NSI - Première

Assembleur Exercices

qkzk

Tous les exercices utilisent l'assembleur présenté en cours et rappelé ci-dessous.

Déplacements

Mnemonique	Exemple	Description
Charger	`LDR R1,78`	Place la valeur stockée à l'adresse mémoire 78 dans le registre R1
Stocker	`STR R3,125`	Place la valeur stockée dans le registre R3 en mémoire vive à l'adresse 125
Déplacer	`MOV R1, #23`	Place 23 dans R0

Opérations arithmétiques

Mnemonique	Exemple	Description
Ajouter	`ADD R1,R0,#128`	Additionne 128 à la valeur du registre R0, place dans R1
Ajouter deux registres	`ADD R0,R1,R2`	Additionne R1 à R2, place dans R0
Soustraire	`SUB R1,R0,#128`	Soustrait 128 de R0, place dans R1
Soustraire deux registres	`SUB R0,R1,R2`	Soustrait R2 de R1, place dans R0

Rupture de séquence

Mnemonique	Exemple	Description
Arrêter la machine	`HALT`	Arrête l'exécution du programme
Saut inconditionnel	`B 45`	La prochaine instruction se situe en mémoire à l'adresse 45
Comparer	`CMP R0, #23`	Compare R0 et le nombre 23 CMP doit précéder un branchement conditionnel
Saut "Égal"	`BEQ 78`	Si le dernier CMP est égal, saute à l'adresse 78
Saut "Différent"	`BNE 78`	Si le dernier CMP est différent, saute à l'adresse 78
Saut "Plus grand"	`BGT 78`	Si le dernier CMP est plus grand, saute à l'adresse 78
Saut "Plus petit"	`BLT 78`	Si le dernier CMP est plus petit, saute à l'adresse 78

Exercice 1

Remplacer chaque instruction suivante par son mnemonique puis dérouler le programme pas à pas.

```
    Charger la valeur 2 dans le registre 3
    Charger la valeur 5 dans le registre 1
    Ajouter les registres 3 et 1 et enregistrer dans le registre 2
    Soustraire 25 du registre 2 et enregistrer dans le registre 1
    Comparer les registres 3 et 1
    Si la comparaison est différente, sauter à FIN
    Charger la valeur 1 dans le registre 2
FIN:
    arrêter le programme
```

Quels sont les états des registres ?

Exercice 2

Dérouler le programme suivant, noter les états finaux des variables `x, y, z` puis le traduire en assembleur et recommencer.

```
x = 8
y = 12
z = x + y
if z == x:
  x = y
else:
  x = z
```

Exercice 3

Traduire le programme suivant en langage naturel et le dérouler ligne par ligne.

```
    MOV R1, #10
    MOV R2, #12
    ADD R3, R2, R1
    CMP R1, R2
    BNE Different
    SUB R3, R2, R1
    B Fin
Different:
    SUB R2, R1, R3
Fin:
    HALT
```

Exercice 4

La mémoire contient aux adresses 200 à 210 les entiers entre 0 et 10 :

adresse	valeur
200	0
201	1
...	...
210	10

```
    LDR R1, 202
    LDR R2, 203
    ADD R3, R2, R1
    STR R3, 200
    LDR R1, 200
    LDR R2, 204
    SUB R3, R2, R1
    STR R3, 203
```

Dérouler le programme suivant, on notera l'état final de la mémoire et des registres.

Exercice 5 - Boucle

Il existe de nombreuses manières d'écrire une boucle en assembleur.

Le principe consiste à renvoyer à la même adresse tant qu'une condition n'est pas remplie.

```
    MOV R1, #0
    MOV R2, #1
    MOV R3, #4
```

```
Boucle:
    CMP R1, R3
    BEQ Fin
    ADD R1, R1, R2
    B Boucle
Fin:
    HALT
```

1. Dérouler le programme précédent pas à pas. Combien de les instructions entre `Boucle:` et `Fin:` seront-elles exécutées ?
2. Modifier le programme afin que 10 tours de boucle soient réalisés.
3. Modifier le programme afin que la boucle soit infinie.

Exercice 6 - Boucle en Python

Voici un programme Python qui utilise une boucle pour calculer un montant après 10 ans sur un compte rapportant 150€ d'intérêts annuels.

```python
capital = 2000
for annee in range(10);
  capital += 150
```

1. Quel est le montant de `capital` après l'exécution du programme ?
2. Traduire ce programme en assembleur.

Exercice 7 - Compteur

Voici un programme qui compte le nombre d'années nécessaires pour doubler un capital.

```python
capital = 3000
double = 2 * capital
annee = 0
while capital < double:
  capital += 150
  annee += 1
```

1. Combien d'années sont nécessaires ? Résoudre mathématiquement la question.
2. Faire tourner en Python en le programmant sur votre calculatrice. Ajouter les instructions nécessaires pour consulter le solde final et le nombre d'années.
3. Traduire ce programme en assembleur.

Exercice 8 - Multiplication

L'assembleur dont nous disposons ne contient aucune instruction permettant de multiplier deux entiers.

Afin de réaliser une *multiplication*, il faut la programmer avec des additions.

Par exemple le produit de 3 par 12 peut être calculé par des additions de deux manières :

$3 \times 12 = 12 + 12 + 12$ ou $12 \times 3 = 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3$

De toute évidence, une approche est meilleure que l'autre...

1. Sans se soucier d'optimiser.

Compléter le programme suivant afin qu'il réalise le produit 3×12

R1: 3, R2: 12, R3: résultat de la multiplication, R4: nombre d'additions déjà réalisées.

```
MOV R1, #3
MOV R2, #12
MOV R3, #0
MOV R4, #0
```

```
  Produit:
    CMP R4, R2
    BEQ Fin
    ADD R3, R3, R1
    ...
    B Produit
Fin:
    HALT
```

2. En cherchant à minimiser le nombre d'instruction

Il est préférable de réaliser $12 + 12 + 12$ que $3 + 3 + \cdots + 3$.

Afin d'améliorer le temps de calcul, nous allons comparer les nombres et utiliser deux registres supplémentaires.

1. On charge dans R5 le plus petit de R1 et R2,
2. On charge dans R6 le plus grand de R1 et R2,
3. On réalise `R5 * R6`

Mémoire :

Adresse	Valeur
200	3
201	12

```
    LDR R1, 200
    LDR R2, 201
    MOV R3, #0
    MOV R4, #0
    CMP R1, R2
    BLT Petit
    MOV R5, R2
    MOV R6, R1
    B Produit
  Petit:
      ...
      ...
  Produit:
    CMP R4, R5
    BEQ Fin
    ...
    ...
    B Produit
Fin:
    HALT
```

Compléter le programme suivant. Vérifier qu'il exécute bien l'opération optimale.

Exercice 9 - Division Euclidienne

Nous allons réaliser la division Euclidienne pour calculer le quotient et le reste d'une division.

$20 = 7 \times 2 + 6$

La division Euclidienne de 20 par 7 a pour quotient 2 et reste 6.

Voici l'algorithme de la division Euclidienne de a par b utilisant des soustractions successives :

```
Diviser(a: entier, b: entier)
  a et b deux entiers.
  q = 0

  Tant que a >= b faire
```

```
  a = a - b
  q = q + 1

r = a

renvoyer q et r
```

1. Traduire ce programme en assembleur.
2. Vérifier qu'il renvoie le résultat correct pour les divisions de 7 par 2 et de 21 par 4