Arcturus Virtual Machine Specification

Contents

1 Bytecode Mnemonic Conversion

1.1 Opcode Reference Table

Mnemonic	Opcode	Category
ADD	0x0001	Arithmetic
SUB	0x0002	Arithmetic
MUL	0x0003	Arithmetic
DIV	0x0004	Arithmetic
MOD	0x0005	Arithmetic
STORE	0x0010	Stack Operations
PUSH	0x0011	Stack Operations
LOAD	0x0012	Stack Operations
COMPARE	0x00C0	Comparison
JUMP_IF	0x00C1	Comparison
CALL_IF	0x00C2	Comparison
EQ	0x00C3	Comparison
GE	0x00C4	Comparison
GT	0x00C5	Comparison
LE	0x00C6	Comparison
LT	0x00C7	Comparison
NE	0x00C8	Comparison
AND	0x00D0	Logical
OR	0x00D1	Logical
NOT	0x00D2	Logical
XOR	0x00D3	Logical
LABEL	0x00E0	Control Flow
JUMP	0x00E1	Control Flow
PRINT	0xE000	I/O
INPUT	0xE001	I/O
CALL	0xE002	Control Flow
RET	0xE003	Control Flow
ARC_START	0xFFF0	Structure
ARC_END	0xFFF1	Structure
ARC_DELIM	0xFFF2	Structure

2 Keywords

2.1 Control Flow

2.1.1 `if` Statement

Basic if statement `if x > 5 jump l1`

```
PUSH <left>
PUSH <right>
```

```
COMPARE <operand>  ; auto pushes true or false
JUMP_IF <label>    ; does not jump if false
```

Complex conditions with AND `if x > 5 && y > 5`

```
; cont1
PUSH <left>
PUSH <right>
COMPARE <operand>

; cont2
PUSH <left>
PUSH <right>
COMPARE <operand>

; finalize
AND        ; takes top two stack items and &gates them, pushes result
JUMP_IF <label>
```

Complex conditions with multiple operators `if x > 5 && y > 5 || z > 5 && a > 5`

```
; cont1
PUSH <left>
PUSH <right>
COMPARE <op>

; cont2
PUSH <left>
PUSH <right>
COMPARE <op>

; cont3
PUSH <left>
PUSH <right>
COMPARE <op>

; cont4
PUSH <left>
PUSH <right>
COMPARE <op>

; finalize
AND
AND
OR  ; all ANDs are done first, then ORs
    ; OR takes the top two values and |gates them
```

Negation `if !(x > 5)`

```
PUSH <left>
PUSH <right>
COMPARE <op>
NOT  ; takes the top value and flips the boolean
```

2.1.2 LABEL

```
LABEL <label>  ; loads a <label, idx> into a jump table
```

2.1.3 CALL/RETURN

```
CALL <label>  ; JUMPs, but also loads current idx
              ; on the call stack before jumping
```

```
RET  ; returns to the top idx of the call stack
```

2.2 I/O Operations

2.2.1 PRINT

```
PRINT  ; prints the top value of the stack
```

2.2.2 INPUT

```
PUSH <prompt>
PRINT
INPUT  ; take input and push it
```

3 Miscellaneous Tokens

3.1 Arithmetic Operations

3.1.1 = (assign)

```
PUSH <value>
STORE <variable>
```

3.1.2 += (addassign)

```
PUSH <number>
; this will be done for however many values are on the other side
; ex: x = 6 + 5, push 6, push 5
; ex: x += 6, same as x = x + 6, so push x, push 6
ADD <x>       ; pop x values deep and add together, auto push
STORE <dest>
```

3.1.3 -= (subassign)

```
PUSH <number>  ; same as addassign
; ex: x = 6 - 5, push 6, push 5
; ex: x -= 6, same as x = x - 6, so push x, push 6
SUB <x>        ; pop x values deep and subtract all
STORE <dest>
```

3.1.4 *= (mulassign)

```
PUSH <number>  ; same as addassign
MUL <x>        ; pop x values deep and multiply all
STORE <dest>
```

3.1.5 /= (divassign)

```
PUSH <number>
DIV <x>
STORE <dest>
```

3.1.6 Non-Assign Operations

All of these are the same as the assign variants, except they don't load onto the stack unless manually assigned via `=`. They can also be used in comparisons.

`+` add

`-` sub

`*` mul

`/` div

3.1.7 Bit Shifting

>> (bsr - bit shift right) `x >>` is the same as `x *= 2`

<< (bsl - bit shift left) `x <<` is the same as `x /= 2`

3.2 Comparison Operations

Most comparison operations follow the same pattern:

```
PUSH <left value>
PUSH <right value>
COMPARE <cmp>  ; auto pushes true or false
```

4 VM Design

4.1 Architecture

The VM uses two stacks and a hashmap (a jump table):

- **Evaluation Stack:** The first stack is an evaluation stack that most values are pushed onto.
- **Call Stack:** The second stack is a call stack exclusively pushed to and popped from by `CALL`, `RET`, and `CALL_IF`.
- **Jump Table:** A hashmap for label-to-address resolution.

4.2 Execution

The VM uses a double-pass design:

4.2.1 First Pass - Label Collection

The VM looks for byte `0x00E0` (LABEL instruction). When found, it saves the byte right-adjacent to it and the current IP to a hashmap table. This continues until it reaches byte `0xFFF1`, the `ARC_END` instruction.

Example:

```
0x1000 0x00E0 0x1200
       ^
       ip path

In hashmap: <1200, 1>
```

4.2.2 Second Pass - Execution

The VM restarts from byte `0xFFF0` (ARC_START) and progresses through instructions, interpreting them as usual. When `CALL`, `JUMP_IF`, or `JUMP` are encountered, the VM refers to its hashmap table.