**THE CRAB VIRTUAL MACHINE**

**(CVM)**

**Authors:**

**StarloExoliz**

**OVERVIEW**

The Crabby Virtual Machine (CVM) is a **custom bytecode-based** execution **engine** designed to run *compiled* Crabby programs. It aims to be *fast, portable, and maintainable,* serving as the runtime core for Crabby’s future compiled mode **(crabc)**.

* Status: Early design phase. *This* document serves as a working draft of the architecture and concepts for CVM.

**1. ARCHITECTURE**

* Stack-Based Execution Model
  + CVM will operate using a value stack for computation.
  + Bytecode **instructions** manipulate the stack, much like the JVM or Lua VM.
* Registers (Optional)
* A simple register-based optimization layer may be introduced later.
* At minimum, and **Instruction Pointer** (IP) and **Call Stack** are maintained.
* Modules & Functions
* **Crabby** programs are compiled into modules with *function tables, constant pools, and symbol tables.*

**2. BYTECODE STRUCTURE**

Each compiled instruction **(opcode)** is represented in **binary/byte** format as:

[OPCODE] [OPERAND\_1] [OPERAND\_2] ...

* **Bytecode** is stored in a contiguous instruction list.
* **Operands** can represent:
* CONSTANTS
* STACK OFFSETS
* LABELS (JUMPS)
* FUNCTION INDICES
* MEMORY ADDRESSES (if memory segment is implemented)

**3. Sample OPCODES**

|  |  |  |  |
| --- | --- | --- | --- |
| **OPCODE** | **HEX** | **DESCRIPTION** | **OPERANDS** |
| LOAD\_CONST | **0x1** | Puts a constant to the stack | const\_index |
| ADD | **0x02** | Pop 2 values, push sum | - |
| SUB | **0x03** | Pop 2 values, push difference | - |
| MUL | **0x04** | Pop 2 values, push product | - |
| DIV | **0x05** | Pop 2 values, push quotient | - |
| PRINT | **0x06** | Pop value and print | - |
| JUMP | **0x07** | Jump to instruction index | address |
| JUMP\_IF | **0x08** | Conditional jump if top is true | address |
| CALL | **0x09** | Call function | function\_index |
| RET | **0x0A** | Return from function | - |

**4. OPERAND & INSTRUCTION ENCODING**

* Opcodes will be represented as **u8** values.
* Operands are **u16 or u32** depending on size.
* An ***instruction decoder*** will break the bytecode into **opcode + operands** for execution.

**EXAMPLE:**

**[0x01][0x00][0x03] => LOAD\_CONST 3**

**[0x02] => ADD**

**5. STACK BEHAVIOR**

**Example for 1 + 2:**

**INSTRUCTIONS**:

LOAD\_CONST 1

LOAD\_CONST 2

ADD

PRINT

**STACK TRACE:**

[]

[1]

[1, 2]

[3]

<prints 3>

**7. FUTURE FEATURES**

* Garbage collection (mark-and-sweep or tracing GC)
* FFI’s to call native libraries
* JIT backend
* WASM target

**8. STATUS & NOTES**

* The *CVM* is still in **pre-alpha** state.
* Currently, it is currently being written in `*src/vm/*` and is not integrated into **Crabby’s** main execution pipeline yet.
* Planned to be activated by `*crabby --vm*` or gets integrated into the **crabc** compiler in the future.

**9. RELATED FILES**

* src/vm/src/bytecode.rs
* src/vm/src/vm.rs
* src/value.rs (holds the *ValueVM enum)*
* docs/MEMORY.md