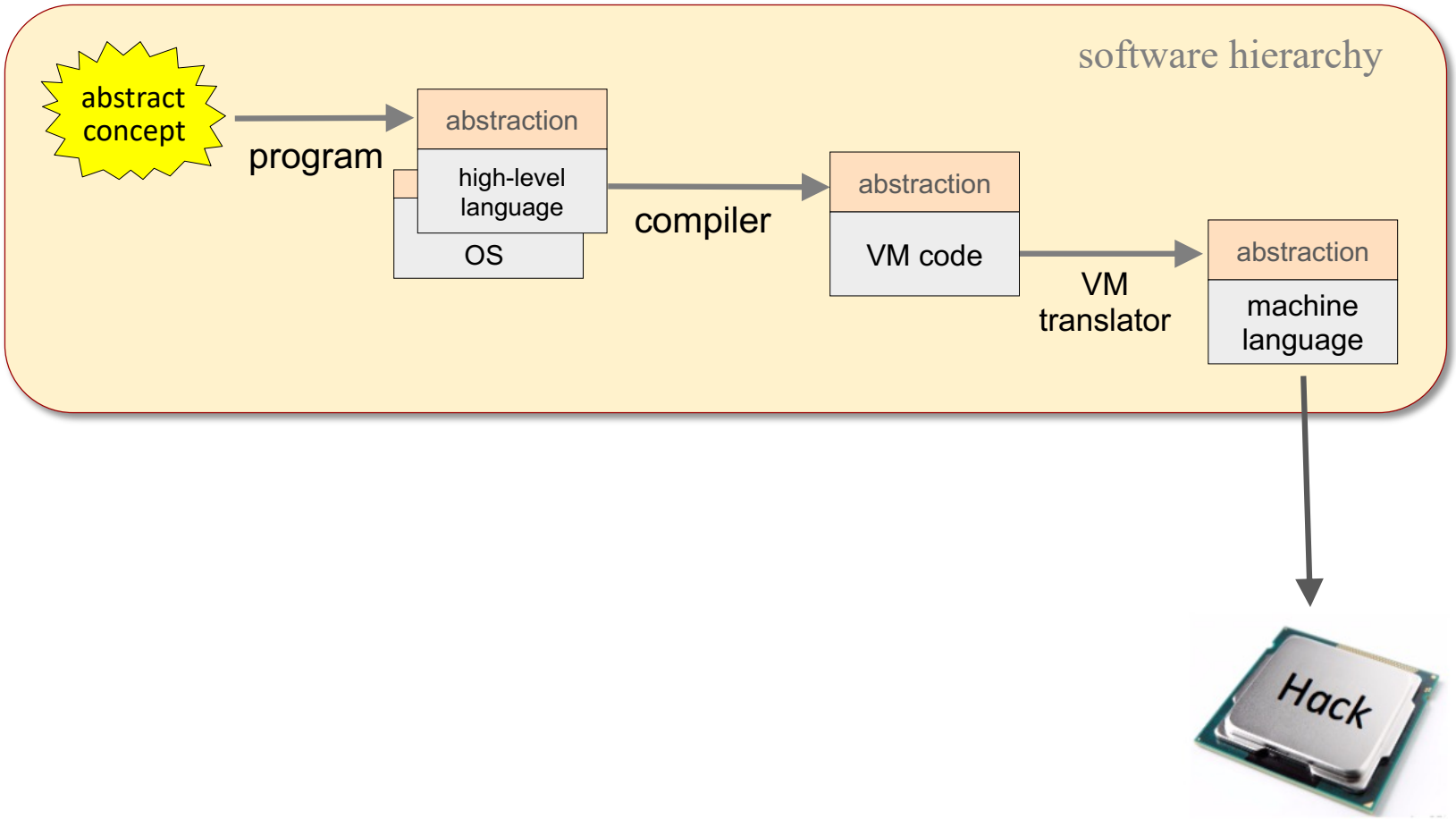
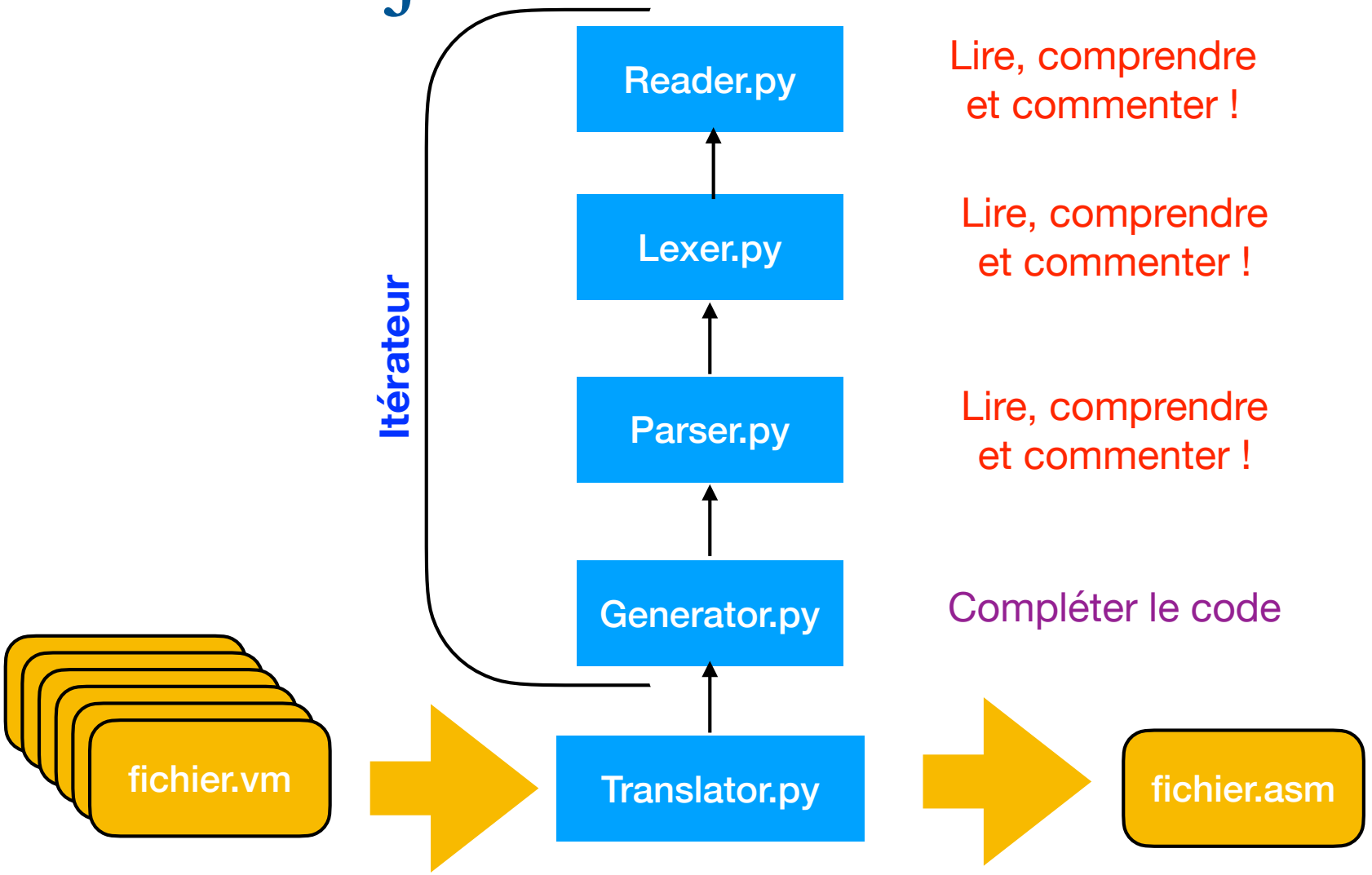
2. Machine Virtuelle

Pile et arithmétique



Projet « VM translator »



Exemple de code bien commenté :
https://github.com/python/cpython/blob/3.12/Lib/string.py

Démo !
et quelques explications du code fourni

Le language de la machine virtuelle

Push / pop commands

- `push` *segment i*
- `pop` *segment i*

Arithmetic / Logical commands

- `add`, `sub` , `neg`
- `eq` , `gt` , `lt`
- `and`, `or` , `not`

Branching commands

- `label` *label*
- `goto` *label*
- `if-goto` *label*

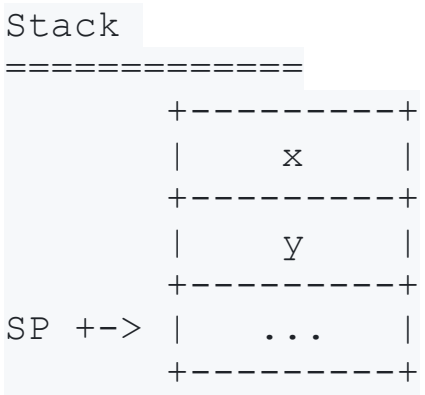
Function commands

- `Function` *functionName nVars*
- `Call` *functionName nArgs*
- `return`

où i est un entier positif et segment est un mot des mots clés suivant :
local, argument, static, constant, this, that, pointer, temp.

Opérations logiques et arithmétiques

Command	Result	Description
add	x + y	Binary Addition (2's complement)
sub	x - y	Binary Subtraction (2's complement)
neg	-y	Arithmetic Negation (2's complement)
eq	*true* if x = y, else *false*	Equality
gt	*true* if x > y, else *false*	Greater Than
lt	*true* if x < y, else *false*	Less Than
and	x & y	Bit-wise AND
or	x | y	Bit-wise OR
not	!y	Bit-wise NOT



avec true=-1 et false=0

Chaque opération dépile les paramètres dans la pile et empile le résultat

A faire !

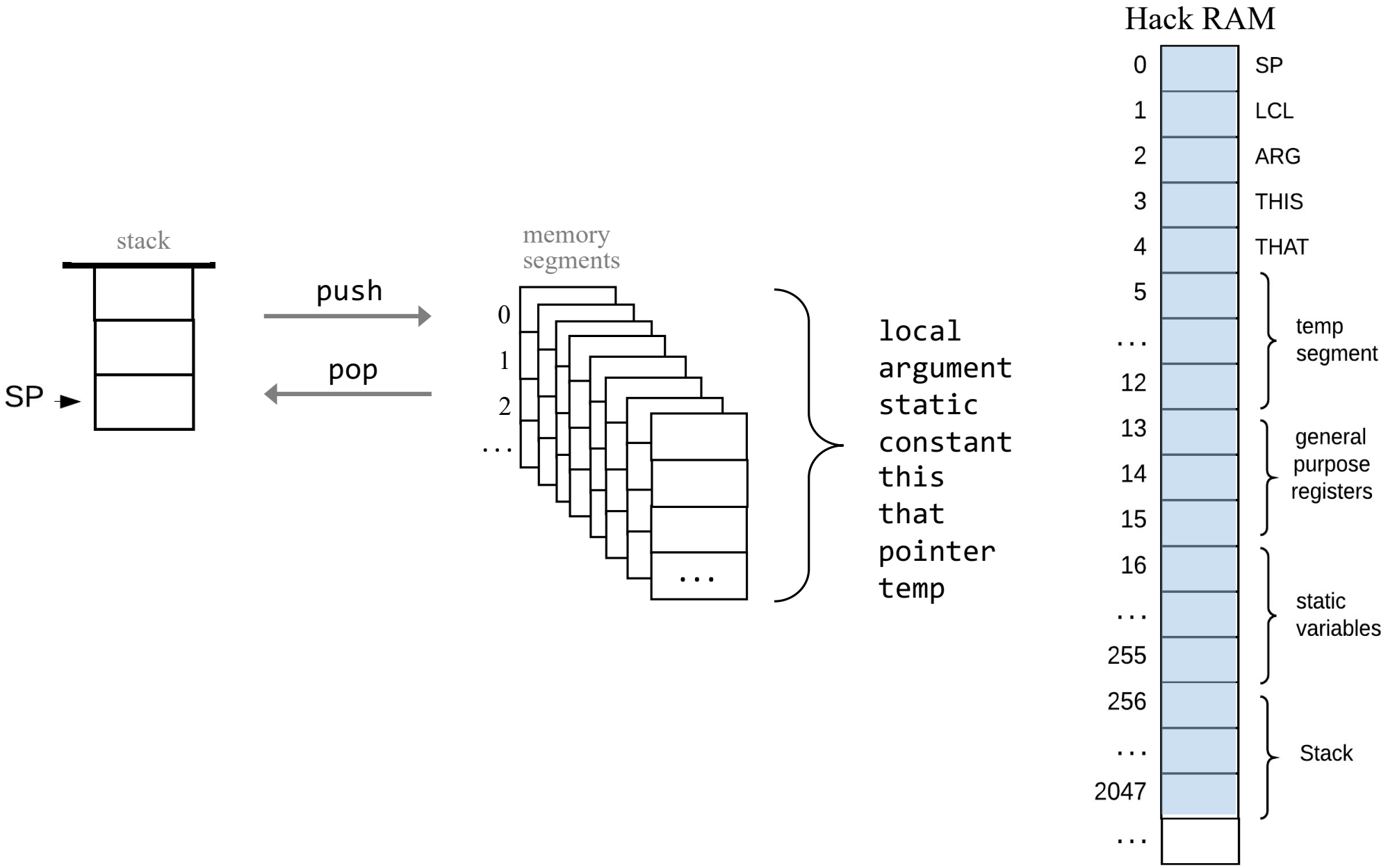
- Ajouter dans Generator.py de nouvelle fonction pour générer le code assembleur de chaque opération
- Adapter la fonction def _next(self)
- Adapter la fonction bootstrap pour réaliser vos tests

DEMO !

Ajoutons l'opération « neg » dans Generator.py et testons !

Les opérations push/pop « segment » i

avec segment = local | argument | static | constant | this | that | pointer



push constant i

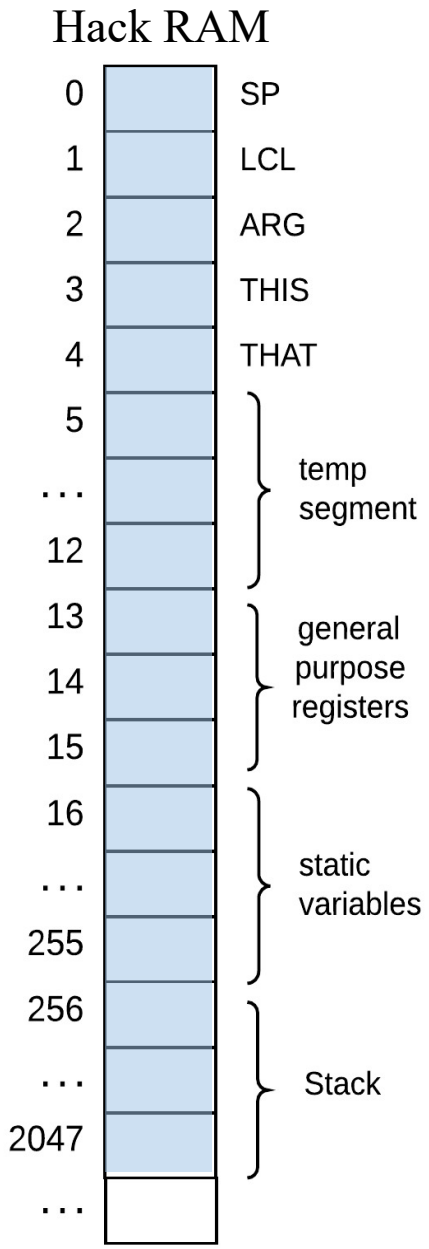
push constant i

VM translator

```
// push constant i
RAM[SP] = i
SP++
```

Pas d'opération pop constant i !

Ajoutons l'opération push constant !



push/pop {local | argument | this | that} i

pop segment i

VM translator

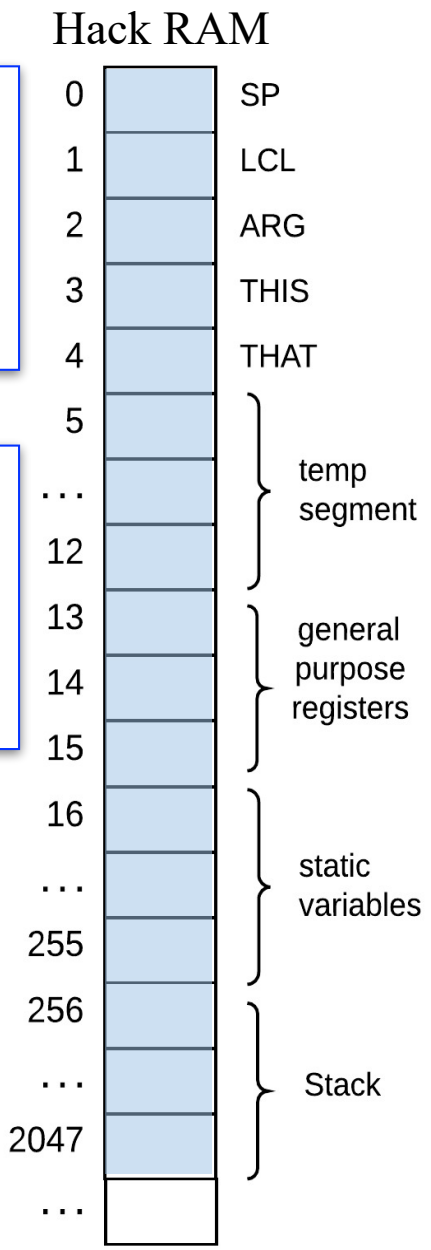
```
// pop segment i
addr = segmentPointer+i
SP --
RAM[addr] = RAM[SP]
```

push segment i

VM translator

```
// push segment i
addr = segmentPointer+i
RAM[SP] = RAM[addr]
SP++
```

avec segmentPointer
= RAM[LCL] pour « local »
= RAM[ARG] pour « argument »
= RAM[THIS] pour « this »
= RAM[THAT] pour « that »



A vous d'ajouter l'opération push segment i !

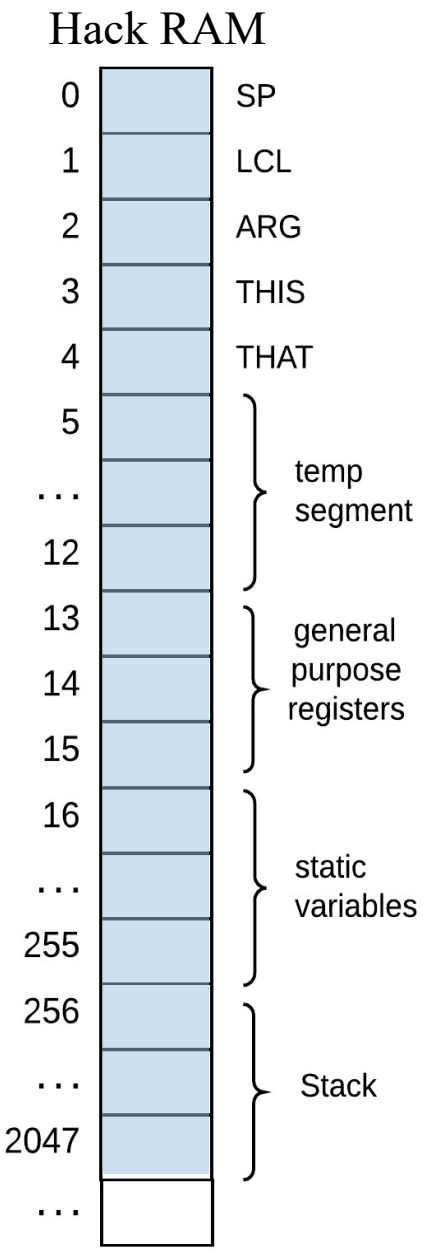
push/pop static i

pop static i → VM translator →

```
// pop static i
SP --
Foo.i = RAM[SP]
```

push static i → VM translator →

```
// push static i
RAM[SP] = Foo.i
SP++
```



avec « Foo » est le nom de la classe (nom du fichier) contenant l'instruction

A vous d'ajouter l'opération push static i !

push/pop temp i

pop temp i

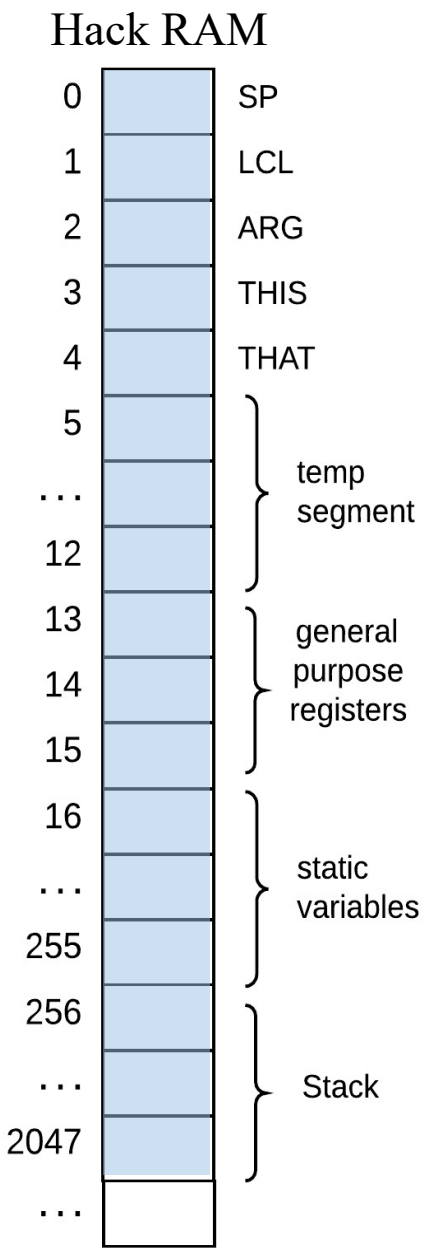
VM translator

```
// pop temp i
SP --
RAM[i+5] = RAM[SP]
```

push temp i

VM translator

```
// push temp i
RAM[SP] = RAM[i+5]
SP++
```



Le segment temp a 8 entrées : temp 0, temp 1, … , temp 7

A vous d'ajouter l'opération push temp i !

push/pop pointer {0 | 1}

pop pointer i

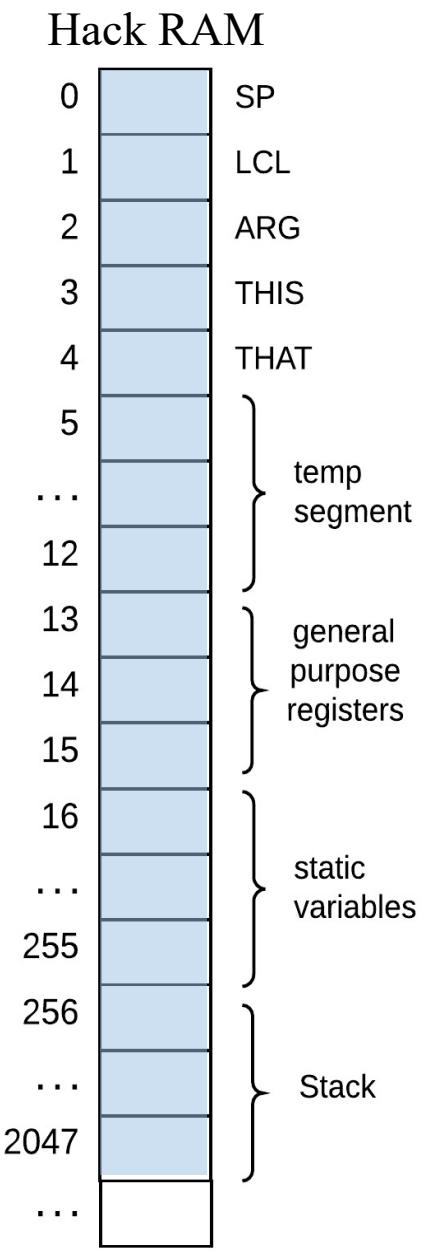
VM translator

```
// pop static i
SP --
RAM[bob] = RAM[SP]
```

push pointer i

VM translator

```
// push static i
RAM[bob] = RAM[addr]
SP++
```



avec bob = THIS si i = 0 ; THAT is i = 1

A vous d'ajouter l'opération push pointer i !

Quoi faire !

- Dès que le dépôt git est prêt, déposer un projet comprenant

  - un readme.md à la racine

  - un répertoire VMTranslator

  - dans ce répertoire, un readme.md et le code python

- Ajouter dans VMTranslator, un répertoire test et préparer des fichiers d'exemples de code vm
- Lire et comprendre le code fourni puis commenter
- Ajouter le code des opérations et prendre le temps de les tester

Déposer le projet sur git !

Votre dépôt est vide

Pour commencer, vous devez exécuter ces commandes dans votre terminal.

Configure Git pour la première fois

```
git config --global user.name "Couvreur Jean-Michel"
git config --global user.email "jean-michel.couvreur@univ-orleans.fr"
```

En train de travailler avec ton dépôt

Je veux juste cloner ce dépôt

Si vous voulez simplement cloner ce dépôt vite exécuter cette commande dans votre terminal.

```
git clone https://jean-michel.couvreur@pdicost.univ-orleans.fr/git/scm/lmt/projetmt.git
```

Mon code est prêt pour être envoyé

Si tu as déjà du code prêt pour être envoyé à ce dépôt alors exécute ceci sur ton terminal.

```
cd existing-project
git init
git add --all
git commit -m "Initial Commit"
git remote add origin https://jean-michel.couvreur@pdicost.univ-orleans.fr/git/scm/lmt/projetmt.git
git push -u origin master
```

Mon code est déjà suivi par Git

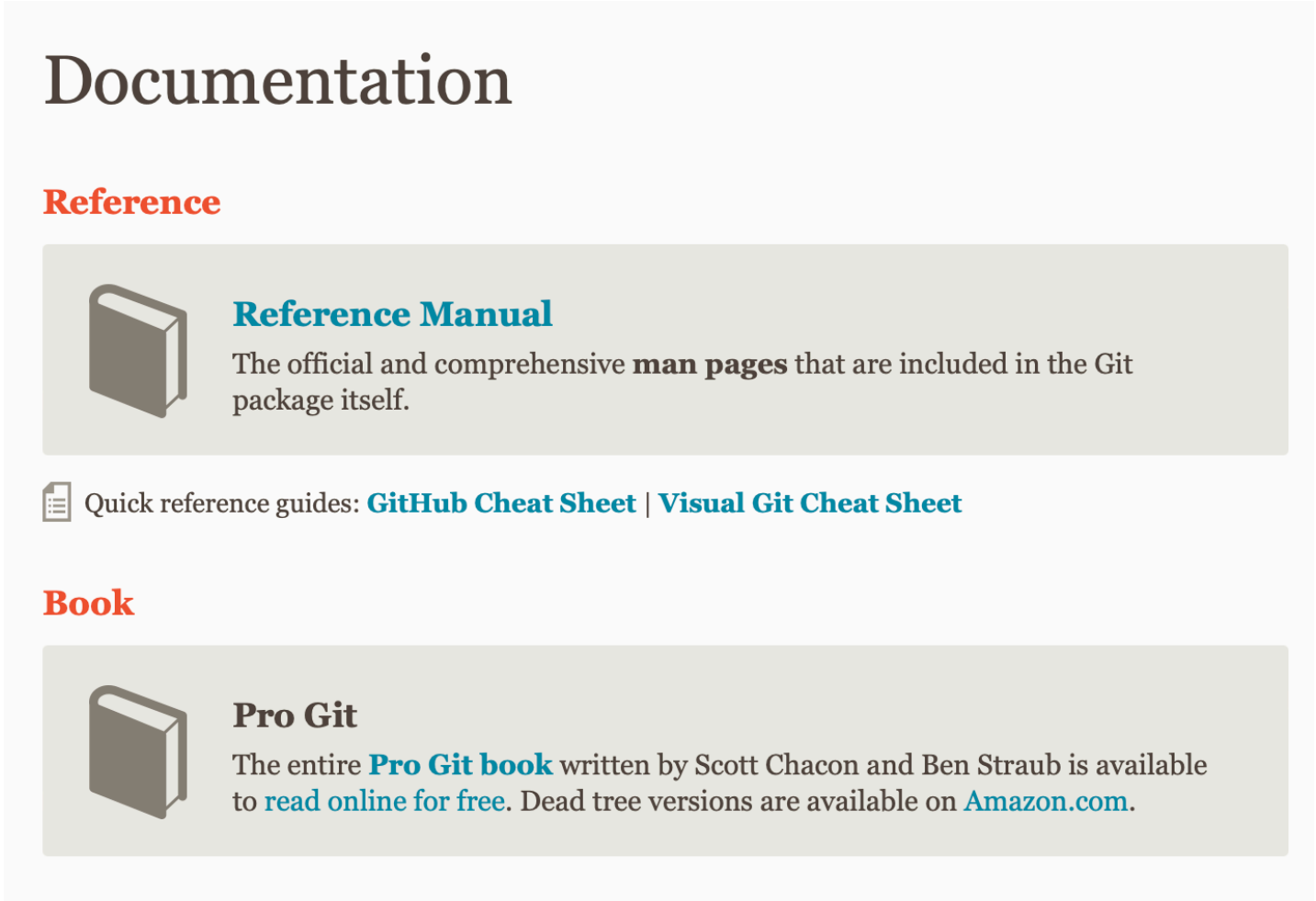
Si ton code est déjà suivi par Git alors établis ce dépôt comme ton «origine» vers où envoyer.

```
cd existing-project
git remote set-url origin https://jean-michel.couvreur@pdicost.univ-orleans.fr/git/scm/lmt/projetmt.git
git push -u origin --all
git push origin --tags
```

Fini avec les commandes ?

DEMO !

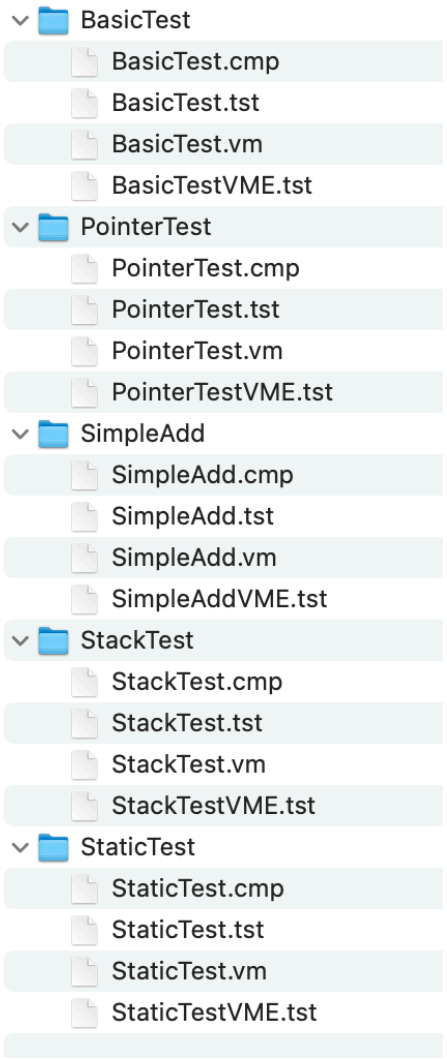
Documentation



https://git-scm.com/doc

Vérifier que votre code est ok !

Dossier 07



disponible à partir du simulateur VMEmulator

SimpleAdd.vm

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/7/StackArithmetic/SimpleAdd/SimpleAdd.vm

// Pushes and adds two constants.

push constant 7
push constant 8
add
```

Générer SimpleAdd.asm avec votre outil

Utiliser le simulateur VMEmulator pour vérifier que le résultat est correct

DEMO !

Attention ! Pour ces tests, bootstrap doit vide