Virtual Machine Compiler Documentation

Table of Contents

1. Memory Organization

1.1 Memory Segments

The VM uses the following memory segments:

- Local (8224-8735): 512 bytes for local variables
- Argument (8736-8767): 32 bytes for function arguments
- Temp (8768-9279): 512 bytes for temporary values
- Stack (9280-10303): 1024 bytes for operation stack
- Heap (10304-11327): 1024 bytes for dynamic memory
- Pointer Section (8216-8220): For managing heap allocations

1.2 Variable Addressing

- Local variables are zero-indexed
- Each variable occupies 4 bytes (aligned)
- Variables are assigned sequential indices based on declaration order

2. Data Types and Segments

2.1 Supported Data Types

```C/C++
INT    : 4 bytes, signed integer
```

```
FLOAT  : 4 bytes, IEEE-754 single precision
CHAR   : 1 byte (stored in 4 bytes)
BOOL   : 1 byte (stored in 4 bytes)
PTR    : 12 bytes (base_address, size, type)
```

2.2 Type Conversion Rules

- Implicit conversions between INT and FLOAT are handled automatically
- CHAR can be converted to INT
- No other implicit conversions are allowed

3. Variable Management

3.1 Variable Declaration and Assignment

```C/C++
// JOI code
int a = 5;

// VM code
push constant 5 INT
pop local 0 INT
```

3.2 Multiple Variable Declaration

```C/C++
// JOI code
int a;
float b;
char c;

// VM code
// a is local 0, b is local 1, c is local 2
// Initialize with default values if needed
```

```
push constant 0 INT
pop local 0 INT
push constant 0.0 FLOAT
pop local 1 FLOAT
push constant 0 CHAR
pop local 2 CHAR
```

4. Array and Pointer Operations

4.1 Array Declaration

```C/C++
// JOI code
int arr[10];

// VM code
alloc 10 INT       // Allocates memory and pushes (base_addr, size, type)
pop local 0 PTR    // Stores pointer in local variable
```

4.2 Array Access and Assignment

```C/C++
// JOI code
int x = arr[0] + 1;

// VM code
push local 0 PTR   // Push array pointer
push constant 0 INT   // Push index
getindex          // Calculate address
access INT
```

```C/C++
// JOI code
```

```
arr[i] = 96;

// VM code
push local 0 PTR   // Push array pointer
push local 1 INT   // Push index
getindex          // Calculate address
push constant 96 INT
store INT
```

4.3 Dynamic Memory Allocation

```c++
// JOI code
int* ptr = new int[10];

// VM code
alloc 10 INT      // Same as static array
pop local 0 PTR

// Deallocation
// delete[] ptr;
push local 0 PTR
delete
```

5. Function Management

5.1 Function Declaration

```c++
// JOI code
int add(int a, int b) {
    return a + b;
}
```

```
// VM code
function add 2 INT    // name, num_args, return_type
push argument 0 INT
push argument 1 INT
add
return INT
```

5.2 Function Call

```C/C++
// JOI code
int result = add(5, 3);

// VM code
push constant 5 INT
push constant 3 INT
call add 2           // name, num_args
pop local 0 INT      // store result
```

6. Control Flow

6.1 If-Else Statement

```C/C++
// JOI code

if (a > b) {

    x = 1;

} else {
```

```
    x = 2;
}


// VM code
push local 0 INT     // a
push local 1 INT     // b
gt                   // greater than
if-goto IF_TRUE0
goto IF_FALSE0
label IF_TRUE0
push constant 1 INT
pop local 2 INT      // x
goto IF_END0
label IF_FALSE0
push constant 2 INT
pop local 2 INT      // x
label IF_END0
```

6.2 While Loop

C/C++

```
// JOI code

while (i < 10) {

    sum += i;

    i++;

}



// VM code

label WHILE_START0

push local 0 INT     // i

push constant 10 INT

lt

if-goto WHILE_BODY0

goto WHILE_END0

label WHILE_BODY0

// loop body

push local 1 INT     // sum

push local 0 INT     // i

add

pop local 1 INT      // sum
```

```
// increment i

push local 0 INT

push constant 1 INT

add

pop local 0 INT

goto WHILE_START0

label WHILE_END0
```

7. Example Translations

7.1 Complete Example

```
C/C++

// JOI code

#include <math.h>



int add(int a, int b);



int joi() {

    int a = 9;

    int b = 20;
```

```
    int c = 45;

    int m1 = max(a,b);

    int m2 = max(b,c);

    int result = add(m1,m2);

    printf("%d",result);

    return 0;

}


// VM code

lib math.jvm

function add 2 INT


function joi

    push constant 9 INT

    pop local 0 INT      // a

    push constant 20 INT

    pop local 1 INT      // b

    push constant 45 INT

    pop local 2 INT      // c

    push local 0 INT     // a

    push local 1 INT     // b
```

```
call max 2
pop local 3 INT      // m1
push local 1 INT     // b
push local 2 INT     // c
call max 2
pop local 4 INT      // m2
push local 3 INT     // m1
push local 4 INT     // m2
call add 2
pop local 5 INT      // result
push local 5 INT
print INT
push constant 0 INT
return INT
```

7.2 Memory Layout Example

For the above code:

- `a`: local 0
- `b`: local 1
- `c`: local 2
- `m1`: local 3
- `m2`: local 4
- `result`: local 5

Implementation Notes

1. Maintain a symbol table during compilation to track:
    - Variable names and their local indices
    - Function names and their argument counts
    - Current scope and nesting level
    - Type information
2. Memory Management Rules:
    - Align all variables to 4 bytes
    - Track heap allocations in pointer section
    - Implement garbage collection if needed
3. Error Checking:
    - Type compatibility
    - Array bounds
    - Valid function calls
    - Memory leaks
4. Optimization Opportunities:
    - Constant folding
    - Dead code elimination
    - Register allocation
    - Inlining small functions

8. Class Implementation

8.1 Memory Organization for Objects

Objects are stored as a pointer Triplet (12 bytes) and are allocated on the heap with the following structure:

- Attributes section (4 bytes per attribute, aligned)

8.2 Class Declaration

Basic Class Structure

```
C/C++
// JOI code
class MyClass {
    private:
        int x;
        float y;
```

```
    public:
        char z;
}

// VM code
class 0                  // Unique class ID
begin
private
begin
declare local 0 INT      // x
declare local 1 FLOAT    // y
end
public
begin
declare local 2 CHAR     // z
end
end
```

Class with Methods

```
C/C++

// JOI code

class Calculator {

    private:

        int result;

    public:

        int add(int a, int b) {

            result = a + b;
```

```
            return result;
        }
}


// VM code
class 1
begin
private
begin
declare local 0 INT      // result
end
public
begin
method add 2 INT         // name, num_args, return_type
    push argument 0 INT  // a
    push argument 1 INT  // b
    add
    dup                 // duplicate result for return
    push this PTR       // load this pointer
```

```
    swap                // swap result and this ptr
    setattribute 0 0 INT // store in result field
    return INT
end
end
```

8.3 Object Operations

Object Creation

```
C/C++
// JOI code
MyClass obj = new MyClass();

// VM code
createobject 0 0        // class_id, constructor_args
pop local 0 PTR              // Store object reference
```

Attribute Access

```
C/C++
// Reading attributes
// obj.x
push local 0 PTR        // Push object reference
getattribute 0 0 INT    // class_id, attribute_index, type

// Writing attributes
// obj.x = 42
```

```
push local 0 PTR        // Push object reference
push constant 42 INT
setattribute 0 0 INT    // class_id, attribute_index, type
```

Method Calls

```C/C++
// obj.add(5, 3)
push local 0 PTR        // Push object reference
push constant 5 INT     // Push first argument
push constant 3 INT     // Push second argument
mcall add 2 INT         // method_name, num_args, return_type
```