コンピュータシステムの理論と実装

7章 バーチャルマシン#1 スタック操作

アジェンダ

- コンパイラとバーチャルマシンについて
- スタックマシンについて
- 課題
- 9時終了です

コンパイラ

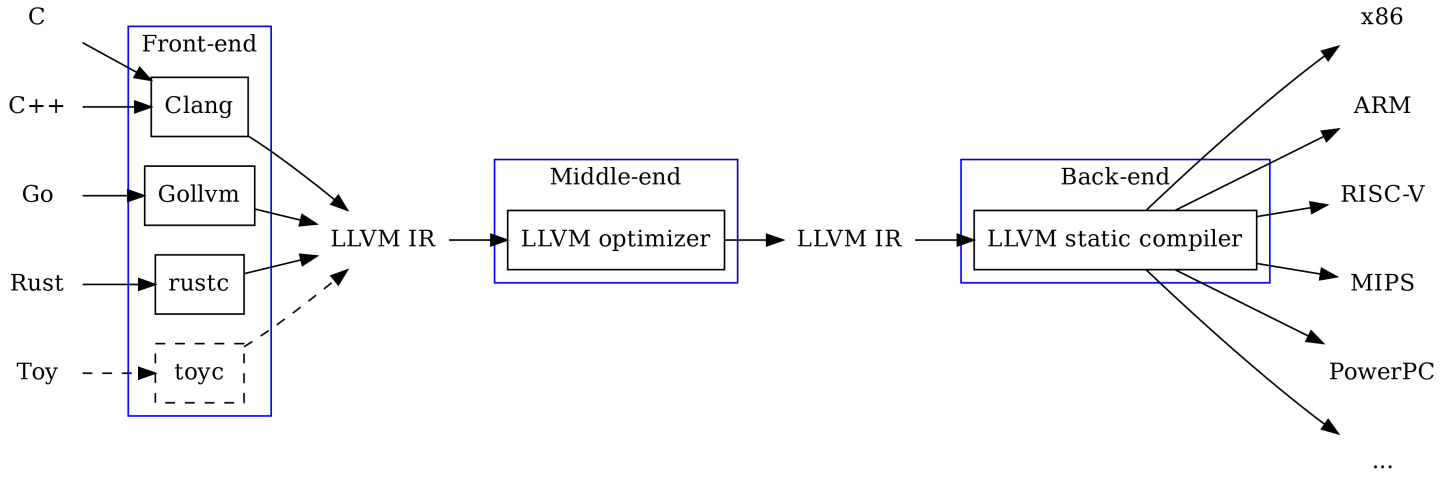
- 何かの言語で書かれたプログラムを入力とし、
  別の言語として作り出すプログラム
  - 高水準言語から機械語
  - Jack言語からHack機械語

コンパイラ

- 「高水準言語」から「機械語」へ直接変換するコンパイラ
- 組み合わせの分だけ実装が必要
  - C++ -> x86
  - C++ -> ARM
  - C++ -> RISC-V
  - Swift -> x86
  - Swift -> ARM
  - ⋮
- 手がどれだけあっても足りないね

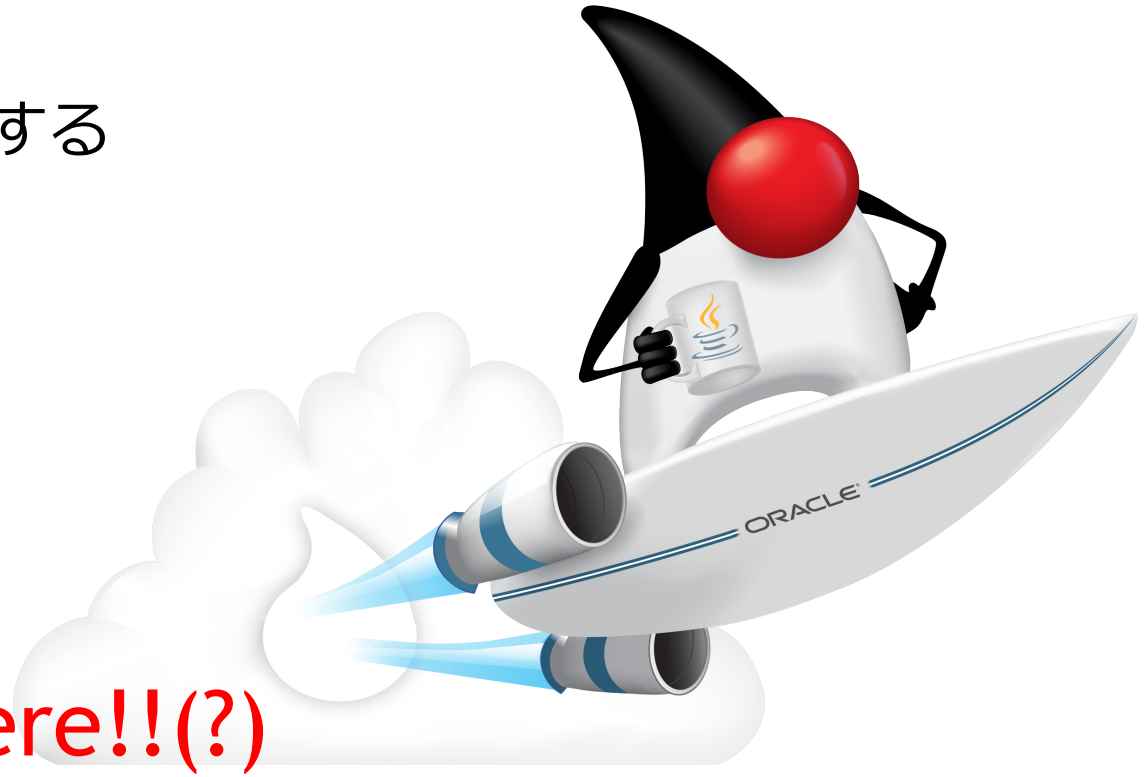
コンパイラ

- 「高水準言語」から「機械語」へ直接せず、中間表現(コード)を挟む
  - 「高水準言語」から「中間コード」へ
  - 「中間コード」から「機械語」へ
- 実装を分離できる
  - 他のプラットフォームに移植したい
  - 共通のVMであれば、違う高水準言語同士で相互呼び出しができる
  - ⋮



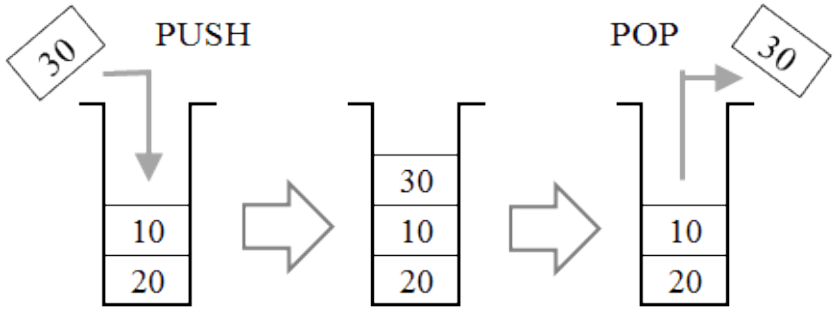
バーチャルマシン

- 中間コードを機械語に変換して実行する
  - VM言語 (中間コード)
  - バーチャルマシン (VM)
    - JVM
    - .NET Framework
    - ⋮
  - Write once, run anywhere!!(?)



スタックマシン

- オペランドや計算結果をどう扱うか(どのようなデータ構造で扱うか)
  - スタック を用いる

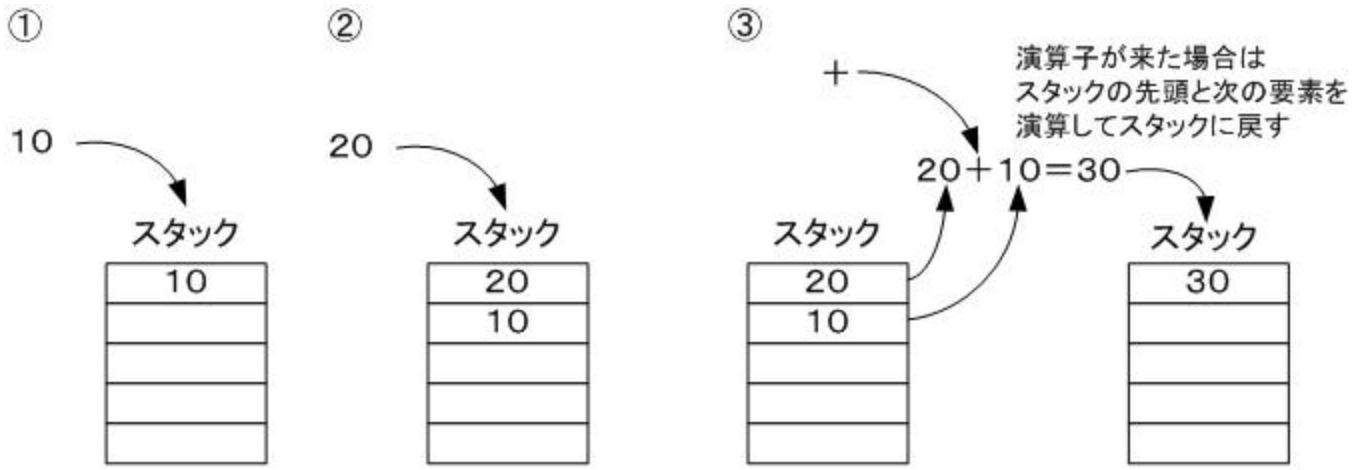


  - Push でデータを追加し
  - Pop でデータを取得する

スタックマシン

- スタックを用いた算術計算
- 例) 10 + 20

```
push 10
push 20
add
```



- 10 20 +
  - 逆ポーランド記法！(久しぶり!
  - 逆ポーランド記法に書き換えた数式はスタックで簡単に計算できる

VM言語

- 16 ビットのデータ型
  - 整数, 真偽値, ポインタ
- 4 種類のコマンド
  - 算術コマンド

```
push constant 17
push constant 17
add
⋮
```

  - メモリアクセスコマンド

```
push constant 17
⋮
```

  - プログラムフローコマンド
  - 関数呼び出しコマンド
    - 次章

```
// example
push constant 17
push constant 17
add
⋮
push constant 892
push constant 891
lt
⋮
push constant 112
sub
neg
and
push constant 82
or
not
```

算術コマンド

コマンド	戻り値（オペランドをpopした後）	コメント
add	$x + y$	整数の加算（2の補数）
sub	$x - y$	整数の減算（2の補数）
neg	$-y$	符号反転（2の補数）
eq	$x = y$ であればtrue、それ以外はfalse	等しい（equality）
gt	$x > y$ であればtrue、それ以外はfalse	～より大きい（greater than）
lt	$x < y$ であればtrue、それ以外はfalse	～より小さい（less than）
and	x And y	ビット単位
or	x Or y	ビット単位
not	Not y	ビット単位

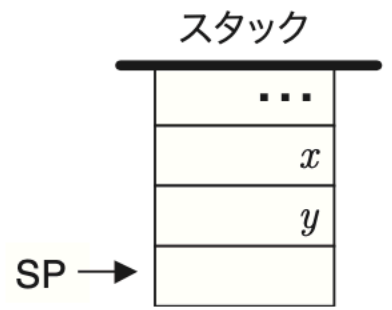


図 7-5　算術と論理に関するスタックコマンド

メモリアクセスコマンド

- push: segment[index] をスタックにプッシュする
- pop: スタックからポップし segment[index] に格納する

```
push <segment> <index>
pop <segment> <index>
```

- 例) push constant 10 とは
  - constant: 0-32767までの定数値を持つ
  - constant[10] <- 10
  - スタックにconstant[10] = 10をプッシュ

セグメント	目的	コメント
argument	関数の引数を格納する	関数に入るとVM実装によって動的に割り当てられる
local	関数のローカル変数を格納する	関数に入るとVM実装によって動的に割り当てられ、0に初期化される
static	スタティック変数を格納する。スタティック変数は、同じ.vmファイルのすべての関数で共有される	各.vmファイルに対して、VM実装により動的に割り当てられる。.vmファイルのすべての関数で共有される
constant	0から32767までの範囲のすべての定数値を持つ擬似セグメント	VM実装によってエミュレートされる。プログラムのすべての関数から見える
this that	汎用セグメント。異なるヒープ領域に対応するように作られている。プログラミングのさまざまなニーズで用いられる	ヒープ上の選択された領域を操作するために、どのような関数でもこれらのセグメントを使うことができる
pointer	thisとthatセグメントのベースアドレスを持つ2つの要素からなるセグメント	VMの関数で、pointerの0番目（または1番目）をあるアドレスに設定することができる。これにより、this（またはthat）セグメントをそアドレスの開始するヒープ領域に設定する
temp	固定された8つの要素からなるセグメント。一時的な変数を格納するために用いられる	目的に応じてVM関数によって使われる。プログラムのすべての関数で共有される

実装

- お好きな言語、お好きな実装方法で
- 実装順序 (おすすめ)
  1. push constant x　と
     9つの算術コマンド　を実装する
  2. push (constant 以外) と pop コマンドを実装する

実装 StackArithmetic

- StackArithmetic / SimpleAdd
  2つの定数を加算し、プッシュする

```
// SimpleAdd.vm
// Pushes and adds
// two constants.
push constant 7
push constant 8
add
```

とりあえず手で
やってみると

SP: スタックポインタ
スタックのアドレスを格納する

SPをインクリメント

```
// push constant 7
@7
D=A
@SP
A=M
M=D
@SP
M=M+1

// push constant 8
@8
D=A
@SP
A=M
M=D
@SP
M=M+1
```

```
// (続き)add
@SP
AM=M-1
D=M

@SP
AM=M-1
M=M+D

@SP
M=M+1
```

SPをデクリメントして
8を取得

SPをデクリメント
7と8の和を格納する

実装 StackArithmetic

- コマンドをアセンブリに対応させられる
- StackTest も演算子が増えただけ
  - あとは💪

`push constant x` →

```
// push constant x
@x
D=A
@SP
A=M
M=D
@SP
M=M+1
```

`add` →

```
// add
@SP
AM=M-1
D=M
@SP
AM=M-1
M=M+D
@SP
M=M+1
```

実装 MemoryAccess

- push, pop のセグメントの対応を行う
  - P145 と P156 を参照すること

セグメント	目的	コメント
argument	関数の引数を格納する	関数に入るとVM実装によって動的に割り当てられる
local	関数のローカル変数を格納する	関数に入るとVM実装によって動的に割り当てられ、0に初期化される
static	スタティック変数を格納する。スタティック変数は、同じ.vmファイルのすべての関数で共有される	各.vmファイルに対して、VM実装により動的に割り当てられる。.vmファイルのすべての関数で共有される
constant	0から32767までの範囲のすべての定数値を持つ擬似セグメント	VM実装によってエミュレートされる。プログラムのすべての関数から見える
this that	汎用セグメント。異なるヒープ領域に対応するように作られている。プログラミングのさまざまなニーズで用いられる	ヒープ上の選択された領域を操作するために、どのような関数でもこれらのセグメントを使うことができる
pointer	thisとthatセグメントのベースアドレスを持つ2つの要素からなるセグメント	VMの関数で、pointerの0番目（または1番目）をあるアドレスに設定することができる。これにより、this（またはthat）セグメントをそアドレスの開始するヒープ領域に設定する
temp	固定された8つの要素からなるセグメント。一時的な変数を格納するために用いられる	目的に応じてVM関数によって使われる。プログラムのすべての関数で共有される

それぞれ ARG,LCL,THIS,THAT レジスタに対応する
レジスタにベースアドレスが格納されている
RAM[base + index]

Xxx.vm ファイルの push static 3 は
@Xxx.3 と変換できる
詳しくは本文参照. トリッキー(本文)

pointer は RAM[3~4]
temp は RAM[5~12] にそれぞれ対応する
RAM[3+index], RAM[5+index]

実装 MemoryAccess

- argument, local, this, that

```
// push local x
@x
D=A
@LCL
A=M+D // base+x
D=M   // D = RAM[base+x]
```

- pointer, temp

```
// push temp 6
@11  // @(5+6)
D=M  // D = RAM[11]
```

- static

```
// StaticTest.vm
// push static 3
@StaticTest.3
D=M
```

終わり

- お疲れさまでした。
- 一応実装はこちらにあります
  - https://github.com/lulichn/nand2tetris/tree/master/src/VMtranslator
  - 汚くてすみません