Oil

Pre-EIP v1

Motivation

- Gas is currently being used for two different purposes:
    - To pay for compute, memory, and storage resources
    - To prevent re-entrancy by hardwiring the amount of gas a call can use.
- Adjusting the gas schedule to better reflect resource usage causes unintended consequences because contracts may be written in a way such that correctness depends on a specific gas schedule.
    - Making an instruction cheaper may make a re-entrancy path feasible
    - Making an instruction more expensive may make a call fail because the amount of gas hard-wired to it is now insufficient to execute the call
- Oil is a new fuel source that works very similarly to gas, but works in parallel to it.

Specification

- A transaction has a `gasLimit` and `gasPrice`.
- Currently, a transaction pays E ether for allocating `gasLimit` amount of gas to the transaction based on the `gasPrice`.
- With oil, a transaction pays E ether for allocating `gasLimit` amount of gas to the transaction based on the `gasPrice`, and additionally `oilLimit` amount of oil to the transaction where `oilLimit` is set equal to `gasLimit`.
- A transaction still only specifies a `gasLimit`. The EVM will internally set the `oilLimit` to be the same as the `gasLimit` specified by the transaction.
- Gas metering and gas semantics do not change.
- If the transaction runs out of oil at any point during execution, the transaction reverts. Unlike with gas, where out-of-gas reverts only the current frame, and lets the caller examine the result, out-of-oil always reverts the entire transaction (all frames).
- A caller contract cannot restrict how much oil a callee contract can use, unlike gas.
- The oil cost of all instructions is exactly the same as the gas cost, until further EIPs to modify oil schedule to reprice EVM operations.
- An `OIL` instruction to read current oil will not be added, and this is intentional.
- The amount of ETH refunded for a transaction is now calculated using the minimum of the unused oil and unused gas, rather than just unused gas.
    - If the transaction has an `EVMC_SUCCESS` status code, the sender is refunded the amount of ETH that is the minimum of the remaining gas and remaining oil in the state, exchanged at the `gasPrice`.
    - Similarly, if the transaction has an `EVMC_REVERT` status code, the state is reverted as usual, and the sender is refunded the amount of ETH that is the minimum of the remaining gas and remaining oil in the state, exchanged at the `gasPrice`.

Example

Consider the following two contracts where contract A is stored at address A_{ADDR} and contract B is stored at B_{ADDR}. Initially, let the gas cost of each instruction equal the oil cost of each instruction.

```
contract A {
    function set(B b) public {
        b.set();
    }
}
```

```
contract B {
    uint public amount;
    function set() public {
        amount = address(this).balance;
    }
}
```

Suppose $SENDER$ sends a transaction TX_1 to A_{ADDR} to invoke $A.set$ on B_{ADDR} with initial gas G_{init}, where G_{init} is set to *exactly* the gas cost of executing $A_{ADDR}.set(B_{ADDR})$. Then, the initial oil O_{init} would be equal to G_{init} and the transaction would be accepted.

Now, suppose the oil cost of the $BALANCE$ opcode is increased and that a TX_2 is sent that is identical to TX_1. This transaction TX_2 would get rejected with an out-of-oil error because the total oil cost would exceed O_{init}.