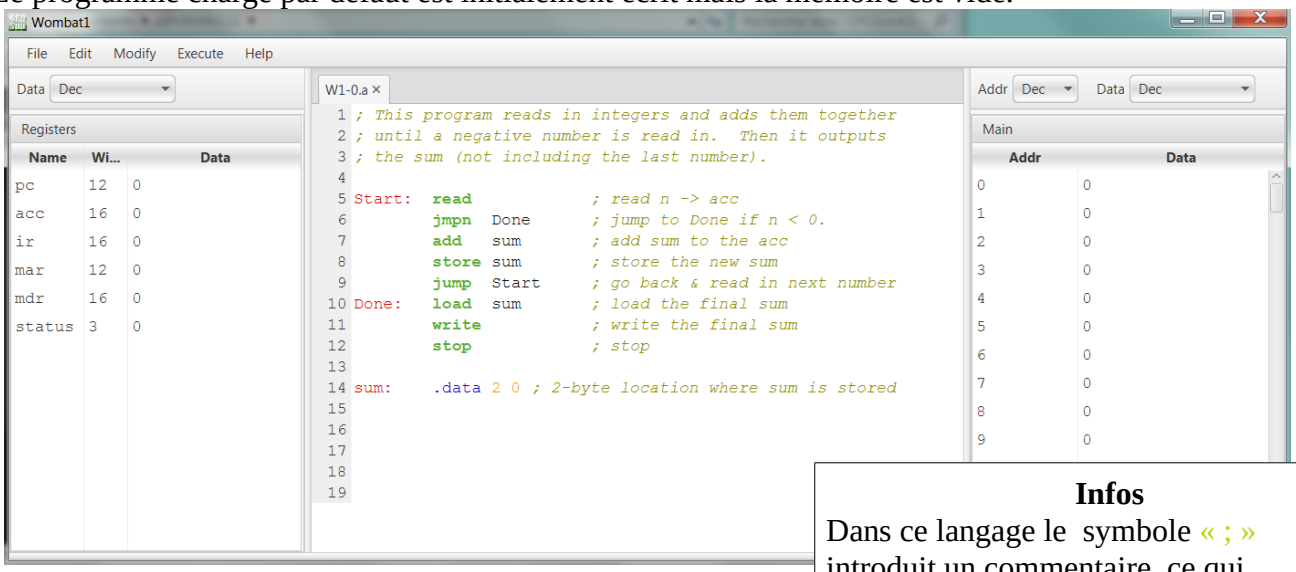
TP1 : Machines

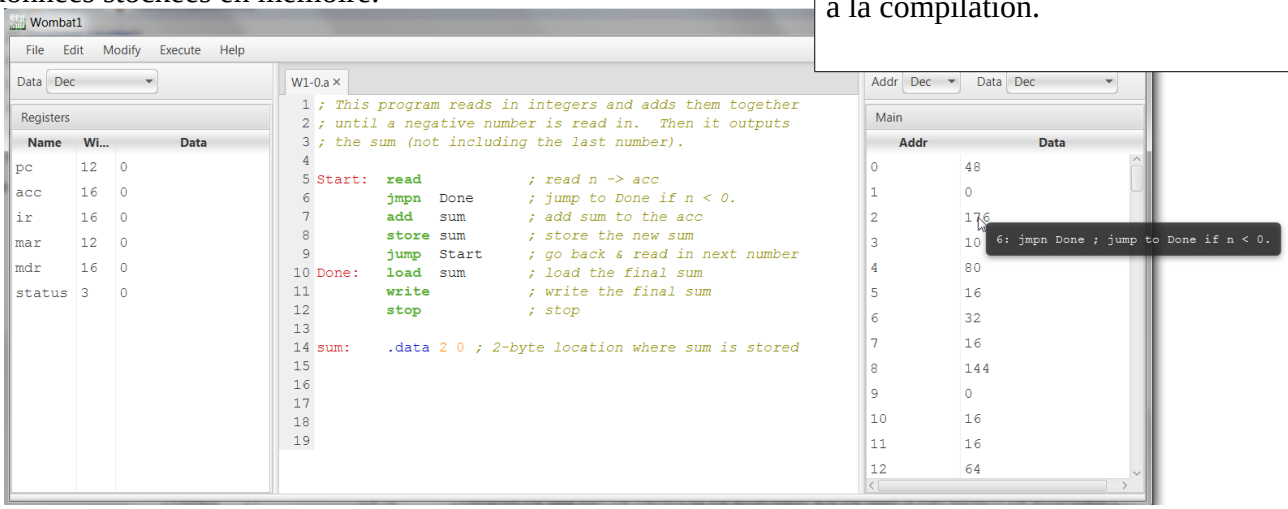
Le programme chargé par défaut est initialement écrit mais la mémoire est vide.



Infos

Dans ce langage le symbole « ; » introduit un commentaire, ce qui le suit n'est pas exécuté.
Le symbole « : » définit une étiquette qui sera remplacée par une adresse mémoire à la compilation.

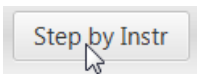
Dans le menu « execute », choisir « Assemble & load ». On voit alors que la mémoire contient à présent des nombres. On peut voir la correspondance avec les instructions en survolant les données stockées en mémoire.

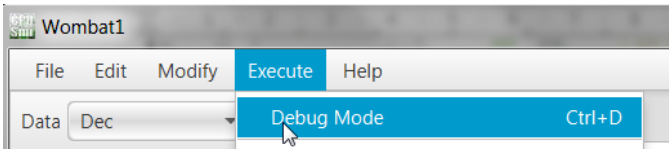


Question 1 : Dans ce langage combien d'octets sont nécessaires pour stocker une instruction et son éventuel argument ?

Tester à présent le programme en l'exécutant (Execute/Run) et entrer la suite de nombres 12,34,56,-7

Mettez ensuite le logiciel en mode « Debug » : Vous pouvez alors suivre l'exécution du programme pas à pas avec le bouton :





Question 2 : Quelle est la signification de l'octet situé à l'adresse 3 ?
Question 3 : Quels sont les significations des octets situés aux adresses 5, 7 et 11 ?

Voici un extrait de la documentation de CPUSim :

Mnemonic	Meaning
stop	stops the program execution
load	transfers data from memory to the accumulator
store	transfers data from the accumulator to the memory
read	puts the data from the IO console to the accumulator
write	sends to the IO console the data from the accumulator
add	add the data from memory to the accumulator and the result is then stored in the accumulator
subtract	subtracts the data from memory from the accumulator and the result is then stored in the accumulator
multiply	multiplies the data from the memory by the accumulator and the result is then stored in the accumulator
divide	divides the data from the memory into the accumulator and the result is then stored in the accumulator
jmpz	jump to address if the accumulator is 0
jmpn	jump to address if the accumulator is negative
jump	unconditioned jump to address

Indications :
-*Accumulator* : mémoire de travail du processeur.
-*IO* : Input/Output (entrées/sorties)

Exercice 1 (*arrêt à zéro*) : Modifier le programme précédent pour qu'il ajoute des nombres jusqu'à ce le nombre zéro soit entré.

Exercice 2 (*somme*):

a)Ecrire un programme qui demande deux nombres en entrée et qui affiche leur somme.

b)Si on entre les nombre 35000 et 36000, que se passe-t-il ?

Exercice 3 (*max*):

a) Ecrire un programme qui prend deux nombres en entrée et qui renvoie le plus grand des deux.

b) Même consigne avec trois nombres.

Défi 1(*boucle*): Le code suivant permet de stocker en mémoire trois nombres (dep, arr et pas).

Réaliser un programme qui génère comme sortie des nombres depuis 'dep' et avant 'arr' avec un pas de 'pas'.

Avec les données précédentes, on doit avoir :

```
Output:  0
Output:  2
Output:  4
Output:  6
Output:  8
```

```
dep:    .data 2 0
arr:    .data 2 10
pas:    .data 2 2
```

Défi 2 (*Moyenne*): Ecrire un programme qui demande des nombres jusqu'à ce que le nombre zéro soit saisi et qui affiche la moyenne des nombres saisis (arrondie à 1 par défaut).

Défi 3(*Moyenne+*) : Améliorer le programme précédent pour qu'il retourne la moyenne arrondie à l'unité près.