Gera Jahja

Class of 2022

IN3007 Project Report

Decompiling Ethereum EVM Bytecode for Static Analysis.

Decompiling Ethereum EVM Bytecode for Static Analysis.

* Prepared by: Gera Jahja, Computer Science BSc
* Prepared for: City UOL, Department of Computer Science
* Email: gera.jahja@city.ac.uk / g\_jahja31@outlook.com
* Consultant: Martin Nyx Brain
* Academic Client: Martin Nyx Brain and Michał Król
* Version Control:
  + V1.0: 16th Feb 2022: Introduction and Project Definition sections added (pages 1-7, Word Count: 1293/15,000)
  + V1.2: 20th March 2022: changed layout, added chapters 2-5 and drafted first three/four chapters (Word Count: 1293/15,000)
  + V1.3:30th March 2022 documented chapters 2-5 (Word Count: 1293/15,000)

This project is an academic proposal supervised by Martin Nyx Brain and Michał Król. The project will be to write a decompiler so that it can convert EVM byte code into C that can be handled by the CPROVER tools.

Table of contents

[Chapter 1 3](#_Toc98980951)

[1.1 Introduction 3](#_Toc98980952)

[1.1.1 Description of the problem 3](#_Toc98980953)

[Project Objectives: 3](#_Toc98980954)

[Project Plan 4](#_Toc98980955)

[Project Structure: 12](#_Toc98980956)

[Iterative Development: 11](#_Toc98980957)

[Project Beneficiaries: 4](#_Toc98980958)

[Risks of the project: Error! Bookmark not defined.](#_Toc98980959)

[Project Management: **Error! Bookmark not defined.**](#_Toc98980960)

[Technical: **Error! Bookmark not defined.**](#_Toc98980961)

[Personal: **Error! Bookmark not defined.**](#_Toc98980962)

[Risk assessment: **Error! Bookmark not defined.**](#_Toc98980963)

[Chapter 2 4](#_Toc98980964)

[Project Outputs (2 pages) 4](#_Toc98980965)

[Chapter 3 6](#_Toc98980966)

[Literature Review 6](#_Toc98980967)

[Chapter 4 11](#_Toc98980968)

[Methodology 11](#_Toc98980969)

[Analysis 11](#_Toc98980970)

[Design 12](#_Toc98980971)

[Implementation 12](#_Toc98980972)

[Evaluation 12](#_Toc98980973)

[Chapter 5 13](#_Toc98980974)

[Results 13](#_Toc98980975)

[References: Error! Bookmark not defined.](#_Toc98980976)

must not exceed 12,000 words for BSc projects, (should be 8000+)

# Chapter 1

## 1.1 Introduction

Ethereum is one of the most exciting block chain technologies as it is not just a cryptocurrency but also supports smart contracts. These are programs that are written in a variety of programming languages and compiled to EVM, a byte-code format like JVM, before they are run on the block chain. The security of these contracts is vital as they can control significant amounts of cryptocurrency.

### 1.1.1 Description of the problem

This project requires me to translate EVM byte code to C code. The purpose of this translation is to see whether we can detect bugs using CPROVER tools. If we can successfully use tools used to detect C code bugs on EVM this means we can add verification to EVM (as well as aiding our understanding of the behaviour of the smart contracts) and ensure that the Ethereum currency that is associated with the byte code is protected and less viable to hacking.

Decompiling is a part of reverse engineering, I will be using this approach to convert EVM byte code to op code, and then generating C code from this.

When looking to verify whether the software works, I will be specifically be looking at answering:

*Will the program crash? Can it be hacked? Can we do it without running the original program? (Static Analysis, i.e., using CPROVER)*

### 1.1.2 Project Objectives:

* Produce a subset of EVM byte code that has been decompiled to C code.
* Develop a program that displays this decomplication
* Ensure this subset (of EVM byte code) can be run through the CPROVER tool CBMC
* Investigate to what degree do these tools allow us to analyze the EVM contracts, I.e. is it able to aid us in detecting bugs?

These objectives will follow this work plan:

Graphical user interface, text, application

Description automatically generated

### 1.1.3 sub objectives? Research questions??

### 1.1.3 Project Beneficiaries

Beneficiaries of the project include:

1. My clients and I:
   * Martin is quite interested in the verification end of the problem we are trying to solve, while Michael is interested in the Ethereum cryptocurrency side of things. My aim is to pave a connection between the two.
2. Academics:
   * The findings of this development may be of interests to academics in similar fields of work some examples include :

Cyber Security, Decomplication EVM bytecode, reverse engineering and more.

1. People using blockchain:
   * If we can validate smart contracts, it can prevent huge financial losses for people that are part of the Ethereum blockchain.
   * In the past there have been times where hacking into the currency has led to cryptocurrency being destroyed. In 2017, “$300m of cryptocurrency was lost after a series of bugs in a popular digital wallet service led one curious developer to accidentally take control of and then lock up the funds” Locating these bugs can prevent similar cases from occurring, adding more security to the currency and more protection for people using block chain.

In general, building tools that are useful is the main benefit of this project. Identifying and ideally preventing bugs with evidence that it works would be the ideal outcome of this project.

### 1.1.4 Outline the work you performed to meet the objectives and answer the research questions

### 1.1.5 State any assumptions that you may have made and, where relevant, how you limited the scope of the project

# Chapter 2

## Project Outputs

|  |  |
| --- | --- |
| Output | Project Definition Document |
| Description | Document that outlined the original set of requirements, project objectives, work plan and project risks that were requested by my academic client |
| Type of Output |  |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

|  |  |
| --- | --- |
| Output | EVM Decompiler |
| Description | A fully functional software product that decompiles EVM byte code to relevant C Code . --- lines of Java code , ---- of which were based on a coding tutorial. |
| Type of Output | This product is built as -- Java classes, amounting to --- lines of Java code (counting comments), of which --- classes (--- lines plus comments) were written by me and the rest are reused from the sources indicated in the Results section. The complete software plus its design documentation can be found in the USB memory stick I provided as offline submission, or on GitHub at the URL : |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

|  |  |
| --- | --- |
| Output | Testing of Disassembly of the EVM byte code |
| Description | Automated testing in java that checks whether the EVM code has been disassembled, if these pass then the EVM is ready to be decompiled to . |
| Type of Output | Software product and documented results of running the code ( --- test classes.) |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

|  |  |
| --- | --- |
| Output | C Prover Testing of generated C code |
| Description | Generated test coverage, back-to-back testing where I have generated line / path coverage and then taken those inputs to run on the original EVM code to see whether they are valid. |
| Type of Output |  |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

|  |  |
| --- | --- |
| Output | Final Testing |
| Description | *Testing validating that the Generated c code can be linked back to the original EVM* |
| Type of Output |  |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

|  |  |
| --- | --- |
| Output | Deployment Guide |
| Description |  |
| Type of Output |  |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

|  |  |
| --- | --- |
| Output | Software Documentation |
| Description |  |
| Type of Output |  |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

|  |  |
| --- | --- |
| Output | Meeting Records |
| Description | A document stating all the client meetings and the topics discussed. |
| Type of Output |  |
| Recipient | Academic Client and any audience of the project report |
| Usage |  |
| Results Location |  |

• A statement of what type of output it is (e.g. software code, a system specification, an evaluation report), its size (if relevant) and any materials it re-uses. Be brief and informative, e.g. you might state “the deliverable is 10,000 lines of C code (not counting comments), of which 3,000 were written by me and the rest re-used”.

• The intended recipient(s) and/or end-user(s) of the output.

• How these recipients will use the output and benefit from it.

• Links to the sections of the Results chapter, identifying how the results described in that chapter contribute to each output.

• Link to one or more Appendices where the complete output is reported.

For instance, this summary may contain information like: “the first output is a software product that classifies gene mutations as prescribed by the standard classification prescribed in reference (Xnjkls, 1999).” or “an analysis and diagnosis of the malfunctions in the automatic management of security patches on BlaxR2 servers at company SplatBfec, with a proposed solution” “This product is built as 33 Java classes, amounting to 10,300 lines of Java code (not counting comments), of which 10 classes (3,200 lines plus comments) were written by me and the rest are reused from the sources indicated in the Results section. The complete software plus its design documentation can be found in the USB memory stick I provided as offline submission and can be used on the web at the URL: https://....” “this product was delivered to my client, whose research team will use it to ..” or “this software will be used by my client CompanyX in their business for..” “a prototype of the web service is hosted at http:... so that the markers can test it”

# Chapter 3

## Literature Review

Ethereum is a popular blockchain platform that supports full-featured smart contracts. Developers typically write smart contracts in a high-level language called Solidity, which is compiled into immutable low-level Ethereum VM (EVM) bytecode for the blockchain’s distributed virtual machine. This bytecode is the ultimate semantics of the contract.

### What is the Ethereum Virtual Machine?

Ethereum is a platform powered by blockchain technology that is best known for its native cryptocurrency. This blockchain supports the development of smart contracts , which are processed by the Ethereum virtual machine.

EVM (Ethereum Virtual Machine ) is a stack-based interpreter of EVM bytecode. EVM does not have any "system interface" handling or “hardware support”—there is no physical machine to connect with, it is completely virtual.

The EVM is a Turing-complete state machine, because all execution steps are limited to a finite number of computational steps. In comparison, on Bitcoin (another popular digital currency)the Stack Machine is a Turing incomplete machine. For the EVM, all execution processes are limited to a finite number of computational steps by the amount of gas available for any given smart contract execution. Every opcode (for example ADD) has a gas cost, and contracts can be set a gas limit.

EVM is designed in stack-based architecture, with a word size of 256-bits. The components that stored information on the EVM are divided into 3 parts:

* An immutable program code ROM, loaded with the bytecode of the smart contract to be executed
* A volatile memory, with every location explicitly initialized to zero
* A permanent storage that is part of the Ethereum state, also zero-initialized

The EVM opcodes are listed below, and a detailed table in the Appendices with their respective EVM bytecodes

:

**Mathematical processing**

ADD

MUL

SUB

DIV

SDIV

MOD

SMOD

ADDMOD

MULMOD

EXP

SIGNEXTEND

SHA3

**Register interactions**

STOP

JUMP

JUMPI

PC

JUMPDEST

**Interactive Commands with stack**

POP

MLOAD

MSTORE

MSTORE8

SLOAD

SSTORE

MSIZE

PUSHx

DUPx

SWAPx

**Command to interact with block**

BLOCKHASH

COINBASE

TIMESTAMP

NUMBER

DIFFICULTY

GASLIMIT

**System commands**

LOGx

CREATE

CALL

CALLCODE

RETURN

DELEGATECALL

STATICCALL

REVERT

INVALID

SELFDESTRUCT

**Logic**

LT

GT

SLT

SGT

EQ

ISZERO

AND

OR

XOR

NOT

BYTE

**Environment**

GAS

ADDRESS

BALANCE

ORIGIN

CALLER

CALLVALUE

CALLDATALOAD

CALLDATASIZE

CALLDATACOPY

CODESIZE

CODECOPY

GASPRICE

EXTCODESIZE

EXTCODECOPY

RETURNDATASIZE

RETURNDATACOPY

**Command to interact with block**

BLOCKHASH

COINBASE

TIMESTAMP

NUMBER

DIFFICULTY

GASLIMIT

### What is Decompilation

Decompilation is the process of converting object code (i.e. executables) to a form of high-level programming that is readable by humans. In this project, the Ethereum virtual machine compiles hexadecimals that all correspond to commands (for example, 01 is the ADD command). This research project converts these commands to a C program. The C program must be executable since we will be feeding it through CPROVER tools that look for bugs in C programs. This adds a higher complexity to the project, as I will need to model the Ethereum virtual machine in C code so that it will run, and the generated code cannot be pseudo code.

Decompilation is a type of reverse engineering that performs the opposite operations of a compiler, as shown by the diagram bellow. Decompiling goes from high-level to low-level programming languages, whereas compilers convert high-level programs to binary, thus fitting the name “reverse” engineering. Compilers allow high level languages like solidity and C to be understood and run by the computer.

This project is to program a decompiler, which will translate EVM byte code to C.

### C Prover Tools

CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by gcc (GCC is The GNU Compiler Collection – a compiler produced by the GNU that can be used for C code) and Visual Studio.

CBMC verifies memory safety (which includes array bounds checks and checks for the safe use of pointers), checks for exceptions, checks for various variants of undefined behavior, and user-specified assertions. The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure.

This project will be investigating whether these tools are helpful in identifying any bugs in EVM byte code, but requires the code to be in C.

### State of the art for Verification of EVM

EVM is binary, and there exists a high-level programming language called Solidity that allows for easier human readable development of EVM. I investigated what else is out there for that top link. Currently, verification of EVM and Solidity (“competitors” of my approach) that directly do verification include :

* Decompilers that translate EVM to pseudo solidity code(such as <https://www.ethervm.io/decompile> )
* Decompilers that extract information from EVM bytecode ( e.g. <https://www.npmjs.com/package/evm>)
* Decompilers that create human readable code and aim to look for vulnerabilities (<https://www.trustlook.com/services/smart.html>, <https://github.com/nevillegrech/gigahorse-toolchain>)
* IDE’s that allow people to program using high level languages such as solidity, and compile contracts without having to worry about the byte code itself (for example, remix, truffle , etc.)

Generally identification of obvious bugs (i.e. syntax errors) can be found by IDEs such as truffle and remix. However when it comes to the bytecode itself, it may be useful to test the logic of the bytecode itself, to see whether there are underlying issues with the EVM commands that are generated. Also, some developers of smart contracts do not use IDEs and instead program solidity using Inline Assembly (Yul is the language). This would be very useful for that purpose.

### Other Decompilers that have similar elements

I had a look at JVM decompilers for comparison, as JVM and EVM can technically be compared (as they are both binary i.e., executables) and existing research used for decompiling JVM could be useful for my project.

The JVM is designed to provide a runtime environment that works irrespective of the underlying host OS or hardware, enabling compatibility across many systems. High-level programming languages such as Java or Scala (which use the JVM) are compiled into the bytecode instruction set of their respective virtual machine. In the same way, the EVM executes its own bytecode, which higher-level smart contract programming languages such as LLL, Serpent or Solidity are compiled into.

Unlike JVM bytecode, EVM does not have the notion of structs or objects, nor does it have a concept of methods.

In JVM bytecode, stack depth is fixed under different control-flow paths: execution cannot reach the same program point with different stack sizes. In the EVM bytecode, no such execution constraints exist, which make the identification of standard control-flow constructs very hard.

All control-flow edges (i.e., jumps) are to variables, not constants. The destination of a jump is a value that is read from the stack, making it difficult to determine what’s happening with the byte code. In contrast, JVM bytecode has a clearly defined set of targets for every jump.

JVM bytecode has defined method invocation and return instructions. In EVM bytecode, function calls inside a contract are translated to just jumps. To call a function, the code pushes a return address to the stack, pushes arguments, pushes the destination block’s identifier (a hash), and performs a jump (which pops the top stack element, to use it as a jump destination). To return, the code pops the caller’s basic block identifier from the stack and jumps to it.

### Type of algorithms and architecture required

The best way to approach the modelling of this problem would be to use visitor methods , as I will be programming using Java. Language processing is a highly related topic that uses the same method. This will be the design pattern I will be primarily using when processing the EVM bytecode. The purpose of a Visitor pattern is to define a new operation without introducing the modifications to an existing object structure, therefore I can safely access parts of my objects without altering them. This requires me to add functions that accept a visitor class to each element of the structure of my program.

The positive side of this is that there will be no dependency on my component’s interfaces, and if they are different, that's fine, since I will have a separate algorithm for processing per concrete element. These elements will be each opcode, and their respective details. However it requires me to make a visitor function for each opcode, which is a lengthy process- but the safety of using visitors rather than directly interfering with my objects is worth it.

UML diagram: <https://www.baeldung.com/java-visitor-pattern>

Diagram

Description automatically generated

The main problem that I am investigating is to see whether CPROVER tools can identify problems with EVM byte code, which if successful could aid in increasing the security of smart contracts. The Ethereum cryptocurrency has had issues with security due to malicious contracts, since the currency is stored on a blockchain which is open to everyone. Any improvement in the protection of the currency can save people a lot of money, as it will prevent people manipulation smart contracts that allow access to other people’s accounts.

This open nature smart contracts make the analysis and validation of contracts essential, therefore it will be useful to see if we can use existing tools (i.e. CPROVER) used for other languages. The task of Decompilation is hindered by the low-level stack-based design of the EVM bytecode that has hardly any abstractions as found as in other languages, such as Java’s virtual machine. For example, there is no notion of functions or call. This means that a decompiler that translates EVM bytecode needs to invent its own conventions for implementing local calls over the stack.

The difficulties of the project are that once a contract is in EVM byte code, the reconstruction back to the original program is near impossible, as the language is simplified so that the program can be run by the computer. Furthermore, converting this to C (which is a completely different language that does not have connection to EVM) requires me to develop equivalent programs for each EVM command. The first step towards this is to simulate the EVM stack itself. If I was to attempt to reconstruct more than this, it would require extensive knowledge of creating a CFG graph.

Some CFG examples:
(a) an if-then-else
(b) a while loopA control-flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution. This is usually done by hand, and can aid in identifying uses of if statements, loops and other high-level programming techniques. The way to do this would be to split the bytecode into blocks where JUMP commands occur and to draw out the edges that connect these jump commands. This will correspond to an existing graph that symbolises either an if statement, while loop etc. In theory this sounds like a simple task, however programming this concept is extremely difficult, and a lot of research projects have been based around accurate CFG reconstruction.

See some examples of CFG graphs to the side, a) representing an if -then-else statement, and b) displaying the relevant edges of a while loop:

# Chapter 4

## 4.1 Methodology

### 4.1.1 Iterative Development

I have decided to use an Iterative development methodology given the time I have to complete this project. (Define the method here -state which one you followed and detail what was specific to your own project. Readers are interested in how you applied the standard methods, how you deviated from them and why; and the history of your project, with the main decisions and changes of plans.)

I will be splitting my project into three builds and will have **numbered objectives** for each:

1. **Minimum Viable Product:** Completed in March:

* Research all existing implementations of similar projects (4 days)
* Write-up report (4 days)
* The program will take some EVM code and decompile it , translate into a list of Op code (or a CGF) (2 weeks)
* Get client feedback, test that this product has no syntax or logical errors. (Client proposed the method could be structured better , these changes were made.)

1. **The Main Product:** functional but with no extra subsets of op code: Complete by mid-April :

* Extend Report with updates (3-4 days)
* The program will take the decompiled Op code and generate reliable C code AND will successfully be usable by CPROVER tests (3+weeks)
* Get client feedback, test that this product has no syntax or logical errors. (2-3 hours)

1. **Additional Features:** Complete towards the start of May:

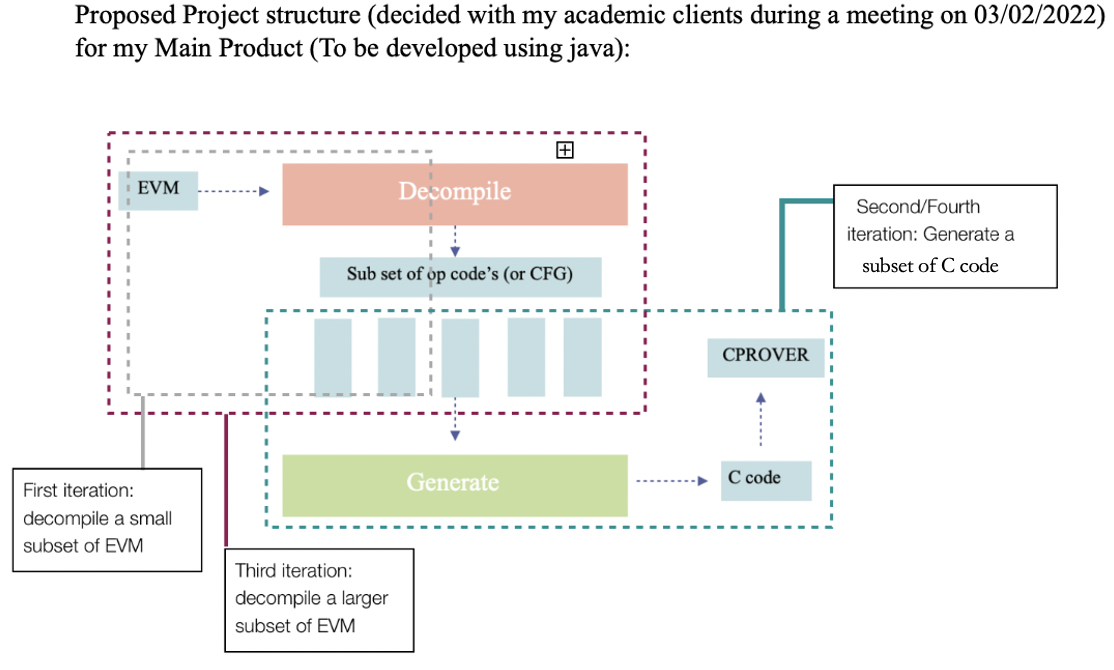
* Look at the behaviour of the decompiled C program and add additional features based on the main product’s current output(1+week(s))
* Add additional op code features (extending the decompiler and code generator made in the first two iterations) (2+ weeks)

Refer to this iterative workflow diagram – Each stage must go through testing or a client meeting before moving on to the next step:

Diagram

Description automatically generated

#### Project Plan: Project Structure



### 4.2 Disassembling EVM

#### 4.2.1 Analysis

#### 4.2.2 Design

Made separate classes for each opcode, interface and visitor functions. Main code receives a user i

#### 4.2.3 Implementation

Using Ethereum’s yellow paper I made a table of each stack number (hexadecimals (i.e. the stack column) correspond to the EVM binaries.) From this I tracked my implementation for the disassembly of EVM binary, to its respective function (see appendix)

#### 4.2.4 Evaluation

### 4.3 Decomplication to C

#### 4.3.1 Analysis

#### 4.3.2 Design

Extended previous classes of opcodes added a c code variable, and computed this in the visitor functions, for each individual op code.

Outputs the code, also gives an option to save as a c file, for ease of cprover testing, as cprover requires you to enter the file you want to validate

#### 4.3.3 Implementation

#### 4.3.4 Evaluation

# Chapter 5

## Results

The Results chapter must present ALL of the results produced during your project. This includes

outputs from analysis

design

prototypes

experiment

evaluation and other activities.

Every activity reported in the Method chapter should deliver results that are reported in the Results chapter.

Be prepared to state the obvious (no detail is too small for the Results chapter and the relevant Appendices).

For a research project there may be sections about the data collected and the analyses that were conducted. Be very clear in identifying what data or other resources you had at the beginning of the project, which ones you collected or created and which you derived in your further analysis. The results should be traceable to descriptions of each output in the Output Summary chapter.

Should you include pieces of source code, design, specification, requirements diagrams, in the Results chapter? This is a frequently asked question. The body of the report must be such that readers can understand the main characteristics of the work done, without the help of the appendices. So, among all the material that you could include in the Results, you must choose on the basis of importance. Those items, or selected parts of those items, that are most important for understanding the work will usually be in the main body of the report. They may also be repeated in full in the appendices, so that readers can see the complete output. For instance, the main body of a report might include:

• a specific sequence diagram, because it clarifies the way a certain essential protocol works;

• a specific class diagram, because it clarifies some important relationships among data items in the state of the software;

• selected requirements that are important and distinctive for your software;

• snippets of source code that are important for understanding your software, or represent important parts of your original work (e.g., algorithms you invented).

The reported results must be a significant portion of your report. We recommend that the Results chapter should be at least 6000 words for BSc

Conclusions and Discussion

The Conclusions and Discussion chapter(s) tie-up and discuss all of the project themes. The first thing that you want to do in this chapter is to revisit the project’s objectives and, if applicable, research questions and hypotheses. In a successful project you will want to demonstrate to what extent each of the objectives have been met, the research questions answered (if any), and the research hypotheses accepted or refuted (if any). You will then want to draw more general conclusions from the above. A good project should also discuss the conclusions drawn from the objectives, research questions and research hypotheses. This might happen in 2 parts. Firstly, each conclusion is discussed in its own right. Secondly you should link each conclusion to the literature in the Literature Review chapter, presenting the contributions that your project work has made to knowledge about the area and comparing this to previous work by other people. The chapter might also discuss implications for future work in the area (perhaps in terms of a future project). Finally, you should also discuss your own progress in the project, including the management and control of the project, what you have learned, the challenges that you faced and how you reacted to them, as well as what you might do differently if you had your time again.

7.2.8 Glossary

You may need to define specialist terms, unlikely to be known by your intended audience, in a project glossary appended at the end of the project report.

7.2.10 Appendices

Any material that would interrupt reading of the report should be presented in the Appendices. These should include all data used in producing the results of your project and all outputs produced that are not part of the main report. The following should be included:

• Appendix A: Project Definition Document (this must always be Appendix A);

• Appendix B: Reuse Summary (this must always be Appendix B, see Section 8.6);

• Complete records of each interview; questionnaires and complete questionnaire replies;

• Requirements;

• Routine design documentation;

• Source code, with instructions for building an executable version;

• Test plans and test results; output listings; displays etc.;

• Any software installation guides and user guides produced;

• An executable version of your software with installation instructions, or a URL if it is a web-based application that can be tested online;

• All the reports produced for a client

• Web pages related to your project

Rule: complete appendices. All data used in producing the results of your project must be included in the Appendices. Failure to do so will raise concerns about the validity of the results reported: you may be called before the Academic Misconduct Panel to prove that you really did the work described in your report. Therefore, include all raw and processed data in the Appendices of the report, and be careful not to delete or lose your own copies of data before the completion of the final year of your degree programme (that is, until you receive the official letter of results from the last Assessment Board concerning you).

Reference List

*Harvard style referencing was used for the books, articles, videos and website links bellow:*

* ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. (n.d.). [online] Available at: <https://ethereum.github.io/yellowpaper/paper.pdf>.
* The Guardian. (2017). “$300m in cryptocurrency” accidentally lost forever due to bug. [online] Available at: <https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether>
* Hollander, L. (2022). EVM Bytecode Decompiler. [online] GitHub. Available at: <https://github.com/MrLuit/evm>
* Ethereum Stack Exchange. (n.d.). EVM - How can you decompile a smart contract? [online] Available at: <https://ethereum.stackexchange.com/questions/188/how-can-you-decompile-a-smart-contract>
* ethereum.org. (n.d.). Introduction to the Ethereum stack. [online] Available at: <https://ethereum.org/en/developers/docs/ethereum-stack/#ethereum-client-apis>
* www.cprover.org. (n.d.). The CPROVER Manual. [online] Available at: <http://www.cprover.org/cprover-manual/cbmc/tutorial/>
* GitHub. (2022). rattle. [online] Available at: <https://github.com/crytic/rattle>
* Online Solidity Decompiler. (n.d.). Online Solidity Decompiler. [online] Available at: <https://www.ethervm.io/decompile>
* ethereum.org. (n.d.). Ethereum development documentation. [online] Available at: <https://ethereum.org/en/developers/docs/>
* Aho, A.V., Ravi Sethi and Ullman, J.D. (2002). Compilers : Principles, Techniques, and Tools. Reading, Mass.: Addison-Wesley.
* OpenZeppelin blog. (2018). Deconstructing a Solidity Contract — Part III: The Function Selector. [online] Available at: <https://blog.openzeppelin.com/deconstructing-a-solidity-contract-part-iii-the-function-selector-6a9b6886ea49/>
* OpenZeppelin blog. (2018). Deconstructing a Solidity Contract —Part I: Introduction. [online] Available at: <https://blog.openzeppelin.com/deconstructing-a-solidity-contract-part-i-introduction-832efd2d7737/>
* www.youtube.com. (n.d.). Ethereum/EVM Smart Contract Reverse Engineering & Disassembly - Blockchain Security #3. [online] Available at: <https://www.youtube.com/watch?v=I6VDBvX9Pkw&t=41s>
* ethereum.org. (n.d.). Reverse Engineering a Contract. [online] Available at: <https://ethereum.org/en/developers/tutorials/reverse-engineering-a-contract/>
* FreddyCoen (2022). EVM development Starter Kit. [online] Medium. Available at: <https://freddycoen.medium.com/evm-starter-kit-1790bcc992ef>
* www.youtube.com. (n.d.). EVM: From Solidity to byte code, memory and storage. [online] Available at: <https://www.youtube.com/watch?v=RxL_1AfV7N4>
* Unix & Linux Stack Exchange. (n.d.). convert executable back to C source code. [online] Available at: <https://unix.stackexchange.com/questions/229802/convert-executable-back-to-c-source-code>
* readthedocs.org. (n.d.). Solidity | Read the Docs. [online] Available at: <https://readthedocs.org/projects/solidity/>
* TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEMS : 28th. (2022). S.L.: Springer Nature.
* ‌Cassez, F., Fuller, J. and Asgaonkar, A. (2022). Formal Verification of the Ethereum 2.0 Beacon Chain. Tools and Algorithms for the Construction and Analysis of Systems, pp.167–182.
* Bytecode and transactions . (n.d.). [online] Available at: <https://cse.iitk.ac.in/users/dwivedi/Blockchain/bytecode.pdf>‌
* Hollander, L. (2019). *The Ethereum Virtual Machine — How does it work?* [online] Medium. Available at: <https://medium.com/mycrypto/the-ethereum-virtual-machine-how-does-it-work-9abac2b7c9e>.
* En.wikipedia.org. 2022. *Control-flow graph - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Control-flow_graph>
* cypherpunks-core.github.io. (n.d.). *Chapter 13: The Ethereum Virtual Machine · GitBook*. [online] Available at: <https://cypherpunks-core.github.io/ethereumbook/13evm.html#evm_architecture>
* Mochi.Market (2021). *Learn about Ethereum Virtual Machine*. [online] Mochilab. Available at: <https://medium.com/mochilab/learn-about-ethereum-virtual-machine-de0a574b13ba>
* www.cprover.org. (n.d.). *The CBMC Homepage*. [online] Available at: <https://www.cprover.org/cbmc/>.
* Szczukocki, D. (2018). *Visitor Design Pattern in Java | Baeldung*. [online] www.baeldung.com. Available at: <https://www.baeldung.com/java-visitor-pattern>.

APENDIX STUFF:

Table of opcodes:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Implemented on | **Stack** | **Name** | **Gas** | **Initial Stack** | **Resulting Stack** | **Mem / Storage** | **Notes** |
| 05/03 | 00 | STOP | 0 |  |  |  | halt execution |
| 05/03 | 01 | ADD | 3 | a, b | a + b |  | (u)int256 addition modulo 2\*\*256 |
| 05/03 | 02 | MUL | 5 | a, b | a \* b |  | (u)int256 multiplication modulo 2\*\*256 |
| 05/03 | 03 | SUB | 3 | a, b | a - b |  | (u)int256 addition modulo 2\*\*256 |
| 05/03 | 04 | DIV | 5 | a, b | a // b |  | uint256 division |
| 05/03 | 05 | SDIV | 5 | a, b | a // b |  | int256 division |
| 05/03 | 06 | MOD | 5 | a, b | a % b |  | uint256 modulus |
| 05/03 | 07 | SMOD | 5 | a, b | a % b |  | int256 modulus |
| 05/03 | 08 | ADDMOD | 8 | a, b, N | (a + b) % N |  | (u)int256 addition modulo N |
| 05/03 | 09 | MULMOD | 8 | a, b, N | (a \* b) % N |  | (u)int256 multiplication modulo N |
|  | 0A | EXP | [A1](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a1-exp) | a, b | a \*\* b |  | uint256 exponentiation modulo 2\*\*256 |
|  | 0B | SIGNEXTEND | 5 | b, x | SIGNEXTEND(x, b) |  | [sign extend](https://wikipedia.org/wiki/Sign_extension) x from (b+1) bytes to 32 bytes |
|  | 0C-0F | *invalid* |  |  |  |  |  |
| 05/03 | 10 | LT | 3 | a, b | a < b |  | uint256 less-than |
| 05/03 | 11 | GT | 3 | a, b | a > b |  | uint256 greater-than |
| 05/03 | 12 | SLT | 3 | a, b | a < b |  | int256 less-than |
| 05/03 | 13 | SGT | 3 | a, b | a > b |  | int256 greater-than |
| 05/03 | 14 | EQ | 3 | a, b | a == b |  | (u)int256 equality |
| 05/03 | 15 | ISZERO | 3 | a | a == 0 |  | (u)int256 iszero |
| 05/03 | 16 | AND | 3 | a, b | a && b |  | bitwise AND |
| 05/03 | 17 | OR | 3 | a, b | a \|\| b |  | bitwise OR |
| 05/03 | 18 | XOR | 3 | a, b | a ^ b |  | bitwise XOR |
| 05/03 | 19 | NOT | 3 | a | ~a |  | bitwise NOT |
|  | 1A | BYTE | 3 | i, x | (x >> (248 - i \* 8)) && 0xFF |  | ith byte of (u)int256 x, from the left |
|  | 1B | SHL | 3 | shift, val | val << shift |  | shift left |
|  | 1C | SHR | 3 | shift, val | val >> shift |  | logical shift right |
|  | 1D | SAR | 3 | shift, val | val >> shift |  | arithmetic shift right |
|  | 1E-1F | *invalid* |  |  |  |  |  |
|  | 20 | SHA3 | [A2](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a2-sha3) | ost, len | keccak256(mem[ost:ost+len]) |  | keccak256 |
|  | 21-2F | *invalid* |  |  |  |  |  |
| 05/03 | 30 | ADDRESS | 2 | . | address(this) |  | address of executing contract |
| 05/03 | 31 | BALANCE | [A5](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a5-balance-extcodesize-extcodehash) | addr | addr.balance |  | balance, in wei |
| 05/03 | 32 | ORIGIN | 2 | . | tx.origin |  | address that originated the tx |
| 05/03 | 33 | CALLER | 2 | . | msg.sender |  | address of msg sender |
| 05/03 | 34 | CALLVALUE | 2 | . | msg.value |  | msg value, in wei |
| 05/03 | 35 | CALLDATALOAD | 3 | idx | msg.data[idx:idx+32] |  | read word from msg data at index idx |
| 05/03 | 36 | CALLDATASIZE | 2 | . | len(msg.data) |  | length of msg data, in bytes |
| 05/03 | 37 | CALLDATACOPY | [A3](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a3-copy-operations) | dstOst, ost, len | . | mem[dstOst:dstOst+len] := msg.data[ost:ost+len | copy msg data |
| 05/03 | 38 | CODESIZE | 2 | . | len(this.code) |  | length of executing contract's code, in bytes |
| 05/03 | 39 | CODECOPY | [A3](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a3-copy-operations) | dstOst, ost, len | . |  | mem[dstOst:dstOst+len] := this.code[ost:ost+len]  copy executing contract's bytecode |
|  | 3A | GASPRICE | 2 | . | tx.gasprice |  | gas price of tx, in wei per unit gas [\*\*](https://github.com/ethereum/EIPs/blob/0341984ff14c8ce398f6d2b3e009c07cd99df8eb/EIPS/eip-1559.md#gasprice) |
|  | 3B | EXTCODESIZE | [A5](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a5-balance-extcodesize-extcodehash) | addr | len(addr.code) |  | size of code at addr, in bytes |
|  | 3C | EXTCODECOPY | [A4](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a4-extcodecopy) | addr, dstOst, ost, len | . | mem[dstOst:dstOst+len] := addr.code[ost:ost+len] | copy code from addr |
|  | 3D | RETURNDATASIZE | 2 | . | size |  | size of returned data from last external call, in bytes |
|  | 3E | RETURNDATACOPY | [A3](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a3-copy-operations) | dstOst, ost, len | . | mem[dstOst:dstOst+len] := returndata[ost:ost+len] | copy returned data from last external call |
|  | 3F | EXTCODEHASH | [A5](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a5-balance-extcodesize-extcodehash) | addr | hash |  | hash = addr.exists ? keccak256(addr.code) : 0 |
| 05/03 | 40 | BLOCKHASH | 20 | blockNum | blockHash(blockNum) |  |  |
| 05/03 | 41 | COINBASE | 2 | . | block.coinbase |  | address of miner of current block |
| 05/03 | 42 | TIMESTAMP | 2 | . | block.timestamp |  | timestamp of current block |
| 05/03 | 43 | NUMBER | 2 | . | block.number |  | number of current block |
| 05/03 | 44 | DIFFICULTY | 2 | . | block.difficulty |  | difficulty of current block |
| 05/03 | 45 | GASLIMIT | 2 | . | block.gaslimit |  | gas limit of current block |
| 05/03 | 46 | CHAINID | 2 | . | chain\_id |  | push current [chain id](https://eips.ethereum.org/EIPS/eip-155) onto stack |
| 05/03 | 47 | SELFBALANCE | 5 | . | address(this).balance |  | balance of executing contract, in wei |
| 05/03 | 48 | BASEFEE | 2 | . | block.basefee |  | base fee of current block |
|  |  |  |  |  |  |  |  |
|  | 49-4F | *invalid* |  |  |  |  |  |
| 05/03 | 50 | POP | 2 | \_anon | . |  | remove item from top of stack and discard it |
| 05/03 | 51 | MLOAD | 3[\*](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a0-1-memory-expansion) | ost | mem[ost:ost+32] |  | read word from memory at offset ost |
| 05/03 | 52 | MSTORE | 3[\*](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a0-1-memory-expansion) | ost, val | . | mem[ost:ost+32] := val | write a word to memory |
| 05/03 | 53 | MSTORE8 | 3[\*](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a0-1-memory-expansion) | ost, val | . | mem[ost] := val && 0xFF | write a single byte to memory |
| 05/03 | 54 | SLOAD | [A6](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a6-sload) | key | storage[key] |  | read word from storage |
| 05/03 | 55 | SSTORE | [A7](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a7-sstore) | key, val | . | storage[key] := val | write word to storage |
| 05/03 | 56 | JUMP | 8 | dst | . |  | $pc := dst mark that pc is only assigned if dst is a valid jumpdest |
| 05/03 | 57 | JUMPI | 10 | dst, condition | . |  | $pc := condition ? dst : $pc + 1 |
| 05/03 | 58 | PC | 2 | . | $pc |  | program counter |
| 05/03 | 59 | MSIZE | 2 | . | len(mem) |  | size of memory in current execution context, in bytes |
|  | 5A | GAS | 2 | . | gasRemaining |  |  |
|  | 5B | JUMPDEST | 1 |  |  | mark valid jump destination | a valid jump destination for example a jump destination not inside the push data |
|  | 5C-5F | *invalid* |  |  |  |  |  |
|  | 60 | PUSH1 | 3 | . | uint8 |  | push 1-byte value onto stack |
|  | 61 | PUSH2 | 3 | . | uint16 |  | push 2-byte value onto stack |
|  | 62 | PUSH3 | 3 | . | uint24 |  | push 3-byte value onto stack |
|  | 63 | PUSH4 | 3 | . | uint32 |  | push 4-byte value onto stack |
|  | 64 | PUSH5 | 3 | . | uint40 |  | push 5-byte value onto stack |
|  | 65 | PUSH6 | 3 | . | uint48 |  | push 6-byte value onto stack |
|  | 66 | PUSH7 | 3 | . | uint56 |  | push 7-byte value onto stack |
|  | 67 | PUSH8 | 3 | . | uint64 |  | push 8-byte value onto stack |
|  | 68 | PUSH9 | 3 | . | uint72 |  | push 9-byte value onto stack |
|  | 69 | PUSH10 | 3 | . | uint80 |  | push 10-byte value onto stack |
|  | 6A | PUSH11 | 3 | . | uint88 |  | push 11-byte value onto stack |
|  | 6B | PUSH12 | 3 | . | uint96 |  | push 12-byte value onto stack |
|  | 6C | PUSH13 | 3 | . | uint104 |  | push 13-byte value onto stack |
|  | 6D | PUSH14 | 3 | . | uint112 |  | push 14-byte value onto stack |
|  | 6E | PUSH15 | 3 | . | uint120 |  | push 15-byte value onto stack |
|  | 6F | PUSH16 | 3 | . | uint128 |  | push 16-byte value onto stack |
|  | 70 | PUSH17 | 3 | . | uint136 |  | push 17-byte value onto stack |
|  | 71 | PUSH18 | 3 | . | uint144 |  | push 18-byte value onto stack |
|  | 72 | PUSH19 | 3 | . | uint152 |  | push 19-byte value onto stack |
|  | 73 | PUSH20 | 3 | . | uint160 |  | push 20-byte value onto stack |
|  | 74 | PUSH21 | 3 | . | uint168 |  | push 21-byte value onto stack |
|  | 75 | PUSH22 | 3 | . | uint176 |  | push 22-byte value onto stack |
|  | 76 | PUSH23 | 3 | . | uint184 |  | push 23-byte value onto stack |
|  | 77 | PUSH24 | 3 | . | uint192 |  | push 24-byte value onto stack |
|  | 78 | PUSH25 | 3 | . | uint200 |  | push 25-byte value onto stack |
|  | 79 | PUSH26 | 3 | . | uint208 |  | push 26-byte value onto stack |
|  | 7A | PUSH27 | 3 | . | uint216 |  | push 27-byte value onto stack |
|  | 7B | PUSH28 | 3 | . | uint224 |  | push 28-byte value onto stack |
|  | 7C | PUSH29 | 3 | . | uint232 |  | push 29-byte value onto stack |
|  | 7D | PUSH30 | 3 | . | uint240 |  | push 30-byte value onto stack |
|  | 7E | PUSH31 | 3 | . | uint248 |  | push 31-byte value onto stack |
|  | 7F | PUSH32 | 3 | . | uint256 |  | push 32-byte value onto stack |
|  | 80 | DUP1 | 3 | a | a, a |  | clone 1st value on stack |
|  | 81 | DUP2 | 3 | \_, a | a, \_, a |  | clone 2nd value on stack |
|  | 82 | DUP3 | 3 | \_, \_, a | a, \_, \_, a |  | clone 3rd value on stack |
|  | 83 | DUP4 | 3 | \_, \_, \_, a | a, \_, \_, \_, a |  | clone 4th value on stack |
|  | 84 | DUP5 | 3 | ..., a | a, ..., a |  | clone 5th value on stack |
|  | 85 | DUP6 | 3 | ..., a | a, ..., a |  | clone 6th value on stack |
|  | 86 | DUP7 | 3 | ..., a | a, ..., a |  | clone 7th value on stack |
|  | 87 | DUP8 | 3 | ..., a | a, ..., a |  | clone 8th value on stack |
|  | 88 | DUP9 | 3 | ..., a | a, ..., a |  | clone 9th value on stack |
|  | 89 | DUP10 | 3 | ..., a | a, ..., a |  | clone 10th value on stack |
|  | 8A | DUP11 | 3 | ..., a | a, ..., a |  | clone 11th value on stack |
|  | 8B | DUP12 | 3 | ..., a | a, ..., a |  | clone 12th value on stack |
|  | 8C | DUP13 | 3 | ..., a | a, ..., a |  | clone 13th value on stack |
|  | 8D | DUP14 | 3 | ..., a | a, ..., a |  | clone 14th value on stack |
|  | 8E | DUP15 | 3 | ..., a | a, ..., a |  | clone 15th value on stack |
|  | 8F | DUP16 | 3 | ..., a | a, ..., a |  | clone 16th value on stack |
|  | 90 | SWAP1 | 3 | a, b | b, a |  |  |
|  | 91 | SWAP2 | 3 | a, \_, b | b, \_, a |  |  |
|  | 92 | SWAP3 | 3 | a, \_, \_, b | b, \_, \_, a |  |  |
|  | 93 | SWAP4 | 3 | a, \_, \_, \_, b | b, \_, \_, \_, a |  |  |
|  | 94 | SWAP5 | 3 | a, ..., b | b, ..., a |  |  |
|  | 95 | SWAP6 | 3 | a, ..., b | b, ..., a |  |  |
|  | 96 | SWAP7 | 3 | a, ..., b | b, ..., a |  |  |
|  | 97 | SWAP8 | 3 | a, ..., b | b, ..., a |  |  |
|  | 98 | SWAP9 | 3 | a, ..., b | b, ..., a |  |  |
|  | 99 | SWAP10 | 3 | a, ..., b | b, ..., a |  |  |
|  | 9A | SWAP11 | 3 | a, ..., b | b, ..., a |  |  |
|  | 9B | SWAP12 | 3 | a, ..., b | b, ..., a |  |  |
|  | 9C | SWAP13 | 3 | a, ..., b | b, ..., a |  |  |
|  | 9D | SWAP14 | 3 | a, ..., b | b, ..., a |  |  |
|  | 9E | SWAP15 | 3 | a, ..., b | b, ..., a |  |  |
|  | 9F | SWAP16 | 3 | a, ..., b | b, ..., a |  |  |
|  | A0 | LOG0 | [A8](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a8-log-operations) | ost, len | . |  | LOG0(memory[ost:ost+len]) |
|  | A1 | LOG1 | [A8](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a8-log-operations) | ost, len, topic0 | . |  | LOG1(memory[ost:ost+len], topic0) |
|  | A2 | LOG2 | [A8](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a8-log-operations) | ost, len, topic0, topic1 | . |  | LOG1(memory[ost:ost+len], topic0, topic1) |
|  | A3 | LOG3 | [A8](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a8-log-operations) | ost, len, topic0, topic1, topic2 | . |  | LOG1(memory[ost:ost+len], topic0, topic1, topic2) |
|  | A4 | LOG4 | [A8](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a8-log-operations) | ost, len, topic0, topic1, topic2, topic3 | . |  | LOG1(memory[ost:ost+len], topic0, topic1, topic2, topic3) |
|  | A5-EF | *invalid* |  |  |  |  |  |
|  | F0 | CREATE | [A9](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a9-create-operations) | val, ost, len | addr |  | addr = keccak256(rlp([address(this), this.nonce])) |
|  | F1 | CALL | [AA](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#aa-call-operations) | gas, addr, val, argOst, argLen, retOst, retLen | success | mem[retOst:retOst+retLen] := returndata |  |
|  | F2 | CALLCODE | [AA](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#aa-call-operations) | gas, addr, val, argOst, argLen, retOst, retLen | success | mem[retOst:retOst+retLen] = returndata | same as DELEGATECALL, but does not propagate original msg.sender and msg.value |
|  | F3 | RETURN | 0[\*](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a0-1-memory-expansion) | ost, len | . |  | return mem[ost:ost+len] |
|  | F4 | DELEGATECALL | [AA](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#aa-call-operations) | gas, addr, argOst, argLen, retOst, retLen | success | mem[retOst:retOst+retLen] := returndata |  |
|  | F5 | CREATE2 | [A9](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a9-create-operations) | val, ost, len, salt | addr |  | addr = keccak256(0xff ++ address(this) ++ salt ++ keccak256(mem[ost:ost+len]))[12:] |
|  | F6-F9 | *invalid* |  |  |  |  |  |
|  | FA | STATICCALL | [AA](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#aa-call-operations) | gas, addr, argOst, argLen, retOst, retLen | success | mem[retOst:retOst+retLen] := returndata |  |
|  | FB-FC | *invalid* |  |  |  |  |  |
|  | FD | REVERT | 0[\*](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a0-1-memory-expansion) | ost, len | . |  | revert(mem[ost:ost+len]) |
|  | FE | INVALID | [AF](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#af-invalid) |  |  | designated invalid opcode - [EIP-141](https://eips.ethereum.org/EIPS/eip-141) |  |
|  | FF | SELFDESTRUCT | [AB](https://github.com/wolflo/evm-opcodes/blob/main/gas.md#ab-selfdestruct) | addr |  |  | destroy contract and sends all funds to addr |