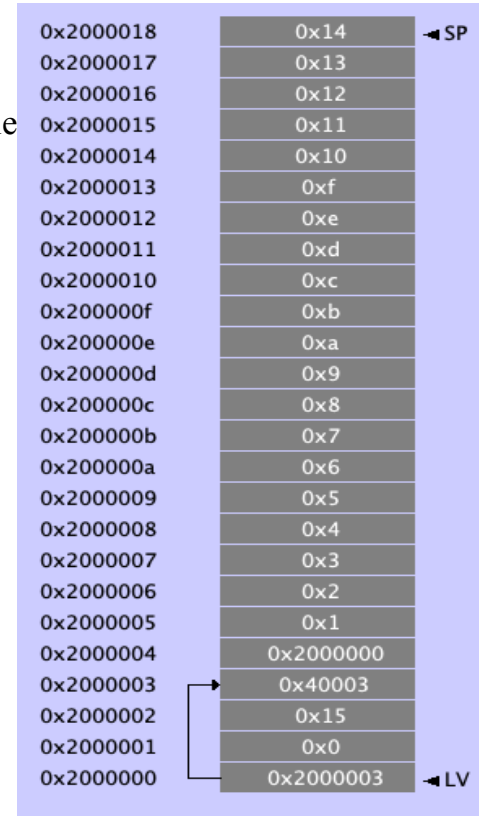
Exercice 1. Créez en assembleur IJVM dans le simulateur une condition : Si le nombre contenu dans i est positif on le met en sommet de pile, si au contraire il est négatif, on place en sommet de pile son opposé.

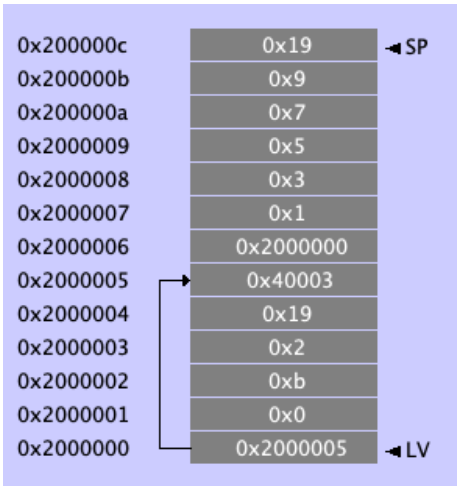
Exercice 2.

a. Créez en assembleur un programme qui place en sommet de pile, la suite de valeur allant de 1 à n. La variable n est initialisée par exemple avec la valeur 20 on obtient la pile à droite ci-contre. Le contenu des variables peut être différent de la capture d'écran suivant votre solution. Que se passe-t-il si n=-12 ?

b. Modifiez votre code pour intégrer l'usage d'une constante pour stocker n .



Exercice 3. Créez en assembleur un programme qui calcule les n premiers termes de la série arithmétique suivante $u_{n+1}=u_n+k \; avec \; u_0=1$ et les affiche dans la pile avec la somme de ces termes tout en sommet de pile. Essayez avec k=2. Le dernier terme affiché est donc u_n , si n=5 on obtient la pile ci-contre :



On a : $\sum_{n=0}^{4} u_n = 25 = 0x19$

Exercice 4. Créez en assembleur un programme itératif qui fait la multiplication de la variable i par la variable j, affiche le résultat en sommet de pile et stocke le résultat.

a. Proposez d'abord une version restreinte aux entiers positifs. Par exemple i=5 et j=3.

b. Proposez une version capable de gérer les négatifs. Par exemple i=-5 et j=-3 , ou encore i=-5 et j=3.

Exercice 5.

a. Dans ce code où sont les boucles ?

```
0x10 0x03 0x36 0x03 0x10 0x01 0x36 0x01 0x10 0x05 0x36 0x02 0x15 0x02
0x15 0x01 0x64 0x9b 0x00 0x14 0x15 0x01 0x59 0x59 0x60 0x60 0x15 0x03
0x60 0x36 0x03 0x84 0x01 0x01 0xa7 0xff 0xea 0x00 0x00 0x00 0x00
```

b. Trouvez et calculez les offsets (qui sont sur 16 bits)

c. Retrouvez le code IJVM à partie de ce code hexadécimal

d. Quelle est la valeur de la troisième variable à la fin de l'exécution ?

Exercice 6. Observez le comportement de ce code :

```
.main
.var
i
j
```

```
k
.end-var
BIPUSH 2
ISTORE i
BIPUSH 3
ISTORE j
BIPUSH 0
ILOAD i
ILOAD j
BIPUSH 10
INVOKEVIRTUAL f
ISTORE i
.end-main

.method f(x,y,z)
.var
a
.end-var
ILOAD x
DUP
IADD
ISTORE a
ILOAD a
ILOAD y
ISUB
ILOAD z
DUP
IADD
IADD
IRETURN
.end-method
```

a. Comment se comporte la pile ?
b. Où sont stockées les variables locales ?
c. Comment passe-t-on les paramètres ?
d. Où trouver la valeur de retour de la procédure ?
e. La procédure appelante peut-elle lire les variables locales de la procédure appelée ?
f. Et l'inverse ?
g. Que se passe-t-il si on ne met pas assez de paramètres ?
h. Comment est codée en binaire la méthode ? Comment la retrouve-t-on lors de l'appel ?
i. Comment à votre avis le système sait combien il y a de paramètres ? De variables locales ?
j. Comment à votre avis est restauré le registre PC à l'issue de l'appel de procédure?
k. Comment à votre avis sont restaurés les registres SP et LV ?