ARSE01-ACT1 LA LANGAGE ASSEMBLEUR

Les registres

- un **registre** est un emplacement de mémoire interne à un processeur. Les registres sont les mémoires les plus rapides d'un ordinateur mais dont le coût de fabrication est le plus élevé.
- les registres sont utilisés pour stocker des opérandes et des résultats intermédiaires lors des opérations effectuées dans l'UAL (unité arithmétique et logique) du processeur.
- la plupart des PC actuels ont des registres de taille 64 bits.

Initiation à l'assembleur

Le langage **assembleur** est un langage de programmation très particulier. Il possède des instructions :

- arithmétiques : addition, soustraction, multiplication ...
- transfert de données : déplace des données du registre vers la mémoire et inversement
- rupture de séquence : des sauts dans le programme pour aller à une ligne particulière.

Simulateur assembleur

https://www.peterhigginson.co.uk/AQA/

Exemple : Voici un petit programme qui effectue une addition et stocke le résultat.

```
MOV R0, #22             # Stocke la valeur 22 dans le registre R0
MOV R1, #36             # Stocke la valeur 36 dans le regsitre R1
ADD R2, R0, R1          # Addition R0 et R1 et mets le résultat dans R2
STR R2, 24              # Stocke R2 à l'adresse mémoire 24
```

Commande	Description	Exemple	
charger	`LDR R1, 78`	`R1 <- @78`	Place la valeur stockée à l'adresse 78 dans `R1`
stocker	`STR R3, 125`	`@125 <- R3`	Place la valeur du registre `R3` à l'adresse 125
déplacer	`MOV R0, #23`	`R0 <- 23`	Place la valeur 23 dans `R0`
ajouter	`ADD R1, R0, #128`	`R1 <- R0 + 128`	Additionne 128 à `R0` et le place dans `R1`
ajouter	`ADD R0, R1, R2`	`R0 <- R1 + R2`	Additionne `R1` et `R2` et le place dans `R0`
soustraire	`SUB R1, R0, #128`	`R1 <- R0 - 128`	Soustrait 128 de `R0` et le place dans `R1`
soustraire	`SUB R0, R1, R2`	`R0<- R1 - R2`	Soustrait `R2` de `R1` et le place dans `R0`
arrêt	`HALT`		Arrêt de l'exécution du programme
saut	`B label`		La prochaine instruction se situe en `label`
comparer	`CMP R0, #23`	`R0 == 23`	Compare `R0` au nombre 23 - suivi par "saut"
saut "égal"	`BEQ label`	`if R0 == 23`	Si le dernier `CMP` est égal, saute à `label`
saut "différent"	`BNE label`	`if R0 != 23`	Si le dernier `CMP` est différent, saute à `label`
saut "plus grand"	`BGT label`	`if R0 > 23`	Si le dernier `CMP` est plus grand, saute à `label`
saut "plus petit"	`BLT label`	`if R0 < 23`	Si le dernier `CMP` est plus petit, saute à `label`
entrée clavier	`INP R0`	`R0 = input()`	Demande une entrée clavier et stocke dans `R0`
sortie affichage	`OUT R0`	`print(R0)`	Affiche le contenu de `R0`
label	`mon_label`		crée un label dans le programme pour le boucles
constante	`mavaleur: 7`	`mavaleur = 7`	crée une constante de valeur 7

Exercice 1

Expliquer ce que fait cet algorithme :

```
MOV R0, #10
MOV R1, #20
ADD R2, R1, R0
ADD R2, R2, #5
```

Exercice 2

Expliquer ce que fait cet algorithme :

```
INP R0
ADD R0, R0, #1
OUT R0
HALT
```

Exercice 3

Expliquer ce que fait cet algorithme :

```
     INP R0
     INP R1
     MOV R2, #0
boucle:
     CMP R1, #0
     BEQ fin
     SUB R1, R1, #1
     ADD R2, R2, R0
     B boucle
fin:
     OUT R2
     HALT
```

Exercice 4

Programmer en Assembleur

```
x = 0
while x < 3:
    x = x + 1
print(x)
```

Exercice 5

Expliquer ce que fait cet algorithme :

```
     MOV R12, #999
     MOV R11, #888
     INP R0
     STR R0, 40
boucle:
     CMP R0, #2
     BLT decision
     SUB R0, R0, #2
     B boucle
decision:
     CMP R0, #1
     BEQ label
     OUT R12
     B halte
label:
     OUT R11
halte:
     HALT
```

Exercice 6

Écrire en assembleur un programme qui calcule la somme des n premiers entiers naturels : $1+2+3+\ldots+n$

Exercice 7

Écrire un programme qui demande 2 entiers positifs et affiche le plus grand

Exercice 8

Écrire en assembleur un programme qui demande un entier naturel n et affiche 2^n

Exercice 9

Écrire un programme qui demande trois entiers positifs et les affiche par ordre croissant

Exercice 10

Traduire en assembleur

```
capital = 2000
for annee in range(10):
    capital = capital + 150
print(capital)
```

Exercice 11

Traduire en assembleur

```
capital = 2000
n = 0
while capital < 3000:
    capital = capital + 123
print(n)
```