CS 168: Blockchain and Cryptocurrencies

Ethereum Virtual Machine (EVM)

Prof. Tom Austin
San José State University

Virtual Machine (VM)

- Code is compiled to *bytecode*
  - low-level
  - platform independent
- The VM interprets bytecode
- Examples: JVM, CLR, Parrot

Sample VM Operations

- **PUSH1** – adds argument to stack
- **PRINT** – pops & prints top of stack
- **ADD**
  - pops top two elements
  - adds them together
  - places result on stack
- **SUB** – subtraction
- **MUL** – multiplication
- **SWAP** – swaps top two arguments on the stack

Bytecode Representation

```
PUSH1 2
PUSH1 3
ADD
PUSH1 4
ADD
PRINT
PUSH1 13
PUSH1 2
PUSH1 4
MUL
SWAP1
SUB
PRINT
PUSH1 10
PUSH1 4
SUB
PUSH1 3
SUB
PRINT
```

Ethereum Virtual Machine (EVM)

- Stack-based
- Similar to JVM
- Quasi-Turing-complete
  - Execution limited to a finite number of steps

What is it composed of?

- Volatile memory
- Permanent storage
- Immutable ROM
  - stores contracts

Ethereum State

- Ethereum world state
- Account state
  - Ether balance
  - Nonce
  - Storage (only smart contracts)
  - Program code (only smart contracts)

Viewing Bytecode with Remix IDE

(in class)

Walking through bytecode execution

(in class)

PART 1 -- Run GLEAM

Test out gleam by running:

```
$ node driver.js test.gleam
```

If everything is set up correctly, it should print '12'.

PART 2 -- Add Opcodes to GLEAM

A slightly more complex example is available in test2.gleam.

Add the `MUL`, `SUB`, and `SWAP1` opcodes to `op-codes.js`.

Details are available from https://ethervm.io/ and https://ethereum.org/en/developers/docs/evm/opcodes/.

Gas

- Gas limit
- Gas cost
- Gas price ether/gas used
- Tx fee = total gas used * gas price
- Excess gas refunded

Lab, part 3 — Track gas usage

Call: `node driver.js test.gleam 0`. The program will run, despite not having any available gas.

Update the code to track gas usage for each instruction. If the available gas is exceeded, throw an error.

Verify that the VM runs test.gleam with sufficient gas, but not with insufficient gas.

Walking through bytecode execution, involving memory and jumps

(in class)

PART 4 -- Extend GLEAM with memory

Add support for the **`MSTORE`** and **`MLOAD`** opcodes. We will simplify them so that they only load a ***byte***.

The gleam class has a "memory" array, where the offset matches to the index of the array. Test solution with `store.gleam`.

PART 5 -- Add JUMP instructions

Add support for **`JUMP`**, **`JUMPI`**, and **`JUMPDEST`**.

For simplicity, the address range will be one byte.

References for Further Reading

- *Mastering Ethereum*, chapter 13
  - https://github.com/ethereumbook/ethereumbook/blob/develop/13evm.asciidoc
- Trustlook's *Understand EVM bytecode*
  - https://blog.trustlook.com/understand-evm-bytecode-part-1/
  - https://blog.trustlook.com/understand-evm-bytecode-part-2/