Code Generation

Dr. Phung Nguyen

Faculty of Computer Science and Engineering
University of Technology
HCMC Vietnam National University

November 25, 2020

Outline

1 Translation to a stack-based machine

Chương này nói về cách tạo ra code cho máy tính. Quá trình này gồm mấy bước:

1. **Tạo code độc lập với máy:** Tạo code ở dạng "chung chung", không quan tâm đến máy tính cụ thể nào.
2. **Tạo code trung gian:** "Dịch" code ở bước 1 sang một dạng trung gian, vừa phụ thuộc vào ngôn ngữ lập trình, vừa hơi hơi liên quan đến máy tính.
3. **Tạo code phụ thuộc vào máy:** "Dịch" code trung gian thành code cụ thể cho từng loại máy tính khác nhau.

Để làm được điều đó, mình cần dùng mấy cái công cụ như:

- **Frame:** Để quản lý thông tin (nhãn, biến, stack, v.v.).
- **Các API:** Để tạo ra các chỉ thị, các phép toán, các thao tác đọc/ghi biến, v.v.

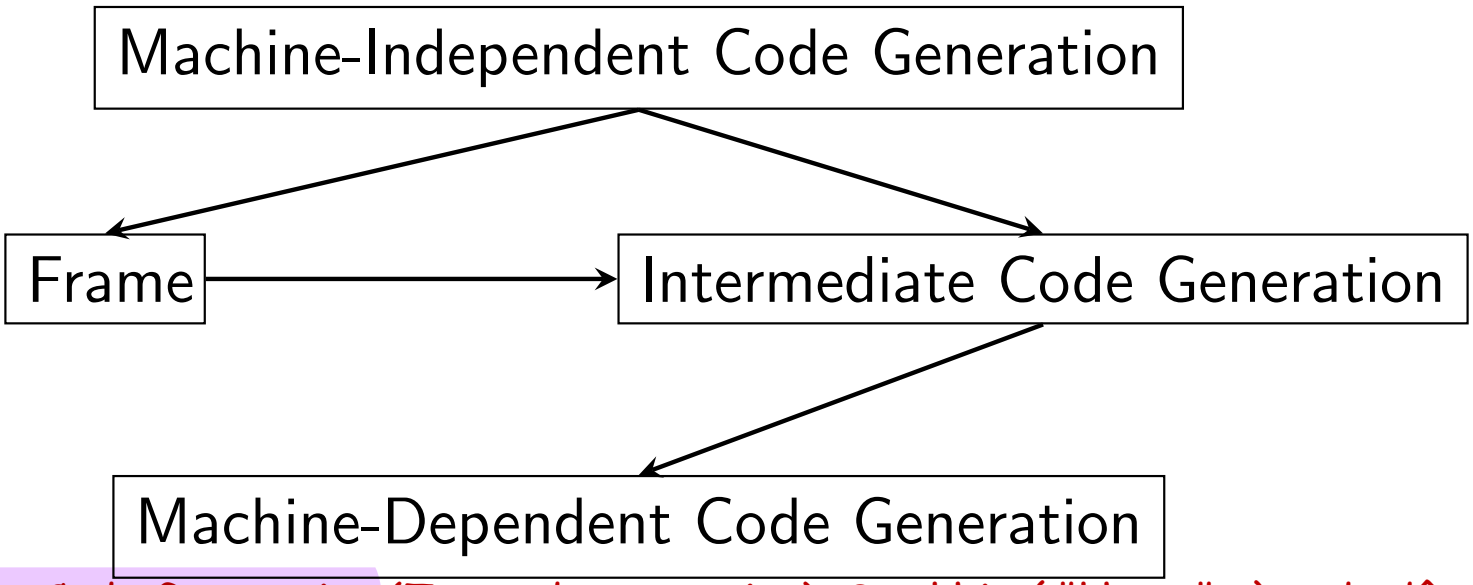
Quan trọng là phải hiểu rõ các kiểu dữ liệu và cách ánh xạ từ ngôn ngữ nguồn sang ngôn ngữ đích.

Tóm lại là, chương này dạy cách "biến" cái code mình viết thành cái mà máy tính hiểu được.

Code Generation Design

Machine-Independent Code Generation (Tạo code độc lập với máy): Đây là bước đầu tiên. Code được tạo ra ở giai đoạn này không phụ thuộc vào kiến trúc cụ thể của máy tính nào cả.

Frame (Khung): Cái này giống như là một bộ công cụ hoặc là một cái "khung" để quản lý thông tin cầ thiết cho việc tạo code, ví dụ như các label (nhãn), biến cục bộ, stack (ngăn xếp) các thứ.



Intermediate Code Generation (Tạo code trung gian): Sau khi có "khung" và code độc lập với máy, mình sẽ chuyển nó thành một dạng code trung gian. Cái code này nó vừa phụ thuộc vào ngôn ngữ lập trình mình dùng, vừa hơi hơi liên quan đến cái máy mình sẽ chạy code đó.

Machine-Dependent Code Generation (Tạo code phụ thuộc vào máy): Cuối cùng, từ cái code trung gian này, mình sẽ tạo ra code cụ thể cho từng loại máy tính khác nhau. Ví dụ, nếu mình muốn chạy trên máy dùng chip Intel thì sẽ có một loại code riêng, còn nếu chạy trên máy dùng chip ARM thì lại có code khác.

Machine-Dependent Code Generation

- Generating specified machine code
  E.g.: emitLDC(20) $\rightarrow$ "ldc 20"
- Implemented in JasminCode

Intermediate Code Generation

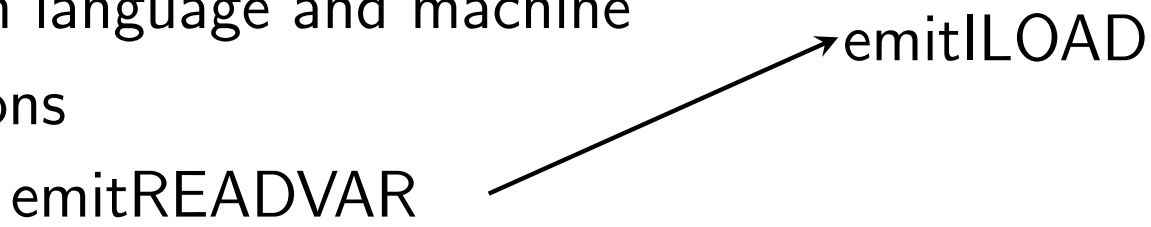
- Depend on both language and machine

Intermediate Code Generation

- Depend on both language and machine
- Select instructions

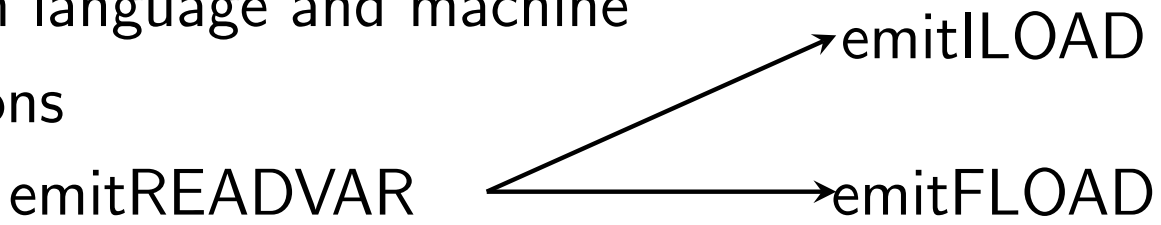
Intermediate Code Generation

- Depend on both language and machine
- Select instructions



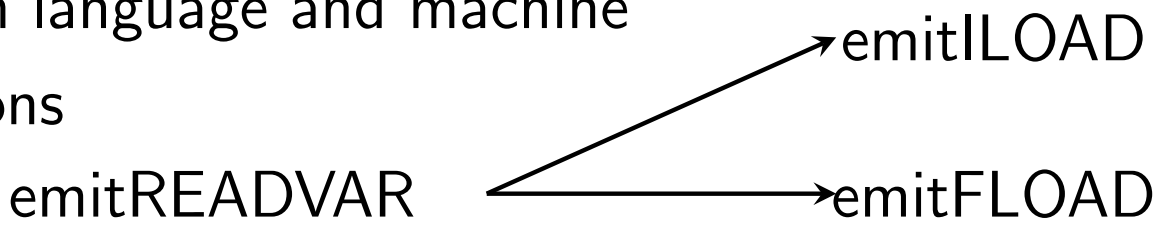
Intermediate Code Generation

- Depend on both language and machine
- Select instructions



Intermediate Code Generation

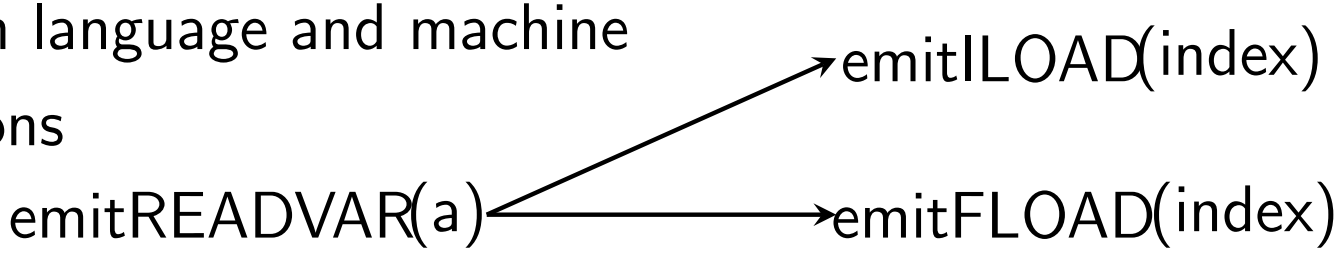
- Depend on both language and machine
- Select instructions



- Select data objects

Intermediate Code Generation

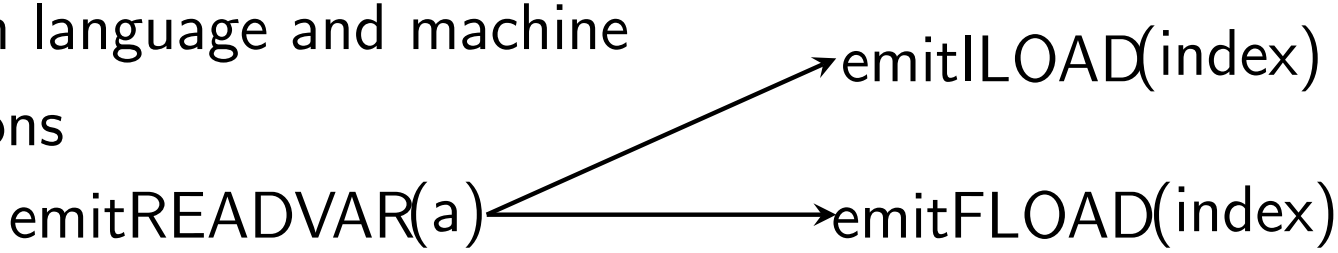
- Depend on both language and machine
- Select instructions



- Select data objects

Intermediate Code Generation

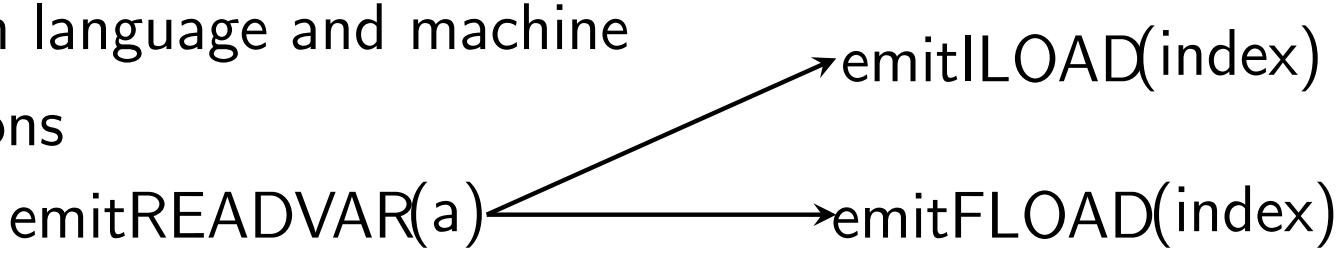
- Depend on both language and machine
- Select instructions



- Select data objects

Intermediate Code Generation

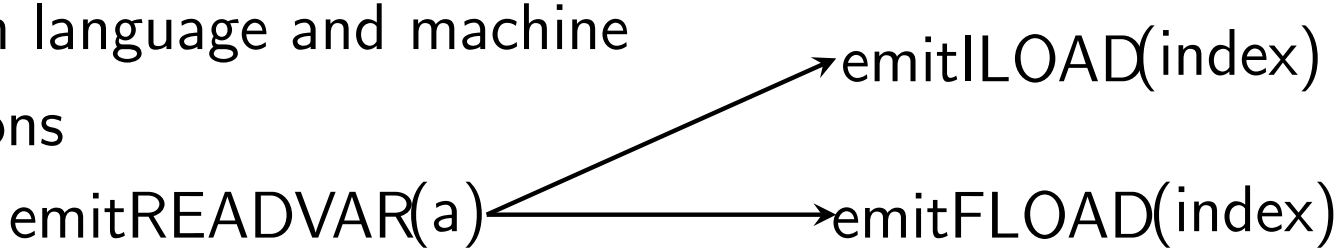
- Depend on both language and machine
- Select instructions



- Select data objects
- Simulate the execution of the machine

Intermediate Code Generation

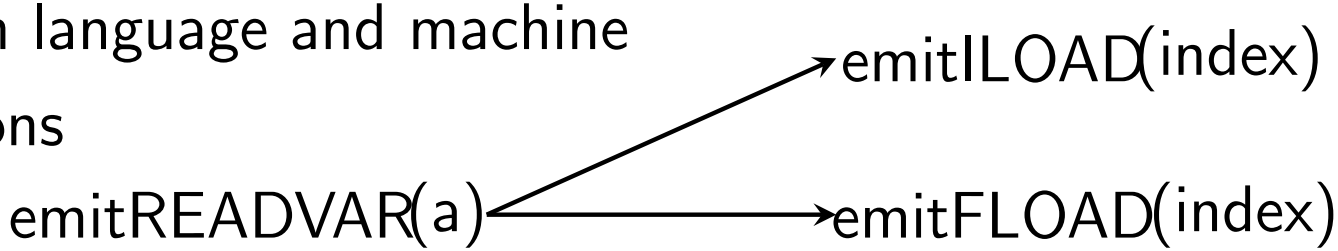
- Depend on both language and machine
- Select instructions



- Select data objects
- Simulate the execution of the machine
  - ◁ emitICONST $\rightarrow$ push()

Intermediate Code Generation

- Depend on both language and machine
- Select instructions

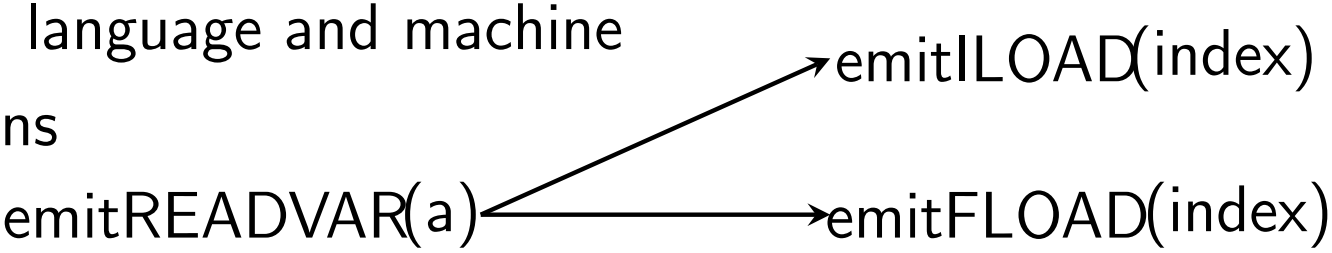


- Select data objects
- Simulate the execution of the machine
  - ◁ emitICONST $\rightarrow$ push()
  - ◁ emitISTORE $\rightarrow$ pop()

Intermediate Code Generation

Depend on both language and machine (Phụ thuộc vào cả ngôn ngữ và máy): Đúng như tên gọi, code trung gian nó là cái cầu nối giữa ngôn ngữ lập trình mình viết và cái máy tính sẽ chạy chương trình.

- Depend on both language and machine
- Select instructions



Select data objects (Chọn đối tượng dữ liệu): Mình cũng cần phải xác định và chọn các đối tượng dữ liệu mà chương trình sẽ sử dụng.

- Select data objects
- Simulate the execution of the machine
  - ◁ emitICONST $\rightarrow$ push()
  - ◁ emitISTORE $\rightarrow$ pop()
- Implemented in class Emitter

Implemented in class Emitter (Được thực hiện trong lớp Emitter): Cái này chỉ là thông tin thêm về mặt kỹ thuật, cho thấy rằng những cái thao tác này được hiện thực hóa trong một cái lớp có tên là Emitter.

Simulate the execution of the machine (Mô phỏng việc thực thi của máy): Để đảm bảo mọi thứ hoạt động đúng, mình cần mô phỏng lại cách mà máy tính sẽ thực thi các lệnh. Ví dụ, lệnh emitICONST (đẩy một giá trị integer lên stack) sẽ tương ứng với thao tác push() trên stack, còn emitISTORE (lấy một giá trị integer từ stack và lưu vào biến) sẽ tương ứng với thao tác pop().

Select instructions (Chọn lệnh): Trong giai đoạn này, mình sẽ chọn ra những cái lệnh cần thiết để thực hiện các thao tác trong chương trình. Ví dụ , nếu mình có một phép cộng, mình sẽ chọn cái lệnh tương ứng để thực hiện phép cộng đó. Ở đây có ví dụ là emitREADVAR(a) (đọc biến a) có thể dẫn đến việc chọn lệnh emitILOAD(index) (load giá trị integer) hoặc emitFLOAD(index) (load giá trị float) tùy thuộc vào kiểu dữ liệu của biến a.

Directives Generation APIs

- emitVAR(self,index, varName, inType, fromLabel, toLabel)
  .var 0 is this Lio; from Label0 to Label1

Directives Generation APIs

- emitVAR(self,index, varName, inType, fromLabel, toLabel)
  .var 0 is this Lio; from Label0 to Label1
- emitATTRIBUTE(self, lexeme, inType, isFinal, value = None)
  .field public static writer Ljava/io/Writer;

Directives Generation APIs

- emitVAR(self,index, varName, inType, fromLabel, toLabel)
  .var 0 is this Lio; from Label0 to Label1
- emitATTRIBUTE(self, lexeme, inType, isFinal, value = None)
  .field public static writer Ljava/io/Writer;
- emitMETHOD(self, lexeme, inType, isStatic)
  .method public foo(I)I

Directives Generation APIs

- emitVAR(self,index, varName, inType, fromLabel, toLabel)
  .var 0 is this Lio; from Label0 to Label1
- emitATTRIBUTE(self, lexeme, inType, isFinal, value = None)
  .field public static writer Ljava/io/Writer;
- emitMETHOD(self, lexeme, inType, isStatic)
  .method public foo(I)I
- emitENDMETHOD(self, frame)
  .limit stack 1
  .limit locals 1
  .end method

Directives Generation APIs

- emitVAR(self,index, varName, inType, fromLabel, toLabel)
  .var 0 is this Lio; from Label0 to Label1
- emitATTRIBUTE(self, lexeme, inType, isFinal, value = None)
  .field public static writer Ljava/io/Writer;
- emitMETHOD(self, lexeme, inType, isStatic)
  .method public foo(I)I
- emitENDMETHOD(self, frame)
  .limit stack 1
  .limit locals 1
  .end method
- emitPROLOG(self, name, parent)
  .source io.java
  .class public io
  .super java/lang/Object

Directives Generation APIs

- emitVAR(self,index, varName, inType, fromLabel, toLabel)
  .var 0 is this Lio; from Label0 to Label1
- emitATTRIBUTE(self, lexeme, inType, isFinal, value = None)
  .field public static writer Ljava/io/Writer;
- emitMETHOD(self, lexeme, inType, isStatic)
  .method public foo(I)I
- emitENDMETHOD(self, frame)
  .limit stack 1
  .limit locals 1
  .end method
- emitPROLOG(self, name, parent)
  .source io.java
  .class public io
  .super java/lang/Object
- emitEPILOG(self)

Đây là những cái hàm hoặc là những cái công cụ mà mình dùng để tạo ra các chỉ thị trong code trung gian. Mỗi cái API này nó sẽ giúp mình tạo ra một loại chỉ thị khác nhau. Để tao giải thích từng cái cho mày dễ hình dung nè:

emitVAR(self, index, varName, inType, fromLabel, toLabel): Cái này dùng để khai báo một biến.
index: chỉ số của biến.
varName: tên của biến.
inType: kiểu dữ liệu của biến.
fromLabel, toLabel: các nhãn (label) chỉ ra phạm vi mà biến này có hiệu lực.
Ví dụ: .var 0 is this Lio; from Label0 to Label1 khai báo biến ở vị trí số 0, tên là this, kiểu Lio và có hiệu lực từ nhãn Label0 đến Label1.

emitATTRIBUTE(self, lexeme, inType, isFinal, value = None): Cái này dùng để khai báo một thuộc tính (attribute) của một class hoặc một đối tượng.
lexeme: tên của thuộc tính.
inType: kiểu dữ liệu của thuộc tính.
isFinal: cho biết thuộc tính này có phải là final (không thay đổi được) hay không.
value: giá trị khởi tạo của thuộc tính (nếu có).
Ví dụ: .field public static writer Ljava/io/Writer; khai báo một thuộc tính public static tên là writer, kiểu Ljava/io/Writer.

emitMETHOD(self, lexeme, inType, isStatic): Cái này dùng để khai báo một phương thức (method).
lexeme: tên của phương thức.
inType: kiểu dữ liệu trả về của phương thức.
isStatic: cho biết phương thức này có phải là static hay không.
Ví dụ: .method public foo(I)I khai báo một phương thức public tên là foo, nhận vào một tham số kiểu I (integer) và trả về kiểu I.

emitENDMETHOD(self, frame): Cái này dùng để kết thúc việc khai báo một phương thức.
frame: một đối tượng Frame (mày nhớ cái "Frame" ở mấy hình trước không?) dùng để quản lý thông tin của phương thức.
Ví dụ:
.limit stack 1
.limit locals 1
.end method
cái này nó kết thúc việc khai báo phương thức, đồng thời chỉ ra giới hạn của stack và số lượng biến local.

emitPROLOG(self, name, parent): Cái này dùng để tạo phần mở đầu (prolog) của một class.
name: tên của class.
parent: tên của class cha (nếu có).
Ví dụ:
.source io.java
.class public io
.super java/lang/Object
cái này nó khai báo class io, có source file là io.java và kế thừa từ java/lang/Object.

emitEPILOG(self): Cái này dùng để tạo phần kết thúc (epilog) của một class.
Nói chung, đây là những cái "mảnh ghép" để xây dựng nên cái code trung gian. Mỗi API nó đảm nhận một nhiệm vụ cụ thể, giúp mình tạo ra code trung gian một cách có cấu trúc và dễ quản lý.

Type

- class IntType(Type)
- class FloatType(Type)
- class StringType(Type)
- class VoidType(Type)
- class BoolType(Type)
- class ClassType(Type): ♯ cname:str
- class ArrayType(Type): ♯ eleType:Type,dimen:List[int]
- class MType(Type): ♯ partype:List[Type],rettype:Type

class MType(Type): partype:List[Type], rettype:Type: Cái này định nghĩa kiểu phương thức hoặc hàm.
partype:List[Type]: Danh sách các kiểu dữ liệu của các tham số (parameter types) của phương thức/hàm.
rettype:Type: Kiểu dữ liệu trả về (return type) của phương thức/hàm.

class ClassType(Type): cname:str: Cái này định nghĩa kiểu class.
cname:str: Tên của class đó.

class ArrayType(Type): eleType:Type, dimen:List[int]: Cái này định nghĩa kiểu mảng (array).
eleType:Type: Kiểu của các phần tử trong mảng.
dimen:List[int]: Kích thước của các chiều của mảng. Ví dụ, [2, 3] là mảng 2 chiều, chiều thứ nhất có 2 phần tử, chiều thứ hai có 3 phần tử.

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub
- emitMULOP(self, lexeme, inType, frame) $\Rightarrow$ imul, fmul, idiv, fdiv

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub
- emitMULOP(self, lexeme, inType, frame) $\Rightarrow$ imul, fmul, idiv, fdiv
- emitDIV(self, frame) $\Rightarrow$ idiv

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub
- emitMULOP(self, lexeme, inType, frame) $\Rightarrow$ imul, fmul, idiv, fdiv
- emitDIV(self, frame) $\Rightarrow$ idiv
- emitMOD(self, frame) $\Rightarrow$ irem

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub
- emitMULOP(self, lexeme, inType, frame) $\Rightarrow$ imul, fmul, idiv, fdiv
- emitDIV(self, frame) $\Rightarrow$ idiv
- emitMOD(self, frame) $\Rightarrow$ irem
- emitANDOP(self, frame) $\Rightarrow$ iand

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub
- emitMULOP(self, lexeme, inType, frame) $\Rightarrow$ imul, fmul, idiv, fdiv
- emitDIV(self, frame) $\Rightarrow$ idiv
- emitMOD(self, frame) $\Rightarrow$ irem
- emitANDOP(self, frame) $\Rightarrow$ iand
- emitOROP(self, frame) $\Rightarrow$ ior

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub
- emitMULOP(self, lexeme, inType, frame) $\Rightarrow$ imul, fmul, idiv, fdiv
- emitDIV(self, frame) $\Rightarrow$ idiv
- emitMOD(self, frame) $\Rightarrow$ irem
- emitANDOP(self, frame) $\Rightarrow$ iand
- emitOROP(self, frame) $\Rightarrow$ ior
- emitREOP(self, op, inType, frame) $\Rightarrow$ code for $>$, $<$ $>=$, $<=$, $!=$, $==$

Operation Generation APIs

- emitADDOP(self, lexeme, inType, frame) $\Rightarrow$ iadd, fadd, isub, fsub
- emitMULOP(self, lexeme, inType, frame) $\Rightarrow$ imul, fmul, idiv, fdiv
- emitDIV(self, frame) $\Rightarrow$ idiv
- emitMOD(self, frame) $\Rightarrow$ irem
- emitANDOP(self, frame) $\Rightarrow$ iand
- emitOROP(self, frame) $\Rightarrow$ ior
- emitREOP(self, op, inType, frame) $\Rightarrow$ code for $>$, $<$ $>=$, $<=$, $!=$, $==$
- emitRELOP(self, op, inType, trueLabel, falseLabel, frame) $\Rightarrow$ code for condition in if statement

mấy cái hàm này giúp mình tạo ra code trung gian để thực hiện các phép tính toán và các phép so sánh.
Để tao giải thích từng cái cho mày:

emitADDOP(self, lexeme, inType, frame): Cái này dùng để tạo code cho các phép cộng và trừ.
lexeme: Cái ký tự đại diện cho phép toán (+ hoặc -).
inType: Kiểu dữ liệu của các toán hạng (có thể là số nguyên hoặc số thực).
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như iadd (cộng số nguyên), fadd (cộng số thực), isub (trừ số nguyên), fsub (trừ số thực) tương ứng.
emitMULOP(self, lexeme, inType, frame): Cái này dùng để tạo code cho các phép nhân và chia.
lexeme: Ký tự đại diện cho phép toán (* hoặc /).
inType: Kiểu dữ liệu của các toán hạng.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như imul (nhân số nguyên), fmul (nhân số thực), idiv (chia số nguyên), fdiv (chia số thực).

emitDIV(self, frame): Cái này dùng để tạo code cho phép chia số nguyên (idiv).
emitMOD(self, frame): Cái này dùng để tạo code cho phép lấy số dư khi chia số nguyên (irem).
emitANDOP(self, frame): Cái này dùng để tạo code cho phép toán logic AND (iand).
emitOROP(self, frame): Cái này dùng để tạo code cho phép toán logic OR (ior).

emitREOP(self, op, inType, frame): Cái này dùng để tạo code cho các phép so sánh (ví dụ: >, <, >=, <=, !=, ==).
op: Cái ký tự đại diện cho phép so sánh.
inType: Kiểu dữ liệu của các toán hạng.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh tương ứng để so sánh.
emitRELOP(self, op, inType, trueLabel, falseLabel, frame): Cái này dùng để tạo code cho các điều kiện trong câu lệnh if.
op: Ký tự đại diện cho phép so sánh.
inType: Kiểu dữ liệu của các toán hạng.
trueLabel: Cái nhãn (label) để nhảy tới nếu điều kiện đúng.
falseLabel: Cái nhãn để nhảy tới nếu điều kiện sai.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra code để kiểm tra điều kiện và nhảy tới các nhãn tương ứng.
Tóm lại, mấy cái API này nó giúp mình "dịch" mấy cái phép toán mà mình viết trong code thành những cái lệnh mà máy tính có thể hiểu và thực hiện được.

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load
- emitALOAD(self, inType, frame)) $\Rightarrow$ [ifa]aload

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load
- emitALOAD(self, inType, frame)) $\Rightarrow$ [ifa]aload
- emitWRITEVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]store

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load
- emitALOAD(self, inType, frame)) $\Rightarrow$ [ifa]aload
- emitWRITEVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]store
- emitASTORE(self, inType, frame) $\Rightarrow$ [ifa]astore

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load
- emitALOAD(self, inType, frame)) $\Rightarrow$ [ifa]aload
- emitWRITEVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]store
- emitASTORE(self, inType, frame) $\Rightarrow$ [ifa]astore
- emitGETSTATIC(self, lexeme, inType, frame) $\Rightarrow$ getstatic

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load
- emitALOAD(self, inType, frame)) $\Rightarrow$ [ifa]aload
- emitWRITEVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]store
- emitASTORE(self, inType, frame) $\Rightarrow$ [ifa]astore
- emitGETSTATIC(self, lexeme, inType, frame) $\Rightarrow$ getstatic
- emitGETFIELD(self, lexeme, inType, frame) $\Rightarrow$ getfield

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load
- emitALOAD(self, inType, frame)) $\Rightarrow$ [ifa]aload
- emitWRITEVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]store
- emitASTORE(self, inType, frame) $\Rightarrow$ [ifa]astore
- emitGETSTATIC(self, lexeme, inType, frame) $\Rightarrow$ getstatic
- emitGETFIELD(self, lexeme, inType, frame) $\Rightarrow$ getfield
- emitPUTSTATIC(self, lexeme, inType, frame) $\Rightarrow$ putstatic

Read/Write Variables APIs

- emitREADVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]load
- emitALOAD(self, inType, frame)) $\Rightarrow$ [ifa]aload
- emitWRITEVAR(self, name, inType, index, frame) $\Rightarrow$ [ifa]store
- emitASTORE(self, inType, frame) $\Rightarrow$ [ifa]astore
- emitGETSTATIC(self, lexeme, inType, frame) $\Rightarrow$ getstatic
- emitGETFIELD(self, lexeme, inType, frame) $\Rightarrow$ getfield
- emitPUTSTATIC(self, lexeme, inType, frame) $\Rightarrow$ putstatic
- emitPUTFIELD(self, lexeme, inType, frame) $\Rightarrow$ putfield

emitREADVAR(self, name, inType, index, frame): Cái này dùng để tạo code đọc giá trị của một biến.
name: Tên của biến.
inType: Kiểu dữ liệu của biến.
index: Vị trí (chỉ số) của biến trong bộ nhớ.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như iload (đọc số nguyên), fload (đọc số thực), aload (đọc đối tượng hoặc mảng), v.v.
emitALOAD(self, inType, frame): Cái này dùng để tạo code đọc một phần tử của mảng.
inType: Kiểu dữ liệu của phần tử mảng.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như iaload (đọc số nguyên từ mảng), faload (đọc số thực từ mảng), aaload (đọc đối tượng từ mảng), v.v.
emitWRITEVAR(self, name, inType, index, frame): Cái này dùng để tạo code ghi giá trị vào một biến.
name: Tên của biến.
inType: Kiểu dữ liệu của biến.
index: Vị trí của biến trong bộ nhớ.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như istore (ghi số nguyên), fstore (ghi số thực), astore (ghi đối tượng hoặc mảng), v.v.
emitASTORE(self, inType, frame): Cái này dùng để tạo code ghi giá trị vào một phần tử của mảng.
inType: Kiểu dữ liệu của phần tử mảng.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như iastore (ghi số nguyên vào mảng), fastore (ghi số thực vào mảng), aastore (ghi đối tượng vào mảng), v.v.

emitGETSTATIC(self, lexeme, inType, frame): Cái này dùng để tạo code đọc giá trị của một biến tĩnh (static).
lexeme: Tên của biến tĩnh.
inType: Kiểu dữ liệu của biến tĩnh.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh getstatic.
emitGETFIELD(self, lexeme, inType, frame): Cái này dùng để tạo code đọc giá trị của một thuộc tính (field) của một đối tượng.
lexeme: Tên của thuộc tính.
inType: Kiểu dữ liệu của thuộc tính.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh getfield.
emitPUTSTATIC(self, lexeme, inType, frame): Cái này dùng để tạo code ghi giá trị vào một biến tĩnh.
lexeme: Tên của biến tĩnh.
inType: Kiểu dữ liệu của biến tĩnh.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh putstatic.
emitPUTFIELD(self, lexeme, inType, frame): Cái này dùng để tạo code ghi giá trị vào một thuộc tính của một đối tượng.
lexeme: Tên của thuộc tính.
inType: Kiểu dữ liệu của thuộc tính.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh putfield.

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) $\Rightarrow$ fconst, ldc

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) $\Rightarrow$ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) $\Rightarrow$ invokestatic

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) $\Rightarrow$ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) $\Rightarrow$ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) $\Rightarrow$ invokespecial

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) $\Rightarrow$ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) $\Rightarrow$ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) $\Rightarrow$ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) $\Rightarrow$ invokevirtual

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) $\Rightarrow$ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) $\Rightarrow$ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) $\Rightarrow$ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) $\Rightarrow$ invokevirtual
- emitIFTRUE(self, label, frame) $\Rightarrow$ ifgt

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) $\Rightarrow$ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) $\Rightarrow$ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) $\Rightarrow$ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) $\Rightarrow$ invokevirtual
- emitIFTRUE(self, label, frame) $\Rightarrow$ ifgt
- emitIFFALSE(self, label, frame) $\Rightarrow$ ifle

Other APIs

- emitPUSHICONST(self, input, frame) $\Rightarrow$ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) $\Rightarrow$ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) $\Rightarrow$ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) $\Rightarrow$ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) $\Rightarrow$ invokevirtual
- emitIFTRUE(self, label, frame) $\Rightarrow$ ifgt
- emitIFFALSE(self, label, frame) $\Rightarrow$ ifle
- emitDUP(self,frame) $\Rightarrow$ dup

Other APIs

- emitPUSHICONST(self, input, frame) ⇒ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) ⇒ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) ⇒ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) ⇒ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) ⇒ invokevirtual
- emitIFTRUE(self, label, frame) ⇒ ifgt
- emitIFFALSE(self, label, frame) ⇒ ifle
- emitDUP(self,frame) ⇒ dup
- emitPOP(self,frame) ⇒ pop

Other APIs

- emitPUSHICONST(self, input, frame) ⇒ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) ⇒ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) ⇒ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) ⇒ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) ⇒ invokevirtual
- emitIFTRUE(self, label, frame) ⇒ ifgt
- emitIFFALSE(self, label, frame) ⇒ ifle
- emitDUP(self,frame) ⇒ dup
- emitPOP(self,frame) ⇒ pop
- emitI2F(self, frame) ⇒ i2f

Other APIs

- emitPUSHICONST(self, input, frame) ⇒ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) ⇒ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) ⇒ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) ⇒ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) ⇒ invokevirtual
- emitIFTRUE(self, label, frame) ⇒ ifgt
- emitIFFALSE(self, label, frame) ⇒ ifle
- emitDUP(self,frame) ⇒ dup
- emitPOP(self,frame) ⇒ pop
- emitI2F(self, frame) ⇒ i2f
- emitRETURN(self, inType, frame) ⇒ return, ireturn

Other APIs

- emitPUSHICONST(self, input, frame) ⇒ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) ⇒ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) ⇒ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) ⇒ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) ⇒ invokevirtual
- emitIFTRUE(self, label, frame) ⇒ ifgt
- emitIFFALSE(self, label, frame) ⇒ ifle
- emitDUP(self,frame) ⇒ dup
- emitPOP(self,frame) ⇒ pop
- emitI2F(self, frame) ⇒ i2f
- emitRETURN(self, inType, frame) ⇒ return, ireturn
- emitLABEL(self, label, frame) ⇒ Label

Other APIs

- emitPUSHICONST(self, input, frame) ⇒ iconst, bipush, sipush, ldc
- emitPUSHFCONST(self, input, frame) ⇒ fconst, ldc
- emitINVOKESTATIC(self, lexeme, inType, frame) ⇒ invokestatic
- emitINVOKESPECIAL(self, frame, lexeme=None, inType=None) ⇒ invokespecial
- emitINVOKEVIRTUAL(self, lexeme, inType, frame) ⇒ invokevirtual
- emitIFTRUE(self, label, frame) ⇒ ifgt
- emitIFFALSE(self, label, frame) ⇒ ifle
- emitDUP(self,frame) ⇒ dup
- emitPOP(self,frame) ⇒ pop
- emitI2F(self, frame) ⇒ i2f
- emitRETURN(self, inType, frame) ⇒ return, ireturn
- emitLABEL(self, label, frame) ⇒ Label
- emitGOTO(self, label, frame) ⇒ goto

emitPUSHICONST(self, input, frame): Cái này dùng để "đẩy" một giá trị số nguyên (integer constant) lên stack.
input: Cái giá trị số nguyên mà mình muốn đẩy lên.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như iconst (đẩy các giá trị nhỏ như -1, 0, 1, 2, 3, 4, 5), bipush (đẩy giá trị từ -128 đến 127), sipush (đẩy giá trị từ -32768 đến 32767), ldc (đẩy các giá trị lớn hơn hoặc các giá trị không phải số nguyên).
emitPUSHFCONST(self, input, frame): Cái này dùng để đẩy một giá trị số thực (floating-point constant) lên stack.
input: Giá trị số thực muốn đẩy lên.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra các lệnh như fconst (đẩy các giá trị 0.0, 1.0, 2.0), ldc (đẩy các giá trị khác).
emitINVOKESTATIC(self, lexeme, inType, frame): Cái này dùng để gọi một phương thức tĩnh (static method).
lexeme: Tên của phương thức tĩnh.
inType: Kiểu dữ liệu trả về của phương thức.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh invokestatic.
emitINVOKESPECIAL(self, frame, lexeme=None, inType=None): Cái này dùng để gọi một phương thức đặc biệt (special method), ví dụ như constructor (<init>) hoặc phương thức của lớp cha.
frame: Cái "khung" để quản lý thông tin.
lexeme: Tên của phương thức (nếu có).
inType: Kiểu dữ liệu trả về của phương thức (nếu có).
Kết quả: Nó sẽ tạo ra lệnh invokespecial.

emitINVOKEVIRTUAL(self, lexeme, inType, frame): Cái này dùng để gọi một phương thức ảo (virtual method) của một đối tượng.
lexeme: Tên của phương thức ảo.
inType: Kiểu dữ liệu trả về của phương thức.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh invokevirtual.
emitIFTRUE(self, label, frame): Cái này dùng để tạo code cho lệnh rẽ nhánh "nếu đúng" (if true).
label: Cái nhãn (label) để nhảy tới nếu điều kiện đúng.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh ifgt (if greater than) hoặc các lệnh tương tự để kiểm tra điều kiện và nhảy.
emitIFFALSE(self, label, frame): Cái này dùng để tạo code cho lệnh rẽ nhánh "nếu sai" (if false).
label: Nhãn để nhảy tới nếu điều kiện sai.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh ifle (if less than or equal) hoặc các lệnh tương tự để kiểm tra điều kiện và nhảy.
emitDUP(self, frame): Cái này dùng để sao chép (duplicate) giá trị trên đỉnh stack.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh dup.
emitPOP(self, frame): Cái này dùng để lấy ra (pop) giá trị trên đỉnh stack và bỏ đi.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh pop.
emitI2F(self, frame): Cái này dùng để chuyển đổi một giá trị số nguyên thành giá trị số thực.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh i2f.
emitRETURN(self, inType, frame): Cái này dùng để tạo code cho lệnh return (trả về giá trị từ một phương thức).
inType: Kiểu dữ liệu của giá trị trả về.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh return (cho phương thức void) hoặc ireturn (cho phương thức trả về số nguyên) hoặc các lệnh tương tự cho các kiểu dữ liệu khác.
emitLABEL(self, label, frame): Cái này dùng để tạo một nhãn (label) trong code.
label: Tên của nhãn.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra một nhãn có tên là Label.
emitGOTO(self, label, frame): Cái này dùng để tạo code cho lệnh nhảy vô điều kiện (goto).
label: Nhãn để nhảy tới.
frame: Cái "khung" để quản lý thông tin.
Kết quả: Nó sẽ tạo ra lệnh goto.

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - ◁ getNewLabel(): return a new label

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - ◁ getNewLabel(): return a new label
  - ◁ getStartLabel(): return the beginning label of a scope

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - ◁ getNewLabel(): return a new label
  - ◁ getStartLabel(): return the beginning label of a scope
  - ◁ getEndLabel(): return the end label of a scope

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - ◁ getNewLabel(): return a new label
  - ◁ getStartLabel(): return the beginning label of a scope
  - ◁ getEndLabel(): return the end label of a scope
  - ◁ getContinueLabel(): return the label where a continue should come

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - ◁ getNewLabel(): return a new label
  - ◁ getStartLabel(): return the beginning label of a scope
  - ◁ getEndLabel(): return the end label of a scope
  - ◁ getContinueLabel(): return the label where a continue should come
  - ◁ getBreakLabel(): return the label where a break should come

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - getNewLabel(): return a new label
  - getStartLabel(): return the beginning label of a scope
  - getEndLabel(): return the end label of a scope
  - getContinueLabel(): return the label where a continue should come
  - getBreakLabel(): return the label where a break should come
  - enterScope()

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - getNewLabel(): return a new label
  - getStartLabel(): return the beginning label of a scope
  - getEndLabel(): return the end label of a scope
  - getContinueLabel(): return the label where a continue should come
  - getBreakLabel(): return the label where a break should come
  - enterScope()
  - exitScope()

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - getNewLabel(): return a new label
  - getStartLabel(): return the beginning label of a scope
  - getEndLabel(): return the end label of a scope
  - getContinueLabel(): return the label where a continue should come
  - getBreakLabel(): return the label where a break should come
  - enterScope()
  - exitScope()
  - enterLoop()

Frame

Tools are used to manage information used to generate code for a method

- Labels: are valid in the body of a method
  - getNewLabel(): return a new label
  - getStartLabel(): return the beginning label of a scope
  - getEndLabel(): return the end label of a scope
  - getContinueLabel(): return the label where a continue should come
  - getBreakLabel(): return the label where a break should come
  - enterScope()
  - exitScope()
  - enterLoop()
  - exitLoop()

"Frame" này giống như một cái "bảng ghi chú" hoặc là một cái "bảng quản lý" để mình theo dõi những thông tin cần thiết khi tạo code cho một phương thức.

Trong đó, nó có mấy cái "tool" (công cụ) chính như sau:

Labels (Nhãn): Mấy cái này dùng để đánh dấu các vị trí trong code. Ví dụ, mình có thể dùng nhãn để đánh dấu chỗ bắt đầu của một vòng lặp, hoặc là chỗ để nhảy tới khi một điều kiện nào đó xảy ra. Mấy cái hàm để quản lý nhãn bao gồm:
getNewLabel(): Lấy một cái nhãn mới.
getStartLabel(): Lấy nhãn bắt đầu của một phạm vi (scope).
getEndLabel(): Lấy nhãn kết thúc của một phạm vi.
getContinueLabel(): Lấy nhãn để nhảy tới khi gặp lệnh continue trong vòng lặp.
getBreakLabel(): Lấy nhãn để nhảy tới khi gặp lệnh break trong vòng lặp.
Mấy cái hàm để quản lý phạm vi và vòng lặp:
enterScope(): Bắt đầu một phạm vi mới.
exitScope(): Kết thúc một phạm vi.
enterLoop(): Bắt đầu một vòng lặp.
exitLoop(): Kết thúc một vòng lặp.
Nói chung, cái "Frame" này nó giúp mình tổ chức và quản lý các thông tin như nhãn, phạm vi, vòng lặp một cách có hệ thống, để mình có thể tạo ra code cho phương thức một cách chính xác và hiệu quả.

Frame (cont'd)

- Local variable array

Frame (cont'd)

- Local variable array
  - ◁ getNewIndex(): return a new index for a variable

Frame (cont'd)

- Local variable array
  - ◁ getNewIndex(): return a new index for a variable
  - ◁ getMaxIndex(): return the size of the local variable array

Frame (cont'd)

- Local variable array
  - ◁ getNewIndex(): return a new index for a variable
  - ◁ getMaxIndex(): return the size of the local variable array
- Operand stack

Frame (cont'd)

- Local variable array
  - ◁ getNewIndex(): return a new index for a variable
  - ◁ getMaxIndex(): return the size of the local variable array
- Operand stack
  - ◁ push(): simulating a push execution

Frame (cont'd)

- Local variable array
  - ◁ getNewIndex(): return a new index for a variable
  - ◁ getMaxIndex(): return the size of the local variable array
- Operand stack
  - ◁ push(): simulating a push execution
  - ◁ pop(): simulating a pop execution

Frame (cont'd)

- Local variable array
  - ◁ getNewIndex(): return a new index for a variable
  - ◁ getMaxIndex(): return the size of the local variable array
- Operand stack
  - ◁ push(): simulating a push execution
  - ◁ pop(): simulating a pop execution
  - ◁ getMaxOpStackSize(): return the max size of the operand stack

Frame (cont'd)

- Local variable array
  - ◁ getNewIndex(): return a new index for a variable
  - ◁ getMaxIndex(): return the size of the local variable array
- Operand stack
  - ◁ push(): simulating a push execution
  - ◁ pop(): simulating a pop execution
  - ◁ getMaxOpStackSize(): return the max size of the operand stack
- Implemented in class Frame

Local variable array (Mảng biến cục bộ): Cái này giống như một cái bảng để lưu trữ các biến cục bộ (local variables) trong một phương thức. Mỗi biến cục bộ sẽ có một cái "index" (chỉ số) để mình truy cập nó.
getNewIndex(): Lấy một cái "index" mới để dùng cho một biến cục bộ mới.
getMaxIndex(): Lấy kích thước tối đa của cái mảng biến cục bộ này.

Operand stack (Ngăn xếp toán hạng): Cái này là một cái stack (ngăn xếp) để lưu trữ các toán hạng (operands) trong quá trình tính toán. Ví dụ, khi mình thực hiện một phép cộng, mình sẽ "push" (đẩy) các toán hạng vào stack, rồi sau đó "pop" (lấy ra) để thực hiện phép tính.
push(): Mô phỏng thao tác đẩy một giá trị vào stack.
pop(): Mô phỏng thao tác lấy một giá trị ra khỏi stack.
getMaxOpStackSize(): Lấy kích thước tối đa của cái stack toán hạng này.

Implemented in class Frame (Được thực hiện trong lớp Frame): Cái này chỉ là thông tin thêm về mặt kỹ thuật, cho biết rằng tất cả những cái chức năng này được hiện thực hóa trong một cái lớp có tên là Frame.

Machine-Independent Code Generation

"Machine-Independent Code Generation" có nghĩa là "tạo code độc lập với máy". Tức là, ở giai đoạn này, mình sẽ tạo ra một dạng code mà nó không phụ thuộc vào cái loại máy tính cụ thể nào cả. Nó chỉ phụ thuộc vào cái ngôn ngữ lập trình mà mình đang dùng thôi.

- Based on the source language
- Use facilities of Frame and Intermediate Code Generation (Emitter)

Based on the source language (Dựa trên ngôn ngữ nguồn): Code được tạo ra sẽ dựa vào cái ngôn ngữ lập trình mà mình viết (ví dụ: C++, Java, Python, v.v.).

Use facilities of Frame and Intermediate Code Generation (Emitter) (Sử dụng các công cụ của Frame và Intermediate Code Generation (Emitter)): Để tạo ra code, mình sẽ dùng mấy cái công cụ mà tao đã giải thích trước đó, ví dụ như cái "Frame" (để quản lý thông tin) và cái "Emitter" (để tạo code trung gian).

BKIT-Java mapping

- A source program $\Rightarrow$ Java class
- A global variable $\Rightarrow$ a static field
- A function $\Rightarrow$ a static method
- A parameter $\Rightarrow$ a parameter
- A local variable $\Rightarrow$ a local variable
- An expression $\Rightarrow$ an expression
- A statement $\Rightarrow$ a statement
- An invocation $\Rightarrow$ an invocation

Summary

- Use BCEL to know which code should be generated

Summary

- Use BCEL to know which code should be generated
- Generate code for expressions first

Summary

- Use BCEL to know which code should be generated
- Generate code for expressions first
- Generate code for statements later

Summary

- Use BCEL to know which code should be generated
- Generate code for expressions first
- Generate code for statements later
- Good luck

Use BCEL to know which code should be generated (Sử dụng BCEL để biết code nào nên được tạo): BCEL là một cái thư viện Java giúp mình làm việc với bytecode của Java. Trong cái ngữ cảnh này, nó có nghĩa là mình có thể dùng BCEL để "soi" cái bytecode của Java, để biết được là mình cần tạo ra những cái lệnh nào, những cái chỉ thị nào.
Generate code for expressions first (Tạo code cho các biểu thức trước): Khi mình tạo code, mình nên ưu tiên tạo code cho các biểu thức trước. Biểu thức là mấy cái kiểu như là 1 + 2, a * b, x > y, v.v.
Generate code for statements later (Tạo code cho các câu lệnh sau): Sau khi tạo code cho các biểu thức xong xuôi, mình mới bắt đầu tạo code cho các câu lệnh. Câu lệnh là mấy cái kiểu như là if...else, for, while, v.v.