NSI 1ère - Architecture

Assembleur AQA - Travaux dirigés

qkzk

2020/07/25

TD assembleur AQA

Dans ce TD nous allons travailler avec l'assembleur AQA disponible ici.

La documentation d'origine est là et une traduction des commandes principales est disponible ici

Après avoir réalisé les premières questions, répondre aux questions suivantes.

Exercice 1 - quelques exemples

Exécuter les différents exemples qui sont proposés (menu **select**, **run**)

Hormis pour "new IO" qui est très long, suivre le parcours de l'information dans chacun des cas.

Exercice 2 - minimum

En vous inspirant de l'exemple proposé, créer un programme qui renvoie dans sa sortie le minimum de deux nombres donnés en entrée.

Exercice 3 - multiplication

On souhaite écrire un programme qui réalise une **multiplication**. (Vérifiez dans les commandes, il n'y a pas...)

- En entrée (saisie clavier), 2 nombres entiers positifs
- En sortie (affichage), leur produit.

1. Comment, à l'aide d'additions, réaliser le produit 3×4 ?
2. Ecrire un programme Python (ou un progammme en langage naturel) qui réalise ce cahier des charges.
3. Ecrire le code assembleur AQA correspondant et le tester.

Exercice 4 - suite de Fibonacci

Vous connaissez peut-être la suite de Fibonacci qui est souvent citée en informatique. En voici une définition simple :

- Son premier terme est 0.
- Son second terme est 1.
- Pour obtenir le terme suivant, on ajoute les deux précédents

Donc $u_0 = 0, u_1 = 1$ et $u_2 = u_1 + u_0 = 1 + 0 = 1$

Ensuite on obtient : $u_2 = 1 + 1 = 2, u_3 = 2 + 1 = 3, u_4 = 3 + 2 = 5, u_5 = 8, u_6 = 13$

les premiers termes, sans indices sont :

$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233$ etc.

On souhaite construire cette suite dans l'assembleur AQA.

Voici un programme Python qui affiche ligne par ligne les termes inférieurs à 255 :

```
x = 0
y = 1
while x < 255:
  print(x)
  z = y
  y = x + y
  x = y
```

1. Vérifiez que ce programme Python fonctionne.

   Remarquez bien ce qui est fait pour échanger les valeurs (`x` devient `y`)

2. Ecrire ce programme en assembleur.

Remarquez qu'on obtient les résultats en lignes, ce qui nous convient tout à à fait

Placements

Nous allons réaliser la version simple de l'exercice du placement.

Robert dispose initialement d'un capital de 1000€ qu'il place en banque.

Chaque mois il dépose 100€.

1. Combien de mois seront nécessaire pour doubler la somme initiale ?
2. Réaliser un programme en assembleur AQA afin de répondre à la question. Il demande à l'utilisateur le capital initial puis le montant versé chaque mois.

   Ensuite une boucle tourne jusqu'à atteindre le double du capital intial et on renvoie dans la sortie le nombre de tours nécessaires.

Retourner une liste de nombres

Voici le cahier des charges à respecter :

- L'utilisateur saisit 3 valeurs.
- On renvoie ensuite ces valeurs dans l'ordre contraire.

Exemple de saisies :

```
3 5 8
```

Sortie correspondante :

```
8 5 3
```

Division

L'assembleur AQA n'a pas d'implémentation de la division entière.

Étant donnés deux entiers positifs n et q, nous allons écrire un programme AQA qui calcule le quotient et le reste de la division entière $n = p \times q + r$ avec r plus petit que q.

Par exemple : $n = 27$ et $q = 6$ alors $p = 4$ et $r = 3$ car $27 = 4 \times 6 + 3$.

Le principe est de réaliser des soustractions successives jusqu'à ce que le reste soit inférieur au quotient.

À chaque étape on soustraie $n \leftarrow n - q$

Étape	p	n	$n < q$
0	0	27	Faux
1	1	21	Faux
2	2	15	Faux
3	3	9	Faux
4	4	3	Vrai

Le programme renvoie alors 4 et 3.

1. Combien de registres sont nécessaires ?
2. Écrire l'algorithme en langage naturel (ou en Python)
3. Écrire un programme en assembleur AQA pour réaliser la division.
4. Améliorer le programme en empêchant la saisie de valeurs négatives. Le programme quitte immédiatement. On peut utiliser la table ascii pour afficher un message :

   ```
   MOV r3, #69 //E pour erreur
   OUT r3, 7
   ```