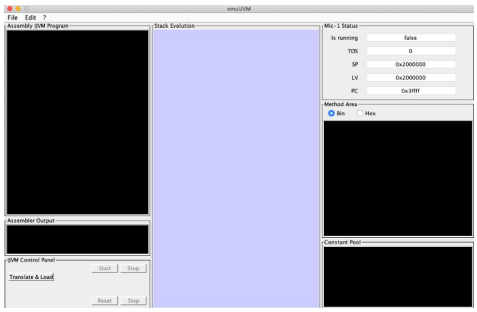
Exercice 1. Découverte du simulateur

a. Récupérez dans Celène le simulateur IJVM en java.

b. Lancez-le : `java –jar Emulateur_IJVM_V2.jar`

Vous obtenez cet affichage et cette fenêtre :

```
---MAL STANDARD OUTPUT---
******MAL Instruction with address assigned
IN    CS    NA    Instruction
80    1      2     PC=PC+1
....
00110000100000111100001000010100000
00110000100000000000000000000001111
00000000010000110101000000100001000
---END MAL STANDARD OUTPUT---
```



Nous étudions plus en détail l'affichage de ce simulateur plus tard dans la période.

Observez le simulateur. Repérez les espaces d'information et de saisie.

Où trouve-t-on :

- **i.** L'éditeur
- **ii.** L'affichage de la pile système
- **iii.** les registres
- **iv.** l'espace mémoire du programme executable
- **v.** la zone mémoire dédiée aux constantes

Faites le lien avec la page 22 du cours : Où peut-on observer les : « pool de constantes », « bloc de variables locales », « pile d'opérandes », « zone méthodes », LV, SP, PC …

Exercice 2. Premier code IJVM

ATTENTION le simulateur plante parfois, il faut donc sauvegarder régulièrement vos codes IJVM, par exemple dans un fichier texte, de plus garder une trace annotée de chacun de vos essais ne nuit pas et même n'est pas sans faciliter grandement vos révisions et votre apprentissage.

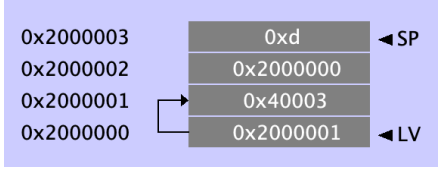
ATTENTION le simulateur n'accepte que les instructions en majuscules et sort une erreur peu explicite sur les espaces qui sont laissés après les mots clefs comme **`.main`** *ou bien* **`.var`** *etc.*

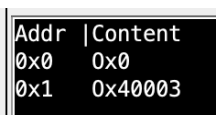
Exemple de message pour un espace après le .main : **main method not found**

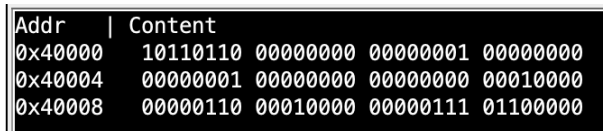
a. Saisissez un code simple dans l'éditeur :

```
.main
BIPUSH 6
BIPUSH 7
IADD
.end-main
```

b. Lancez la traduction. Observez le simulateur , lancez la traduction et l'exécution et interprétez comme vous le pouvez ce qui a changé.







c. Avec l'aide de votre enseignant interprétez :

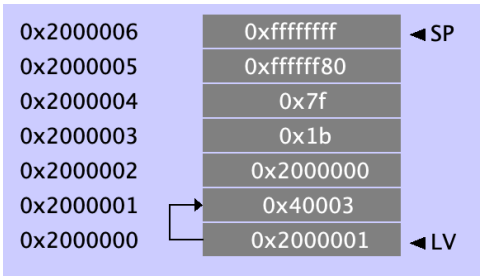
- **i.** 0xd

ii. SP
iii. LV
iv. 0x2000000
v. 0x2000003
vi. 0x4000c (dans PC)
vii. TOS
viii. 0x40003 (dans Constant Pool et dans la pile)

Exercice 3. Observez l'état de la pile à chaque étape de la liste d'instructions suivantes

```
.main
BIPUSH 1
DUP
BIPUSH 2
SWAP
BIPUSH 3
SWAP
BIPUSH 4
DUP
.end-main
```

Exercice 4. Proposez un code IJVM qui permet d'obtenir cette sortie dans le simulateur :



Exercice 5. Arithmétique de base : écrivez un code qui fait le calcul suivant : ((3 + -4) + 2)*2 + 5 – 8

Exercice 6. Logique de base : expliquez l'état de la pile à la fin de la liste d'instructions suivantes

```
.main
BIPUSH 12
BIPUSH 7
BIPUSH 12
BIPUSH 7
IAND

BIPUSH 12
BIPUSH 7
BIPUSH 12
BIPUSH 7
IOR
.end-main
```

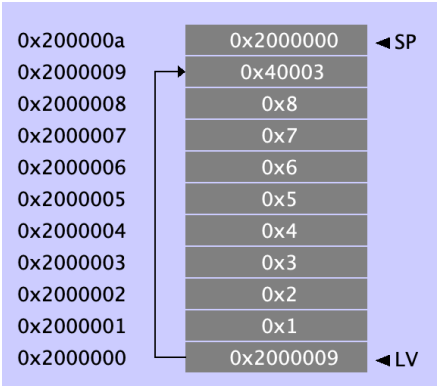
Exercice 7. Ajoutez dans le code une partie déclaration de variables :

```
.var
i
j
.end-var
```

a. Où faut-il ajouter ces variables dans le code ? Quelle est leur portée ?

b. Cela change-t-il quelque chose dans l'affichage du simulateur ? Où ? Quand?
c. Ajoutez d'autres variables et observez ce que cela change ?

Utilisez ces variables avec des instructions appropriées : ISTORE ILOAD
d. Vérifiez où sont stockées les valeurs ?



Exercice 8. Proposez un code IJVM qui permet d'obtenir cette sortie dans le simulateur :

Exercice 9.

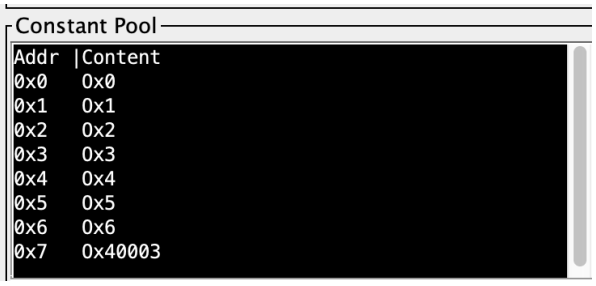
a. Que fait le code de haut niveau suivant ?
b. Traduisez instruction par instruction en IJVM ce code JAVA sans réfléchir de façon globale.
c. Optimisez-le en IJVM en tirant parti de l'expressivité supérieure de l'assembleur vis-à-vis de la pile système. Combien gagne-t-on d'instructions ?

```
main(){
i=1;
k=2;
aux=i;
i=k;
k=aux;
}
```

Exercice 10. Ajoutez dans le code une partie déclaration de constantes :

```
.constant
X  100
.end-constant
```

a. Où faut-il ajouter ces constantes dans le code ?
b. Cela change-t-il quelque chose dans l'affichage du simulateur ? Où ? Quand?
c. Ajoutez d'autres constantes et observez ce que cela change ?
d. Proposez un code IJVM qui permet d'obtenir dans le simulateur l'affichage ci-contre.
e. Initialiser des variables x, y, z à 1023, 1024, et -1024 respectivement.
f. En relisant le cours, additionner une constante égale à **2 milliards** à elle-même. Le résultat en sommet de pile est-il celui attendu, expliquez ce résultat ?



Exercice 11.

a. Quel est le code assembleur IJVM utilisant le BIPUSH pour le code JAVA suivant :

```
main(){
i=1;
i=i+12;
}
```

b. Optimisez ce code en utilisant l'instruction IINC (voir cours page 26)
c. Même question en remplaçant `i=i+12;` par `i=i-12;` puis par `i=i+128;` puis par `i=i-128;` Est-ce possible ?

Exercice 12. Si vous avez fini, passez les codes du TD dans le simulateur afin de vérifier leur comportement.