PUSH2 0276 // [0x276, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[385] JUMP

[276] JUMPDEST // [4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[277] PUSH1 00 // [0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[279] DUP3 // [calldata.length, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[27a] PUSH1 1f // [0x1f, calldata.length, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[27c] DUP4 // [4 + firstWord, 0x1f, calldata.length, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[27d] ADD // [4 + firstWord + 0x1f, calldata.length, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[27e] SLT // [4 + firstWord + 0x1f < calldata.length, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[27f] PUSH2 028b // [0x28b, 4 + firstWord + 0x1f < calldata.length, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[282] JUMPI

// 4 + firstWord + 0x1f < calldata.length check if exists the position in calldata of the first pointer

[28b] JUMPDEST // [0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[28c] DUP2 // [4 + firstWord, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[28d] CALLDATALOAD // [firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[28e] PUSH2 029b // [0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[291] DUP5 // [calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[292] DUP3 // [firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[293] PUSH1 20 // [0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[295] DUP7 // [4 + firstWord, 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[296] ADD // [4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[297] PUSH2 0234 // [0x234, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[29a] JUMP

[234] JUMPDEST // [4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[235] PUSH1 00 // [0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[237] PUSH2 0247 // [0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[23a] PUSH2 0242 // [0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[23d] DUP5 // [firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[23e] PUSH2 01f4 // [0x1f4, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[241] JUMP

[1f4] JUMPDEST // [firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1f5] PUSH1 00 // [0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1f7] PUSH8 ffffffffffffffff// [ffffffffffffffff, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[200] DUP3 // [firstBytes, ffffffffffffffff, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[201] GT // [firstBytes > ffffffffffffffff, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[202] ISZERO // [firstBytes > ffffffffffffffff == 0, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[203] PUSH2 020f

[206] JUMPI

// firstBytes <= ffffffffffffffff

[20f] JUMPDEST // [0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[210] PUSH2 0218 // [0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[213] DUP3 // [firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[214] PUSH2 0168

[217] JUMP

[168] JUMPDEST // [firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[169] PUSH1 00 // [0, firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[16b] PUSH1 1f // [0x1f, 0, firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[16d] NOT // [not(0x1f), 0, firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[16e] PUSH1 1f // [0x1f, not(0x1f), 0, firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[170] DUP4 // [firstBytes, 0x1f, not(0x1f), 0, firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[171] ADD // [firstBytes + 0x1f, not(0x1f), 0, firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[172] AND // [firstBytes + 0x1f + not(0x1f), 0, firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[173] SWAP1 // [0, firstBytes + 0x1f + not(0x1f), firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[174] POP // [firstBytes + 0x1f + not(0x1f), firstBytes, 0x218, 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[175] SWAP2 // [0x218, firstBytes, firstBytes + 0x1f + not(0x1f), 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[176] SWAP1 // [firstBytes, 0x218, firstBytes + 0x1f + not(0x1f), 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[177] POP // [0x218, firstBytes + 0x1f + not(0x1f), 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[178] JUMP

[218] JUMPDEST // [firstBytes + 0x1f + not(0x1f), 0, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[219] SWAP1 // [0, firstBytes + 0x1f + not(0x1f), firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[21a] POP // [firstBytes + 0x1f + not(0x1f), firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[21b] PUSH1 20 // [0x20, firstBytes + 0x1f + not(0x1f), firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[21d] DUP2 // [firstBytes + 0x1f + not(0x1f), 0x20, firstBytes + 0x1f + not(0x1f), firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[21e] ADD // [firstBytes + 0x1f + not(0x1f) + 0x20, firstBytes + 0x1f + not(0x1f), firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[21f] SWAP1 // [firstBytes + 0x1f + not(0x1f), firstBytes + 0x1f + not(0x1f) + 0x20, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[220] POP // [firstBytes + 0x1f + not(0x1f) + 0x20, firstBytes, 0x242, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[221] SWAP2 // [0x242, firstBytes, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[222] SWAP1 // [firstBytes, 0x242, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[223] POP // [0x242, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[224] JUMP

[242] JUMPDEST // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[243] PUSH2 01d9

[246] JUMP

[1d9] JUMPDEST // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1da] PUSH1 00 // [0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1dc] PUSH2 01e3 // [0x1e3, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1df] PUSH2 014a

[1e2] JUMP

[14a] JUMPDEST // [0x1e3, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[14b] PUSH1 00 // [0, 0x1e3, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[14d] PUSH1 40 // [0x40, 0, 0x1e3, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[14f] MLOAD // [0x80, 0, 0x1e3, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[150] SWAP1 // [0, 0x80, 0x1e3, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[151] POP // [0x80, 0x1e3, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[152] SWAP1 // [0x1e3, 0x80, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[153] JUMP

[1e3] JUMPDEST // [0x80, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1e4] SWAP1

[1e5] POP // [0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1e6] PUSH2 01ef // [0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1e9] DUP3 // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1ea] DUP3 // [0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1eb] PUSH2 01a8 // [0x1a8, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1ee] JUMP

[1a8] JUMPDEST

[1a9] PUSH2 01b1 // [0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1ac] DUP3 // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1ad] PUSH2 0168

[1b0] JUMP

[168] JUMPDEST

[169] PUSH1 00 // [0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[16b] PUSH1 1f // [0x1f, 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[16d] NOT // [not(0x1f), 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[16e] PUSH1 1f // [0x1f, not(0x1f), 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[170] DUP4 // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x1f, not(0x1f), 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[171] ADD // [firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f), 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[172] AND // [AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[173] SWAP1 // [0, AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[174] POP // [AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[175] SWAP2 // [0x1b1, firstBytes + 0x1f + not(0x1f) + 0x20, AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[176] SWAP1 // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x1b1, AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[177] POP // [0x1b1, AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[178] JUMP

[1b1] JUMPDEST // [AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1b2] DUP2 // [0x80, AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1b3] ADD // [0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1b4] DUP2 // [0x80, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1b5] DUP2 // [0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1b6] LT // [(0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f))) < 0x80, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1b7] PUSH8 ffffffffffffffff// [0xffffffffffffffff, (0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f))) < 0x80, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1c0] DUP3 // [0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, 0xffffffffffffffff, (0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f))) < 0x80, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1c1] GT // [0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f > 0xffffffffffffffff, (0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f))) < 0x80, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1c2] OR // [OR(0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f > 0xffffffffffffffff, (0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f))) < 0x80) , 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1c3] ISZERO // [OR(0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f > 0xffffffffffffffff, (0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f))) < 0x80) == 0 , 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1c4] PUSH2 01d0

[1c7] JUMPI

// OR(,) == 0

// 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f <= 0xffffffffffffffff

// 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)) >= 0x80

[1d0] JUMPDEST // [0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1d1] DUP1 // [0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1d2] PUSH1 40 // [0x40, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1d4] MSTORE // store at 0x40 the value 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)) // [0x40, 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80 + AND(firstBytes + 0x1f + not(0x1f) + 0x20 + 0x1f, not(0x1f)), 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1d5] POP // [0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1d6] POP // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1d7] POP // [0x1ef, 0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1d8] JUMP

[1ef] JUMPDEST // [0x80, firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1f0] SWAP2 // [0x247, firstBytes + 0x1f + not(0x1f) + 0x20, 0x80, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1f1] SWAP1 // [firstBytes + 0x1f + not(0x1f) + 0x20, 0x247, 0x80, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1f2] POP // [0x247, 0x80, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[1f3] JUMP

[247] JUMPDEST // [0x80, 0, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[248] SWAP1 // [0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[249] POP // [0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[24a] DUP3 // [firstBytes, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[24b] DUP2 // [0x80, firstBytes, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[24c] MSTORE // save firstBytes at 0x80 // [0x80, firstBytes, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[24d] PUSH1 20 // [0x20, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[24f] DUP2 // [0x80, 0x20, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[250] ADD // [0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[251] DUP5 // [calldata.length, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[252] DUP5 // [firstBytes, calldata.length, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[253] DUP5 // [4 + firstWord + 0x20, firstBytes, calldata.length, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[254] ADD // [4 + firstWord + 0x20 + firstBytes, calldata.length, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[255] GT // [4 + firstWord + 0x20 + firstBytes > calldata.length, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[256] ISZERO // [4 + firstWord + 0x20 + firstBytes > calldata.length == 0, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[257] PUSH2 0263

[25a] JUMPI

// 4 + firstWord + 0x20 + firstBytes <= calldata.length

[263] JUMPDEST // [0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[264] PUSH2 026e // [0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[267] DUP5 // [firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[268] DUP3 // [0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[269] DUP6 // [4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[26a] PUSH2 0225

[26d] JUMP

[225] JUMPDEST

[226] DUP3 // [firstBytes, 4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[227] DUP2 // [4 + firstWord + 0x20, firstBytes, 4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[228] DUP4 // [0xa0, 4 + firstWord + 0x20, firstBytes, 4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[229] CALLDATACOPY // stored at memory position 0xa0 the value of calldata offset 4 + firstWord + 0x20 with firstBytes of size // [4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[22a] PUSH1 00 // [0, 4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[22c] DUP4 // [firstBytes, 0, 4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[22d] DUP4 // [0xa0, firstBytes, 0, 4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[22e] ADD // [0xa0 + firstBytes, 0, 4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[22f] MSTORE // store at position 0xa0 + firstBytes the 0 value // [4 + firstWord + 0x20, 0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[230] POP // [0xa0, firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[231] POP // [firstBytes, 0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[232] POP // [0x26e, 0xa0, 0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[233] JUMP

[26e] JUMPDEST

[26f] POP // [0x80, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x29b, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[270] SWAP4 // [0x29b, 4 + firstWord + 0x20, firstBytes, calldata.length, 0x80, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[271] SWAP3 // [calldata.length, 4 + firstWord + 0x20, firstBytes, 0x29b, 0x80, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[272] POP // [4 + firstWord + 0x20, firstBytes, 0x29b, 0x80, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[273] POP

[274] POP // [0x29b, 0x80, firstBytes, 0, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[275] JUMP

[29b] JUMPDEST

[29c] SWAP2 // [0, firstBytes, 0x80, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[29d] POP

[29e] POP // [0x80, 4 + firstWord, calldata.length, 0x386, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[29f] SWAP3 // [0x386, 4 + firstWord, calldata.length, 0x80, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[2a0] SWAP2 // [calldata.length, 4 + firstWord, 0x386, 0x80, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[2a1] POP

[2a2] POP

[2a3] JUMP

[386] JUMPDEST // [0x80, firstWord, 0, 0, 4, calldata.length, 0x45, 0x4a, selector]

[387] SWAP3 // [0, firstWord, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[388] POP

[389] POP

[38a] PUSH1 20 // [0x20, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[38c] DUP4 // [4, 0x20, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[38d] ADD // [0x24, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[38e] CALLDATALOAD // [secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[38f] PUSH8 ffffffffffffffff// [0xffffffffffffffff, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[398] DUP2 // [secondPointer, 0xffffffffffffffff, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[399] GT // [secondPointer > 0xffffffffffffffff, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[39a] ISZERO // [secondPointer > 0xffffffffffffffff == 0, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[39b] PUSH2 03a7

[39e] JUMPI

// secondPointer <= 0xffffffffffffffff

[3a7] JUMPDEST

[3a8] PUSH2 03b3 // [0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[3ab] DUP6 // [calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[3ac] DUP3 // [secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[3ad] DUP7 // [4, secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[3ae] ADD // [4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[3af] PUSH2 0317

[3b2] JUMP

[317] JUMPDEST

[318] PUSH1 00 // [0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[31a] DUP3 // [calldata.length, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[31b] PUSH1 1f // [0x1f, calldata.length, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[31d] DUP4 // [4 + secondPointer, 0x1f, calldata.length, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[31e] ADD // [4 + secondPointer + 0x1f, calldata.length, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[31f] SLT // [4 + secondPointer + 0x1f < calldata.length, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[320] PUSH2 032c

[323] JUMPI

// 4 + secondPointer + 0x1f < calldata.length

[32c] JUMPDEST // [0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[32d] DUP2 // [4 + secondPointer, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[32e] CALLDATALOAD // [secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[32f] PUSH2 033c // [0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[332] DUP5 // [calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[333] DUP3 // [secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[334] PUSH1 20 // [0x20, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[336] DUP7 // [4 + secondPointer, 0x20, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[337] ADD // [secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[338] PUSH2 02d5

[33b] JUMP

[2d5] JUMPDEST

[2d6] PUSH1 00 // [0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2d8] PUSH2 02e8 // [0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2db] PUSH2 02e3 // [0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2de] DUP5 // [secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2df] PUSH2 02a4 // [secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2e2] JUMP

[2a4] JUMPDEST

[2a5] PUSH1 00 // [0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2a7] PUSH8 ffffffffffffffff// [0xffffffffffffffff, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2b0] DUP3 // [secondBytes, 0xffffffffffffffff, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2b1] GT // [secondBytes > 0xffffffffffffffff, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2b2] ISZERO // [secondBytes > 0xffffffffffffffff == 0, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2b3] PUSH2 02bf

[2b6] JUMPI

// secondBytes <= 0xffffffffffffffff

[2bf] JUMPDEST

[2c0] PUSH2 02c8 // [0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2c3] DUP3 // [secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2c4] PUSH2 0168

[2c7] JUMP

[168] JUMPDEST

[169] PUSH1 00 // [0, secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[16b] PUSH1 1f // [0x1f, 0, secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[16d] NOT // [not(0x1f), 0, secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[16e] PUSH1 1f // [0x1f, not(0x1f), 0, secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[170] DUP4 // [secondBytes, 0x1f, not(0x1f), 0, secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[171] ADD // [secondBytes + 0x1f, not(0x1f), 0, secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[172] AND // [AND(secondBytes + 0x1f, not(0x1f)), 0, secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[173] SWAP1 // [0, AND(secondBytes + 0x1f, not(0x1f)), secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[174] POP // [AND(secondBytes + 0x1f, not(0x1f)), secondBytes, 0x2c8, 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[175] SWAP2 // [0x2c8, secondBytes, AND(secondBytes + 0x1f, not(0x1f)), 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[176] SWAP1 // [secondBytes, 0x2c8, AND(secondBytes + 0x1f, not(0x1f)), 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[177] POP // [0x2c8, AND(secondBytes + 0x1f, not(0x1f)), 0, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[178] JUMP

[2c8] JUMPDEST

[2c9] SWAP1 // [0, AND(secondBytes + 0x1f, not(0x1f)), secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2ca] POP // [AND(secondBytes + 0x1f, not(0x1f)), secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2cb] PUSH1 20 // [0x20, AND(secondBytes + 0x1f, not(0x1f)), secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2cd] DUP2 // [AND(secondBytes + 0x1f, not(0x1f)), 0x20, AND(secondBytes + 0x1f, not(0x1f)), secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2ce] ADD // [AND(secondBytes + 0x1f, not(0x1f)) + 0x20, AND(secondBytes + 0x1f, not(0x1f)), secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2cf] SWAP1 // [AND(secondBytes + 0x1f, not(0x1f)), AND(secondBytes + 0x1f, not(0x1f)) + 0x20, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2d0] POP // [AND(secondBytes + 0x1f, not(0x1f)) + 0x20, secondBytes, 0x2e3, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2d1] SWAP2 // [0x2e3, secondBytes, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2d2] SWAP1 // [secondBytes, 0x2e3, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2d3] POP // [0x2e3, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2d4] JUMP

[2e3] JUMPDEST

[2e4] PUSH2 01d9

[2e7] JUMP

[1d9] JUMPDEST

[1da] PUSH1 00 // [0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1dc] PUSH2 01e3 // [0x1e3, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1df] PUSH2 014a

[1e2] JUMP

[14a] JUMPDEST

[14b] PUSH1 00 // [0, 0x1e3, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[14d] PUSH1 40 // [0x40, 0, 0x1e3, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[14f] MLOAD // [fmp, 0, 0x1e3, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[150] SWAP1 // [0, fmp, 0x1e3, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[151] POP // [fmp, 0x1e3, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[152] SWAP1 // [0x1e3, fmp, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[153] JUMP

[1e3] JUMPDEST // [fmp, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1e4] SWAP1 // [0, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1e5] POP

[1e6] PUSH2 01ef // [0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1e9] DUP3 // [AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1ea] DUP3 // [fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1eb] PUSH2 01a8

[1ee] JUMP

[1a8] JUMPDEST

[1a9] PUSH2 01b1 // [0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1ac] DUP3 // [AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1ad] PUSH2 0168

[1b0] JUMP

[168] JUMPDEST

[169] PUSH1 00 // [0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[16b] PUSH1 1f // [0x1f, 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[16d] NOT // [not(0x1f), 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[16e] PUSH1 1f // [0x1f, not(0x1f), 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[170] DUP4 // [AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1f, not(0x1f), 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[171] ADD // [AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f), 0, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[172] AND // [AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1b1, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[173] SWAP1

[174] POP

[175] SWAP2 // [0x1b1, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[176] SWAP1

[177] POP // [0x1b1, AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[178] JUMP

[1b1] JUMPDEST

[1b2] DUP2 // [fmp, AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1b3] ADD // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1b4] DUP2 // [fmp, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1b5] DUP2 // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1b6] LT // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) < fmp, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1b7] PUSH8 ffffffffffffffff// [0xffffffffffffffff, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) < fmp, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1c0] DUP3 // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), 0xffffffffffffffff, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) < fmp, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1c1] GT // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) > 0xffffffffffffffff, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) < fmp, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1c2] OR // [OR(fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) > 0xffffffffffffffff, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) < fmp), fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1c3] ISZERO // [OR(fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) > 0xffffffffffffffff, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) < fmp) == 0, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1c4] PUSH2 01d0

[1c7] JUMPI

// OR(,) == 0

// fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) <= 0xffffffffffffffff

// fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) >= fmp

[1d0] JUMPDEST // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1d1] DUP1 // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1d2] PUSH1 40 // [0x40, fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1d4] MSTORE // store at position 0x40 the value fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)) // [fmp + AND(AND(secondBytes + 0x1f, not(0x1f)) + 0x20 + 0x1f, not(0x1f)), fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1d5] POP

[1d6] POP

[1d7] POP

[1d8] JUMP // [0x1ef, fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1ef] JUMPDEST // [fmp, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1f0] SWAP2 // [0x2e8, AND(secondBytes + 0x1f, not(0x1f)) + 0x20, fmp, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1f1] SWAP1 // [AND(secondBytes + 0x1f, not(0x1f)) + 0x20, 0x2e8, fmp, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1f2] POP // [0x2e8, fmp, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[1f3] JUMP

[2e8] JUMPDEST // [fmp, 0, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2e9] SWAP1 // [0, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2ea] POP // [fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2eb] DUP3 // [secondBytes, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2ec] DUP2 // [fmp, secondBytes, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2ed] MSTORE // store at fmp the secondBytes // [fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2ee] PUSH1 20 // [0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f0] DUP2 // [fmp, 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f1] ADD // [fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f2] DUP5 // [calldata.length, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f3] DUP5 // [secondBytes, calldata.length, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f4] DUP5 // [secondPointer + 0x24, secondBytes, calldata.length, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f5] ADD // [secondPointer + 0x24 + secondBytes, calldata.length, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f6] GT // [secondPointer + 0x24 + secondBytes > calldata.length, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f7] ISZERO // [secondPointer + 0x24 + secondBytes > calldata.length == 0, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[2f8] PUSH2 0304

[2fb] JUMPI

// secondPointer + 0x24 + secondBytes <= calldata.length

[304] JUMPDEST // [fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[305] PUSH2 030f // [0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[308] DUP5 // [secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[309] DUP3 // [fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[30a] DUP6 // [secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[30b] PUSH2 0225

[30e] JUMP

[225] JUMPDEST

[226] DUP3 // [secondBytes, secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[227] DUP2 // [secondPointer + 0x24, secondBytes, secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[228] DUP4 // [fmp + 0x20, secondPointer + 0x24, secondBytes, secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[229] CALLDATACOPY // stored at memory position fmp + 0x20 the value of calldata offset secondPointer + 0x24 with secondBytes of size // [secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[22a] PUSH1 00 // [0, secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[22c] DUP4 // [secondBytes, 0, secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[22d] DUP4 // [fmp + 0x20, secondBytes, 0, secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[22e] ADD // [fmp + 0x20 + secondBytes, 0, secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[22f] MSTORE // store at position fmp + 0x20 + secondBytes the value 0 // [secondPointer + 0x24, fmp + 0x20, secondBytes, 0x30f, fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[230] POP

[231] POP

[232] POP

[233] JUMP

[30f] JUMPDEST // [fmp + 0x20, fmp, secondPointer + 0x24, secondBytes, calldata.length, 0x33c, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[310] POP

[311] SWAP4 // [0x33c, secondPointer + 0x24, secondBytes, calldata.length, fmp, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[312] SWAP3 // [calldata.length, secondPointer + 0x24, secondBytes, 0x33c, fmp, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[313] POP

[314] POP

[315] POP

[316] JUMP

[33c] JUMPDEST // [fmp, secondBytes, 0, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[33d] SWAP2 // [0, secondBytes, fmp, 4 + secondPointer, calldata.length, 0x3b3, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[33e] POP

[33f] POP

[340] SWAP3 // [0x3b3, 4 + secondPointer, calldata.length, fmp, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[341] SWAP2 // [calldata.length, 4 + secondPointer, 0x3b3, fmp, secondPointer, 0, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[342] POP

[343] POP

[344] JUMP

[3b3] JUMPDEST

[3b4] SWAP2 // [0, secondPointer, fmp, 0x80, 4, calldata.length, 0x45, 0x4a, selector]

[3b5] POP

[3b6] POP

[3b7] SWAP3 // [calldata.length, 0x80, 4, fmp, 0x45, 0x4a, selector]

[3b8] POP

[3b9] SWAP3 // [0x45, 4, fmp, 0x80, 0x4a, selector]

[3ba] SWAP1 // [4, 0x45, fmp, 0x80, 0x4a, selector]

[3bb] POP

[3bc] JUMP

[45] JUMPDEST

[46] PUSH2 004c // [0x4c, fmp, 0x80, 0x4a, selector]

[49] JUMP

[4c] JUMPDEST

[4d] PUSH1 00 // [0, fmp, 0x80, 0x4a, selector]

[4f] PUSH1 40 // [0x40, 0, fmp, 0x80, 0x4a, selector]

[51] MLOAD // [newFmp, 0, fmp, 0x80, 0x4a, selector]

[52] DUP1 // [newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[53] PUSH1 40 // [0x40, newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[55] ADD // [0x40 + newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[56] PUSH1 40 // [0x40, 0x40 + newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[58] MSTORE // store at 0x40 0x40 + newFmp // [newFmp, 0, fmp, 0x80, 0x4a, selector]

[59] DUP1 // [newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[5a] PUSH1 0d // [0xd, newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[5c] DUP2 // [newFmp, 0xd, newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[5d] MSTORE // store at newFmp the value 0xd // [newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[5e] PUSH1 20 // [0x20, newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[60] ADD // [0x20 + newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[61] PUSH32 616c6d6f73742074686572652100000000000000000000000000000000000000 // [0x616c6d6f73742074686572652100000000000000000000000000000000000000, 0x20 + newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[82] DUP2 // [0x20 + newFmp, 0x616c6d6f73742074686572652100000000000000000000000000000000000000, 0x20 + newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[83] MSTORE // store at 0x20 + newFmp the value 0x616c6d6f73742074686572652100000000000000000000000000000000000000 // [0x20 + newFmp, newFmp, 0, fmp, 0x80, 0x4a, selector]

[84] POP // [newFmp, 0, fmp, 0x80, 0x4a, selector]

[85] SWAP1 // [0, newFmp, fmp, 0x80, 0x4a, selector]

[86] POP // [newFmp, fmp, 0x80, 0x4a, selector]

[87] PUSH1 20 // [0x20, newFmp, fmp, 0x80, 0x4a, selector]

[89] DUP3 // [fmp, 0x20, newFmp, fmp, 0x80, 0x4a, selector]

[8a] ADD // [fmp + 0x20, newFmp, fmp, 0x80, 0x4a, selector]

[8b] MLOAD // [mem(fmp + 0x20), newFmp, fmp, 0x80, 0x4a, selector]

[8c] PUSH1 20 // [0x20, mem(fmp + 0x20), newFmp, fmp, 0x80, 0x4a, selector]

[8e] DUP3 // [newFmp, 0x20, mem(fmp + 0x20), newFmp, fmp, 0x80, 0x4a, selector]

[8f] ADD // [newFmp + 0x20, mem(fmp + 0x20), newFmp, fmp, 0x80, 0x4a, selector]

[90] MLOAD // [616c6d6f73742074686572652100000000000000000000000000000000000000, mem(fmp + 0x20), newFmp, fmp, 0x80, 0x4a, selector]

[91] EQ // [616c6d6f73742074686572652100000000000000000000000000000000000000 == mem(fmp + 0x20), newFmp, fmp, 0x80, 0x4a, selector]

[92] PUSH2 009a

[95] JUMPI

// 616c6d6f73742074686572652100000000000000000000000000000000000000 == mem(fmp + 0x20)

[9a] JUMPDEST // [newFmp, fmp, 0x80, 0x4a, selector]

[9b] PUSH1 00 // [0, newFmp, fmp, 0x80, 0x4a, selector]

[9d] PUSH1 40 // [0x40, 0, newFmp, fmp, 0x80, 0x4a, selector]

[9f] MLOAD // [newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[a0] DUP1 // [newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[a1] PUSH2 0720 // [0x720, newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[a4] ADD // [0x720 + newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[a5] PUSH1 40 // [0x40, 0x720 + newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[a7] MSTORE // stored at memory 0x40 the value 0x720 + newFmp2 // [newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[a8] DUP1 // [newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[a9] PUSH2 06ed // [0x6ed, newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[ac] DUP2 // [newFmp2, 0x6ed, newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[ad] MSTORE // stored at newFmp2 the value 0x6ed // [newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[ae] PUSH1 20 // [0x20, newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[b0] ADD // [0x20 + newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[b1] PUSH2 0475 // [0x475, 0x20 + newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[b4] PUSH2 06ed // [0x6ed, 0x475, 0x20 + newFmp2, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[b7] SWAP2 // [0x20 + newFmp2, 0x475, 0x6ed, newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[b8] CODECOPY // stored at memory newFmp2 + 0x20 the code in the offset 0x475, 0x6ed amount of bytes // [newFmp2, 0, newFmp, fmp, 0x80, 0x4a, selector]

[b9] SWAP1 // [0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[ba] POP // [newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[bb] PUSH1 00 // [0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[bd] DUP2 // [newFmp2, 0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[be] MLOAD // [0x6ed, 0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[bf] PUSH1 20 // [0x20, 0x6ed, 0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[c1] DUP4 // [newFmp2, 0x20, 0x6ed, 0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[c2] ADD // [newFmp2 + 0x20, 0x6ed, 0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[c3] PUSH1 00 // [0, newFmp2 + 0x20, 0x6ed, 0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[c5] CREATE // created a contract with 0 wei, and code as memory starting at newFmp2 + 0x20, the amount of 0x6ed bytes // [contractAddress, 0, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[c6] SWAP1 // [0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[c7] POP

[c8] PUSH1 00

[ca] DUP2 // [contractAddress, 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[cb] PUSH20 ffffffffffffffffffffffffffffffffffffffff // [0xffffffffffffffffffffffffffffffffffffffff, contractAddress, 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[e0] AND // [AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[e1] DUP7 // [0x80, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[e2] PUSH1 40 // [0x40, 0x80, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[e4] MLOAD // [newFmp3, 0x80, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[e5] PUSH2 00ee // [0xee, newFmp3, 0x80, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[e8] SWAP2 // [0x80, newFmp3, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[e9] SWAP1 // [newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[ea] PUSH2 042e

[ed] JUMP

[42e] JUMPDEST

[42f] PUSH1 00 // [0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[431] PUSH2 043a // [0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[434] DUP3 // [newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[435] DUP5 // [0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[436] PUSH2 03fd

[439] JUMP

[3fd] JUMPDEST

[3fe] PUSH1 00 // [0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[400] PUSH2 0408 // [0x408, 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[403] DUP3 // [0x80, 0x408, 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[404] PUSH2 03bd

[407] JUMP

[3bd] JUMPDEST

[3be] PUSH1 00 // [0, 0x80, 0x408, 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c0] DUP2 // [0x80, 0, 0x80, 0x408, 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c1] MLOAD // [mem(0x80), 0, 0x80, 0x408, 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c2] SWAP1 // [0, mem(0x80), 0x80, 0x408, 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c3] POP // [mem(0x80), 0x80, 0x408, 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c4] SWAP2 // [0x408, 0x80, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c5] SWAP1 // [0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c6] POP // [0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3c7] JUMP

[408] JUMPDEST

[409] PUSH2 0412 // [0x412, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[40c] DUP2 // [0x408, 0x412, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[40d] DUP6 // [0x80, 0x408, 0x412, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[40e] PUSH2 03c8

[411] JUMP

[3c8] JUMPDEST

[3c9] PUSH1 00 // [0, 0x80, 0x408, 0x412, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3cb] DUP2 // [0x80, 0, 0x80, 0x408, 0x412, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3cc] SWAP1 // [0, 0x80, 0x80, 0x408, 0x412, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3cd] POP // [0x80, 0x80, 0x408, 0x412, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3ce] SWAP3 // [0x412, 0x80, 0x408, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3cf] SWAP2 // [0x408, 0x80, 0x412, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3d0] POP

[3d1] POP

[3d2] JUMP

[412] JUMPDEST // [0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[413] SWAP4 // [0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[414] POP // [0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[415] PUSH2 0422 // [0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[418] DUP2 // [0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[419] DUP6 // [0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[41a] PUSH1 20 // [0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[41c] DUP7 // [mem(0x80), 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[41d] ADD // [mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[41e] PUSH2 03d3

[421] JUMP

// Loop here starts

[3d3] JUMPDEST

[3d4] PUSH1 00 // [0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3d6] JUMPDEST

[3d7] DUP4 // [0x80, 0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3d8] DUP2 // [0, 0x80, 0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3d9] LT // [0 < 0x80, 0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3da] ISZERO // [0 < 0x80 == 0, 0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3db] PUSH2 03f1

// to leave the loop 0 >= 0x80

[3de] JUMPI

[3df] DUP1 // [0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e0] DUP3 // [0, 0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e1] ADD // [0, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e2] MLOAD // [mem(0), mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e3] DUP2 // [mem(0x80) + 0x20, mem(0), mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e4] DUP5 // [0x80, mem(0x80) + 0x20, mem(0), mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e5] ADD // [0x80 + mem(0x80) + 0x20, mem(0), mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e6] MSTORE // store are 0x80 + mem(0x80) the value mem(0) // [mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e7] PUSH1 20 // [0x20, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3e9] DUP2 // [mem(0x80) + 0x20, 0x20, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3ea] ADD // [mem(0x80) + 0x20 + 0x20, mem(0x80) + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3eb] SWAP1 // [mem(0x80) + 0x20, mem(0x80) + 0x20 + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3ec] POP // [mem(0x80) + 0x20 + 0x20, 0, 0x80, 0x422, 0x80, 0x408, mem(0x80), 0, 0x80, newFmp3, 0x43a, 0, newFmp3, 0x80, 0xee, AND(0xffffffffffffffffffffffffffffffffffffffff, contractAddress), 0, contractAddress, newFmp2, newFmp, fmp, 0x80, 0x4a, selector]

[3ed] PUSH2 03d6

[3f0] JUMP

// End of loop

[3f1] JUMPDEST

[3f2] PUSH1 00

[3f4] DUP5

[3f5] DUP5

[3f6] ADD

[3f7] MSTORE

[3f8] POP

[3f9] POP

[3fa] POP

[3fb] POP

[3fc] JUMP

[422] JUMPDEST

[423] DUP1

[424] DUP5

[425] ADD

[426] SWAP2

[427] POP

[428] POP

[429] SWAP3

[42a] SWAP2

[42b] POP

[42c] POP

[42d] JUMP

[43a] JUMPDEST

[43b] SWAP2

[43c] POP

[43d] DUP2

[43e] SWAP1

[43f] POP

[440] SWAP3

[441] SWAP2

[442] POP

[443] POP

[444] JUMP

[ee] JUMPDEST

[ef] PUSH1 00

[f1] PUSH1 40

[f3] MLOAD

[f4] DUP1

[f5] DUP4

[f6] SUB

[f7] DUP2

[f8] PUSH1 00

[fa] DUP7

[fb] GAS

[fc] CALL

[fd] SWAP2

[fe] POP

[ff] POP

[100] RETURNDATASIZE

[101] DUP1

[102] PUSH1 00

[104] DUP2

[105] EQ

[106] PUSH2 012b

[109] JUMPI

// Here are 2 branches before reaching the end

// First branch (No jump):

[10a] PUSH1 40

[10c] MLOAD

[10d] SWAP2

[10e] POP

[10f] PUSH1 1f

[111] NOT

[112] PUSH1 3f

[114] RETURNDATASIZE

[115] ADD

[116] AND

[117] DUP3

[118] ADD

[119] PUSH1 40

[11b] MSTORE

[11c] RETURNDATASIZE

[11d] DUP3

[11e] MSTORE

[11f] RETURNDATASIZE

[120] PUSH1 00

[122] PUSH1 20

[124] DUP5

[125] ADD

[126] RETURNDATACOPY

[127] PUSH2 0130

[12a] JUMP

// Second branch (Jump):

[12b] JUMPDEST

[12c] PUSH1 60

[12e] SWAP2

[12f] POP

// Final opcodes

[130] JUMPDEST

[131] POP

[132] POP

[133] SWAP1

[134] POP

[135] DUP1

[136] PUSH2 0142

[139] JUMPI

[142] JUMPDEST

[143] POP

[144] POP

[145] POP

[146] POP

[147] POP

[148] POP

[149] JUMP

[4a] JUMPDEST

[4b] STOP // Win