VM-18 Instruction Set

Format "**INSTRUCTION** $[\mathbf{arg_1}]$ $[\mathbf{arg_2}]$ … $[\mathbf{arg}_n]$"

where $[\text{arg}_i]$ is pushed onto the stack before $[\text{arg}_{i+1}]$ and $[\text{arg}_i]$ represents a single byte.

opcode: 0x01

PUSH_BYTE [byte]

Pushes next byte found in the instruction stream onto the stack

opcode: 0x02

HALT

Stops execution of instruction stream

opcode: 0x03

ADD_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes the result of Number_1 + Number_2 onto stack

opcode: 0x04

SUB_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes the result of Number_1 - Number_2 onto stack opcode: 0x05

MUL_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes the result of $\mathrm{Number}_1 \cdot \mathrm{Number}_2$ onto stack

opcode: 0x06

DIV_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes the result of $\mathrm{Number}_1 \div \mathrm{Number}_2$ onto stack

opcode: 0x07

MOD_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes the result of Number_1 modulo Number_2 onto stack

opcode: 0x08

LOAD_BYTE $[\mathbf{ADDRESS_1}]$ $[\mathbf{ADDRESS_2}]$ … $[\mathbf{ADDRESS}_n]$

Fetches byte stored at address, $\mathrm{ADDRESS}_1$ being MSB, and pushes onto stack. Address size is determined by the VM implementation.

opcode: 0x09

STORE_BYTE [BYTE] $[\mathbf{ADDRESS_1}]$ $[\mathbf{ADDRESS_2}]$ … $[\mathbf{ADDRESS}_n]$

Stores byte at address, $\mathrm{ADDRESS}_1$ being MSB. Address size is determined by the VM implementation.

opcode: 0x0A

SEND_INTERFACE [I] $[\mathbf{A_1}]$ $[\mathbf{A_2}]$ … $[\mathbf{A}_n]$ $[\mathbf{N_1}]$ $[\mathbf{N_2}]$ … $[\mathbf{N}_n]$

Sends N bytes starting at address A to interface number I. A_1 and N_1 are the MSBs, and n is the number of bytes in the implementation’s memory address.

opcode: 0x0B

RECV_INTERFACE [I] $[\mathbf{A_1}]$ $[\mathbf{A_2}]$ … $[\mathbf{A}_n]$ $[\mathbf{N_1}]$ $[\mathbf{N_2}]$ … $[\mathbf{N}_n]$

Receives N bytes to address A from interface number I. A_1 and N_1 are the MSBs, and n is the number of bytes in the implementation's memory address.

opcode: 0x0C

AND_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes result of performing the bitwise AND on $\mathrm{Number_1}$ and $\mathrm{Number_2}$.

opcode: 0x0D

OR_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes result of performing the bitwise OR on $\mathrm{Number_1}$ and $\mathrm{Number_2}$.

opcode: 0x0E

NOT_BYTE $[\mathbf{Number}]$

Pushes result of performing the bitwise NOT on Number.

opcode: 0x0F

XOR_BYTE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$

Pushes result of performing the bitwise XOR on $\mathrm{Number_1}$ and $\mathrm{Number_2}$.

opcode: 0x10

JUMPG $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$ $[\mathbf{I_1}]$ $[\mathbf{I_2}]$ … $[\mathbf{I}_n]$

If $\mathrm{Number_1} > \mathrm{Number_2}$, move instruction pointer to instruction address specified by I_i where n is the implementation's address size.

opcode: 0x11

JUMPE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$ $[\mathbf{I_1}]$ $[\mathbf{I_2}]$ … $[\mathbf{I}_n]$

If Number_1 == Number_2, move instruction pointer to instruction address specified by I_i where n is the implementation's address size.

opcode: 0x12

JUMPL $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$ $[\mathbf{I_1}]$ $[\mathbf{I_2}]$ … $[\mathbf{I}_n]$

If Number_1 < Number_2, move instruction pointer to instruction address specified by I_i where n is the implementation's address size.

opcode: 0x13

JUMPNE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$ $[\mathbf{I_1}]$ $[\mathbf{I_2}]$ … $[\mathbf{I}_n]$

If Number_1 != Number_2, move instruction pointer to instruction address specified by I_i where n is the implementation's address size.

opcode: 0x14

JUMPLE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$ $[\mathbf{I_1}]$ $[\mathbf{I_2}]$ … $[\mathbf{I}_n]$

If Number_1 <= Number_2, move instruction pointer to instruction address specified by I_i where n is the implementation's address size.

opcode: 0x15

JUMPGE $[\mathbf{Number_1}]$ $[\mathbf{Number_2}]$ $[\mathbf{I_1}]$ $[\mathbf{I_2}]$ … $[\mathbf{I}_n]$

If Number_1 >= Number_2, move instruction pointer to instruction address specified by I_i where n is the implementation's address size.

opcode: 0x16

JUMP $[\mathbf{I_1}]$ $[\mathbf{I_2}]$ … $[\mathbf{I}_n]$

Move instruction pointer to instruction address specified by I_i where n is the implementation's address size.