ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT (CO2003)

Đặc tả

JAVM - Just Another Virtual Machine

Phiên bản 1.1.0

1 Giới thiệu chung

Máy ảo (Virtual machine) là một chương trình phần mềm có thể chạy trên một máy vật lý để hoạt động như một hệ thống máy tính ảo cùng với những thành phần khác, trong đó có một bộ lệnh riêng.

Máy ảo có rất nhiều ứng dụng khác nhau:

- Trình mô phỏng/giả lập hệ thống máy tính vật lý trong thực tế để có thể thực thi các phần mềm trên đó. (VirtualBox, VMWare, v.v.)
- Nền tảng để thực thi cùng một chương trình theo cùng một cách trên các hệ thống vật lý khác nhau. (Java virtual machine, v.v.)
- Các ứng dụng sử dụng nhiều loại dữ liệu phức tạp. (Microsoft Word với văn bản, hình ảnh, bảng, v.v.)

Các phần tiếp theo sẽ đặc tả các thành phần của máy ảo JAVM (Just Another Virtual Machine) - đối tượng cần quan tâm của các bài tập lớn trong khóa học này.

2 Máy ảo JAVM

Máy ảo JAVM là một máy dựa trên ngăn xếp (stack-based machine - tương tự Java Virtual Machine) hoạt động theo cơ chế đơn luồng (single-threaded). Thành phần chính của JAVM là một ngăn xếp gọi là JA Stack. Mỗi khi một hàm được thực thi, một khung ngăn xếp (stack frame) được tạo ra trên JA Stack và toàn bộ hàm sẽ được thực thi tại đây.

Các thành phần cần quan tâm của một khung ngăn xếp bao gồm:

- Ngăn xếp toán hạng (operand stack): một ngăn xếp sử dụng để đẩy/lấy (push/pop) các giá trị toán hạng, kết quả của các phép toán, các tham số để gọi hàm, giá trị trả về của một hàm, v.v.

- Không gian biến cục bộ (local variable space): một không gian chứa toàn bộ các biến cục bộ của hàm đang thực thi.

JAVM có một tập lệnh riêng để thực thi chương trình.

2.1 Kiểu dữ liệu (Data types)

Máy ảo JAVM hỗ trợ các kiểu dữ liệu trong bảng sau:

Kiểu	Miền giá trị	Kích thước	Mã
`boolean`	$0, 1$	1 byte	
`char`	$-2^7 \dots 2^7 - 1$	1 byte	
`short`	$-2^{15} \dots 2^{15} - 1$	2 byte	
`int`	$-2^{31} \dots 2^{31} - 1$	4 byte	0
`float`	32-bit IEEE 754 single-precision float	4 byte	1

2.2 Ngăn xếp toán hạng (Operand stack)

Ngăn xếp toán hạng dùng để thực thi các phép toán, lưu các tham số để gọi hàm, giá trị trả về của một hàm, v.v. Ngăn xếp toán hạng có cơ chế lưu trữ giống một ngăn xếp (stack) thông thường, chỉ có phép dữ liệu được đẩy/lấy từ phần tử trên cùng của ngăn xếp. Ngoài giá trị dữ liệu, ngăn xếp còn lưu thêm kiểu dữ liệu của mỗi phần tử (mã của kiểu dữ liệu) dưới dạng một giá trị kiểu `int`. Kích thước tối đa của ngăn xếp do máy ảo quy định.

Với JAVM, ngăn xếp toán hạng có đơn vị lưu trữ là word (4 byte), do đó các giá trị kiểu `boolean`, `char` và `short` khi được đẩy vào ngăn xếp sẽ được ép kiểu về `int`.

Cách thức thực thi một phép toán trên ngăn xếp toán hạng được mô tả trong ví dụ sau.

Xét phép toán: $1 + 2 * 3 - 4.0$ được mô tả bằng các dòng lệnh sau:

Lệnh	Mô tả	Trạng thái ngăn xếp
	Trạng thái ban đầu	`<>`
`iconst 1`	Đẩy giá trị 1 và 0 (mã kiểu `int`) vào ngăn xếp.	`<1, 0>`
`iconst 2`	Đẩy giá trị 2 và 0 (mã kiểu `int`) vào ngăn xếp.	`<1, 0, 2, 0>`
`iconst 3`	Đẩy giá trị 3 và 0 (mã kiểu `int`) vào ngăn xếp.	`<1, 0, 2, 0, 3, 0>`
`imul`	Lấy hai phần tử ở đỉnh ngăn xếp Kiểm tra kiểu của hai phần tử (mã đều là 0, hợp lệ) Thực hiện phép nhân hai phần tử, đẩy kết quả là 6 cùng mã kiểu là 0 vào ngăn xếp	`<1, 0, 6, 0>`
`iadd`	Lấy hai phần tử ở đỉnh ngăn xếp Kiểm tra kiểu của hai phần tử (mã đều là 0, hợp lệ) Thực hiện phép cộng hai phần tử, đẩy kết quả là 7 cùng mã kiểu là 0 vào ngăn xếp	`<7, 0>`
`fconst 4.0`	Đẩy giá trị 4.0 và 1 (mã kiểu `float`) vào ngăn xếp	`<7, 0, 4.0, 1>`
`fsub`	Lấy hai phần tử ở đỉnh ngăn xếp Kiểm tra kiểu của hai phần tử (`int` và `float`, hợp lệ) Thực hiện phép trừ hai phần tử, đẩy kết quả là 3.0 cùng mã kiểu là 1 vào ngăn xếp	`<3.0, 1>`

Như vậy, đỉnh của ngăn xếp lúc này là kết quả của phép toán cùng kiểu dữ liệu của nó.

2.3 Không gian biến cục bộ (Local variable space)

Không gian biến cục bộ của mỗi khung ngăn xếp trong JAVM được xây dựng bằng một cây AVL với key tại mỗi node là tên biến, value tương ứng là giá trị hiện tại cùng kiểu dữ liệu của biến đó. Kích thước tối đa của cây AVL do máy ảo quy định.

Vị trí chèn vào của một node phụ thuộc vào phép so sánh giữa hai key (trong trường hợp này là hai chuỗi kí tự). Chuỗi a được xem là bé hơn chuỗi b nếu như kí tự tại vị trí đầu tiên không giống nhau của a có giá trị tương đương bảng mã ASCII bé hơn b hoặc tất cả kí tự của a đều giống các kí tự của b tại cùng vị trí nhưng a có độ dài ngắn hơn. Vì tính chất của không gian biến cục bộ, cùng một lúc không thể tồn tại hai node có cùng key.

Đơn vị lưu trữ của cây AVL này là word (4 byte), do đó các giá trị kiểu `boolean`, `char` và `short` khi được lưu vào cây sẽ được ép kiểu về `int`. Mỗi phần tử bao gồm mã kiểu dữ liệu (kiểu `int`) tương ứng và giá trị biến đang lưu trữ.

Xét các thao tác: `int a = 1; int b = a;` được mô tả bằng các dòng lệnh sau:

Lệnh	Mô tả	Mảng biến cục bộ	Ngăn xếp toán hạng
	Trạng thái ban đầu	`{}`	`<>`
`iconst 1`	Đẩy giá trị 1 và 0 (mã kiểu `int`) vào ngăn xếp.	`{}`	`<1, 0>`
`istore a`	Lấy phần tử đầu ra khỏi ngăn xếp Kiểm tra kiểu (mã kiểu là 0, hợp lệ) Lưu mã kiểu là 0 cùng giá trị 1 vào cây AVL với key là `a`.	`{a: [0, 1]}`	`<>`
`iload a`	Kiểm tra kiểu của biến `a`, lấy giá trị và kiểu của biến `a` đẩy vào ngăn xếp.	`{a: [0, 1]}`	`1, 0`
`istore b`	Lấy phần tử đầu ra khỏi ngăn xếp Kiểm tra kiểu (mã kiểu là 0, hợp lệ) Lưu mã kiểu là 0 cùng giá trị 1 vào cây AVL với key là `b`.	`{a: [0, 1], b: [0, 1]}`	`<>`

Như vậy, trạng thái của các biến cục bộ tại một thời điểm được lưu trữ trong không gian biến cục bộ.

2.4 Tập lệnh (Instructions)

Tập lệnh của máy ảo JAVM được chia thành nhiều nhóm lệnh, ta cần quan tâm một số nhóm lệnh chính được mô tả ở bảng sau:

No	Cú pháp	Mô tả
Nhóm lệnh phép toán (Arithmetic Instructions)		
1	`iadd`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép toán cộng và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
2	`fadd`	Tương tự `iadd` với kiểu `float`.
3	`isub`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép toán trừ (phần tử trên cùng là toán hạng thứ hai) và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
4	`fsub`	Tương tự `isub` với kiểu `float`.
5	`imul`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép toán nhân và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
6	`fmul`	Tương tự `imul` với kiểu `float`.
7	`idiv`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép toán chia (phần tử trên cùng là toán hạng thứ hai) và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`. Nếu toán hạng thứ hai là 0, cần throw lỗi `DivideByZero`.
8	`fdiv`	Tương tự `idiv` với kiểu `float`.
9	`irem`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép toán chia lấy phần dư (phần tử trên cùng là toán hạng thứ hai) và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`. Nếu toán hạng thứ hai là 0, cần throw lỗi `DivideByZero`. Phép chia lấy phần dư `a rem b` được định nghĩa bằng `a - (a div b) * b`, với `div` là phép chia lấy phần nguyên hai số nguyên.
10	`ineg`	Lấy ra phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép đảo dấu và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
11	`fneg`	Tương tự `ineg` với kiểu `float`.
12	`iand`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép bitwise and và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
13	`ior`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép bitwise or và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
14	`ieq`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép so sánh bằng (phần tử trên cùng là toán hạng thứ hai) và đẩy kết quả (0 nếu false, 1 nếu true, kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
15	`feq`	Tương tự `ieq` với kiểu `float`. Kết quả đẩy vào ngăn xếp là kiểu `int`.
16	`ineq`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép so sánh không bằng (phần tử trên cùng là toán hạng thứ hai) và đẩy kết quả (0 nếu false, 1 nếu true, kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.

No	Cú pháp	Mô tả
17	`fneq`	Tương tự `ineq` với kiểu `float`. Kết quả đẩy vào ngăn xếp là kiểu `int`.
18	`ilt`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép so sánh bé hơn (phần tử trên cùng là toán hạng thứ hai) và đẩy kết quả (0 nếu false, 1 nếu true, kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
19	`flt`	Tương tự `ilt` với kiểu `float`. Kết quả đẩy vào ngăn xếp là kiểu `int`.
20	`igt`	Lấy ra hai phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`), thực hiện phép so sánh lớn hơn (phần tử trên cùng là toán hạng thứ hai) và đẩy kết quả (0 nếu false, 1 nếu true, kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
21	`fgt`	Tương tự `igt` với kiểu `float`. Kết quả đẩy vào ngăn xếp là kiểu `int`.
22	`ibnot`	Lấy ra phần tử đầu ở ngăn xếp, kiểm tra kiểu hợp lệ (`int`) và đẩy kết quả (0 nếu giá trị khác 0, 1 nếu giá trị bằng 0, kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
Nhóm lệnh nạp và lưu (Load and Store Instructions)		
23	`iconst <val>`	Đẩy giá trị `<val>` (kiểu `int`) vào ngăn xếp. `<val>` là một hằng số kiểu nguyên, cấu tạo từ các chữ số 0-9 và dấu âm (-) ở đầu nếu là số âm.
24	`fconst <val>`	Tương tự `iconst` với kiểu `float`. `<val>` là một hằng số kiểu chấm động, cấu tạo từ các chữ số 0-9, một dấu chấm (.) giữa các chữ số và dấu âm (-) ở đầu nếu là số âm.
25	`iload <var>`	Sao chép giá trị biến `var` trong không gian biến cục bộ, kiểm tra kiểu hợp lệ (kiểu `int`) và đẩy giá trị vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`. `<var>` là một chuỗi ký tự tạo thành từ các kí tự `a-zA-Z`.
26	`fload <var>`	Tương tự `iload` với kiểu `float`.
27	`istore <var>`	Lấy ra phần tử đầu tiên từ ngăn xếp, kiểm tra kiểu hợp lệ (kiểu `int`) và lưu vào trong không gian biến cục bộ (kiểu `int`) với key là `var`. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`. `<var>` là một chuỗi ký tự tạo thành từ các kí tự `a-zA-Z`.
28	`fstore <var>`	Tương tự `istore` với kiểu `float`.
Nhóm lệnh chuyển đổi kiểu (Type conversion Instructions)		
29	`i2f`	Lấy ra phần tử đầu tiên từ ngăn xếp, kiểm tra kiểu hợp lệ (kiểu `int`), thực hiện chuyển đổi sang kiểu `float` và đẩy kết quả (kiểu `float`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
30	`f2i`	Lấy ra phần tử đầu tiên từ ngăn xếp, kiểm tra kiểu hợp lệ (kiểu `float`), thực hiện chuyển đổi sang kiểu `int` (lấy phần nguyên) và đẩy kết quả (kiểu `int`) vào ngăn xếp. Nếu kiểu không hợp lệ, cần throw lỗi `TypeMismatch`.
Nhóm lệnh quản lý trạng thái ngăn xếp toán hạng (Operand Stack Management Instructions)		
31	`top`	In ra console giá trị của phần tử ở đầu ngăn xếp (không phải mã kiểu) cùng một kí tự `\n`. Lệnh này không làm thay đổi trạng thái của khung ngăn xếp.
Nhóm lệnh quản lý biến cục bộ (Local Variable Management Instructions)		

No	Cú pháp	Mô tả
32	`val <var>`	In ra console giá trị của biến `var` trong không gian biến cục bộ (không phải mã kiểu) cùng một kí tự `\n`. Lệnh này không làm thay đổi trạng thái của khung ngăn xếp. `<var>` là một chuỗi ký tự tạo thành từ các kí tự `a-zA-Z`.
33	`par <var>`	In ra console tên của biến là node cha của biến `var` trong cây AVL không gian biến cục bộ (hoặc in ra chuỗi `"null"` nếu `a` là node root) cùng một kí tự `\n`. Lệnh này không làm thay đổi trạng thái của khung ngăn xếp. `<var>` là một chuỗi ký tự tạo thành từ các kí tự `a-zA-Z`.

Lưu ý: Đối với những lệnh yêu cầu toán hạng kiểu `float` trong các lệnh 1-22, nếu toán hạng lấy ra là kiểu `int` thì sẽ được ép kiểu thành `float` trước khi thực hiện phép toán. Trường hợp này không xuất hiện lỗi.

2.5 Các lỗi thực thi (Exceptions)

Các lỗi thực thi cần được xem xét được mô tả ở bảng sau:

No	Lỗi	Mô tả
1	`TypeMismatch(line)`	Xảy ra khi các toán hạng/phần tử lấy từ ngăn xếp toán hạng hay không gian biến cục bộ không phù hợp kiểu với câu lệnh đang thực hiện.
2	`DivideByZero(line)`	Xảy ra khi toán hạng thứ hai trong phép chia là 0 hoặc 0.0.
3	`StackFull(line)`	Xảy ra khi ngăn xếp toán hạng đã đầy, không thể đẩy thêm vào được nữa.
4	`StackEmpty(line)`	Xảy ra khi phải lấy một phần tử ra khỏi ngăn xếp đang trống.
5	`LocalSpaceFull(line)`	Xảy ra khi lưu (store) một biến mới vào trong không gian biến cục bộ đã đầy.
6	`UndefinedVariable(line)`	Xảy ra khi nạp (load) dữ liệu từ vị trí trong mảng không gian bộ nhớ cục bộ chưa được lưu (store) trước đó.

Khi một câu lệnh có thể phát sinh nhiều lỗi, cần xem xét thứ tự thực hiện trong mô tả của câu lệnh để xác định thứ tự lỗi xảy ra.

Khi trả về các lỗi, cần kèm theo giá trị `line` là số thứ tự dòng lệnh phát sinh lỗi. Dòng lệnh bắt đầu của chương trình có số thứ tự là 1. Sau khi gặp lỗi, chương trình dừng và không thực thi các lệnh sau đó.

Giả định ngoài các lỗi thực thi trên, không có lỗi nào khác xảy ra.

3 Changelog

Phiên bản 1.0.1:

- Cập nhật mô tả cho `irem`, `iload`, `istore`, `i2f`, `f2i`.
- Cập nhật mô tả về thứ tự phát sinh lỗi.

Phiên bản 1.1.0:

- Cập nhật mô tả cho `f2i`.
- Thay đổi các mô tả chuyển không gian biến cục bộ từ mảng thành cây AVL.