Constructions élémentaires

Christophe Viroulaud

Première - NSI

Mach 05

Observation

- En informatique, écrire un algorithme consiste à détailler toutes les étapes réalisables par une machine pour répondre à une problématique.
- Certaines constructions permettent d'écrire n'importe quel algorithme.

Définir les constructions élémentaires nécessaires et suffisantes.

Sommaire

Entrées / Sorties

À retenir

Dans le modèle de von Neumann, pour communiquer avec l'utilisateur, l'ordinateur utilise des interfaces d'entrée (clavier, ...) et de sortie (écran, ...).

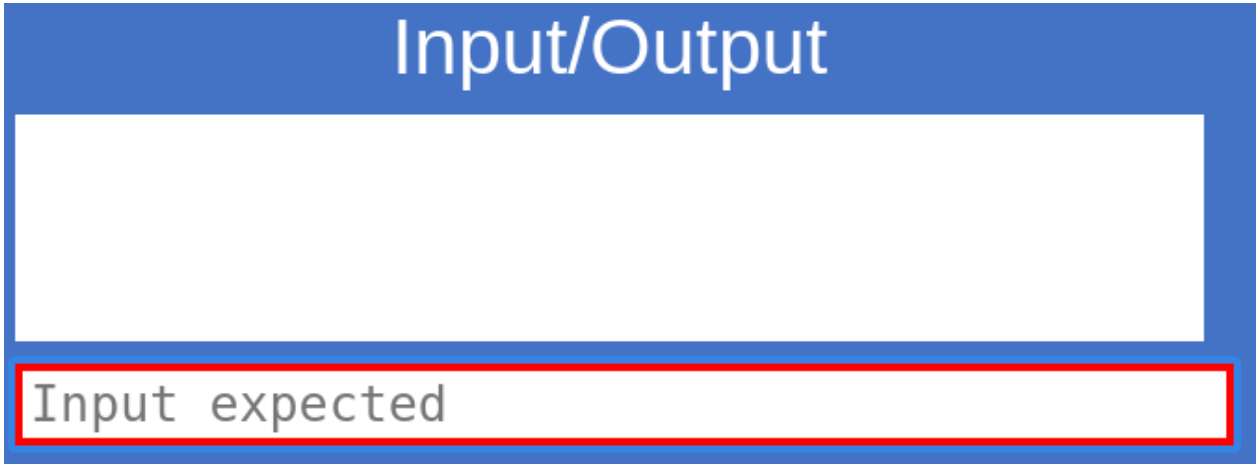


Figure 1 – Dans le simulateur, le premier cadre représente la sortie *écran* et le second l'entrée *clavier*.

```
1 // charge une entrée clavier (un nombre) dans R0
2 LDR R0, .InputNum
3 HALT
```

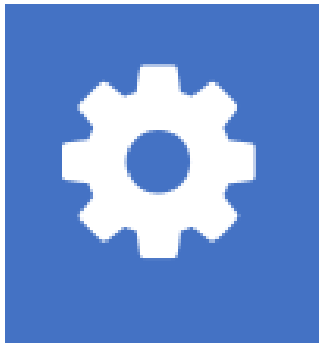


Figure 2 – **Entrée :** La roue tourne : l'ordinateur attend l'entrée d'un nombre (et la validation) pour poursuivre l'exécution.

```
1 LDR R0, .InputNum
2 STR R0, .WriteUnsignedNum
3 HALT
```

Code 1 – **Sortie :** Affichage d'un nombre dans la sortie.

Activité 1 : Écrire un programme qui :

- demande un nombre à l'utilisateur,
- lui ajoute 42,
- affiche le résultat.

```
1 LDR R0, .InputNum
2 ADD R0,R0,#42
3 STR R0, .WriteUnsignedNum
4 HALT
```

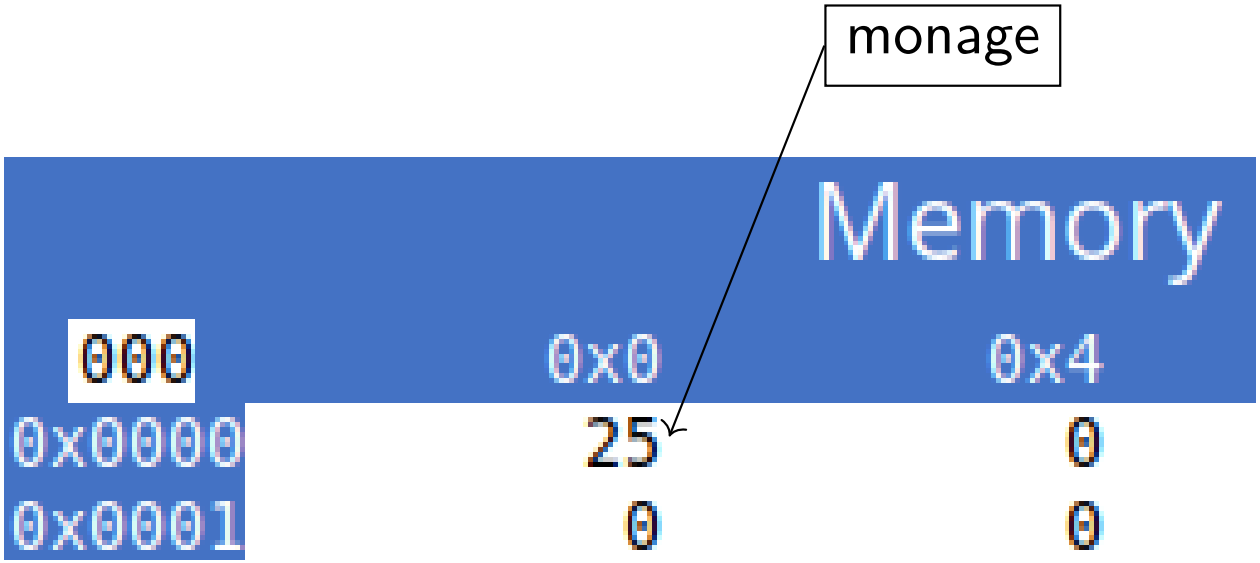
Sommaire

Variable

À retenir

Une variable est une étiquette (un label) qui référence un mot-mémoire.

```
1 monage:25
```



```
1 LDR R0, monage
2 HALT
3 monage: 25
```

Code 2 – **Utilisation d'une variable dans un programme :** les mots-mémoires réservés pour les données sont déclarés après le programme

```
1 // Copie du texte GAGNE dans R2
2 MOV R2,#GAGNE
3 // affiche dans la console de sortie
4 STR R2, .WriteString
5 HALT
6 GAGNE:.ASCIZ "Bien joué!"
```

Code 3 – Stockage et affichage d'un texte dans la sortie.

Memory

000	0x0	0x4	0x8	0xc
0x0000	3818921996	3842973932	3774873712	1852139842
0x0001	1970235936	8681	0	0

Figure 3 – La chaîne de caractères est stockée dans plusieurs mots-mémoires.

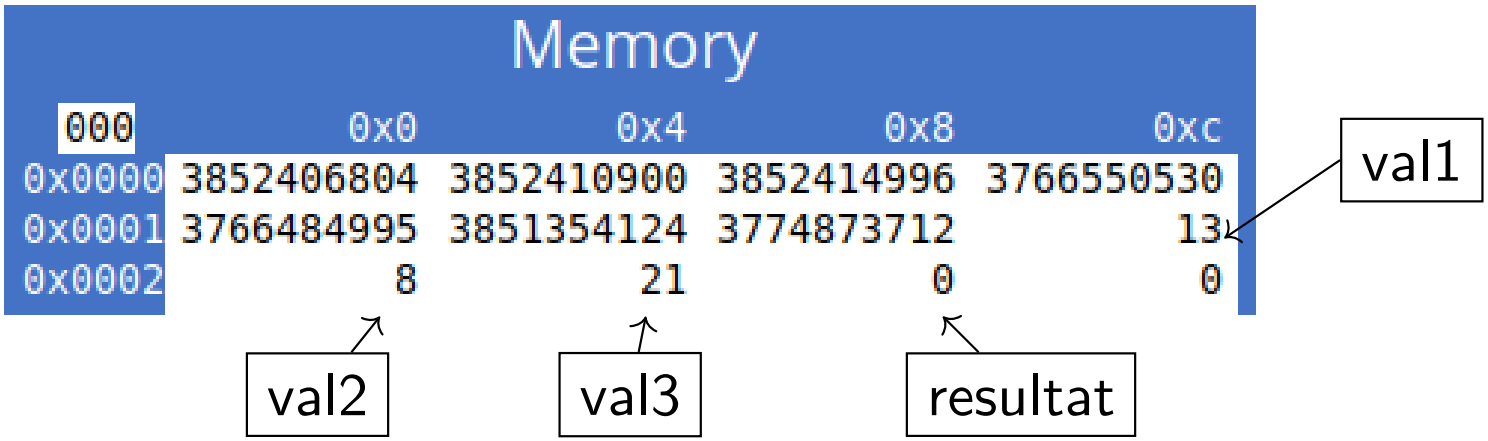
Remarque

- Selon le type de mémoire, les mnémoniques utilisés sont différents.
- Le mnémonique `MOV` manipule les données dans la mémoire registre.

Activité 2 :

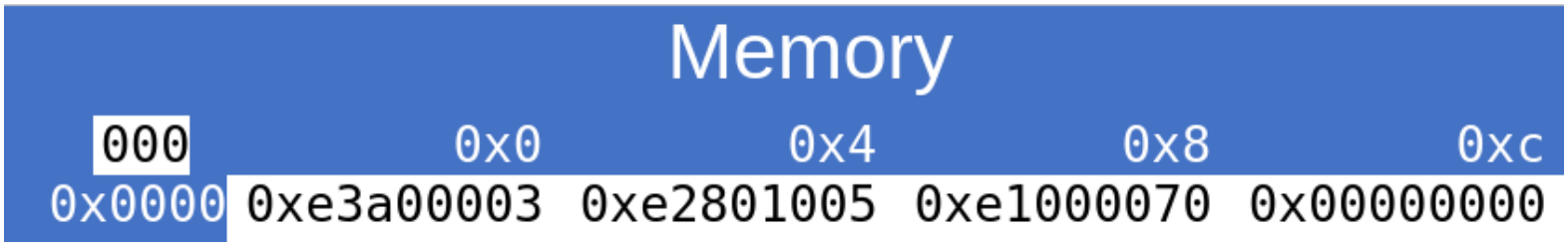
1. Stocker trois valeurs (13, 8, 21) dans la mémoire avec les étiquettes **`val1, val2, val3`**.
2. Stocker la valeur `0` dans la mémoire avec une étiquette **`resultat`**.
3. Écrire un programme qui additionne les trois valeurs et stocke le résultat à l'emplacement **`resultat`**.

```
1     LDR R1,val1
2     LDR R2,val2
3     LDR R3,val3
4     ADD R0,R1,R2
5     ADD R0,R0,R3
6     STR R0,resultat
7     HALT
8   val1: 13
9   val2: 8
10  val3: 21
11  resultat: 0
```



Sommaire

Structure conditionnelle



À retenir

Quand l'exécution d'une instruction est terminée, le processeur lit celle du mot-mémoire suivant : l'exécution d'un programme est **séquentielle** : on parle de **programmation impérative**.

```
1     MOV R0, #10
2     MOV R1, #10
3     CMP R0, R1
4     BEQ compegal
5     MOV R2, R0
6     HALT
7  compegal:
8     STR R0, mavaleur
9     HALT
10 mavaleur: 5
```

Activité 3 :

1. Exécuter le programme suivant.
2. Dans la ligne 2 remplacer la valeur 10 par 11.

```
1 CMP R0, R1
```

Figure 4 – Le processeur stocke le résultat de la comparaison

```
1 BEQ compegal
```

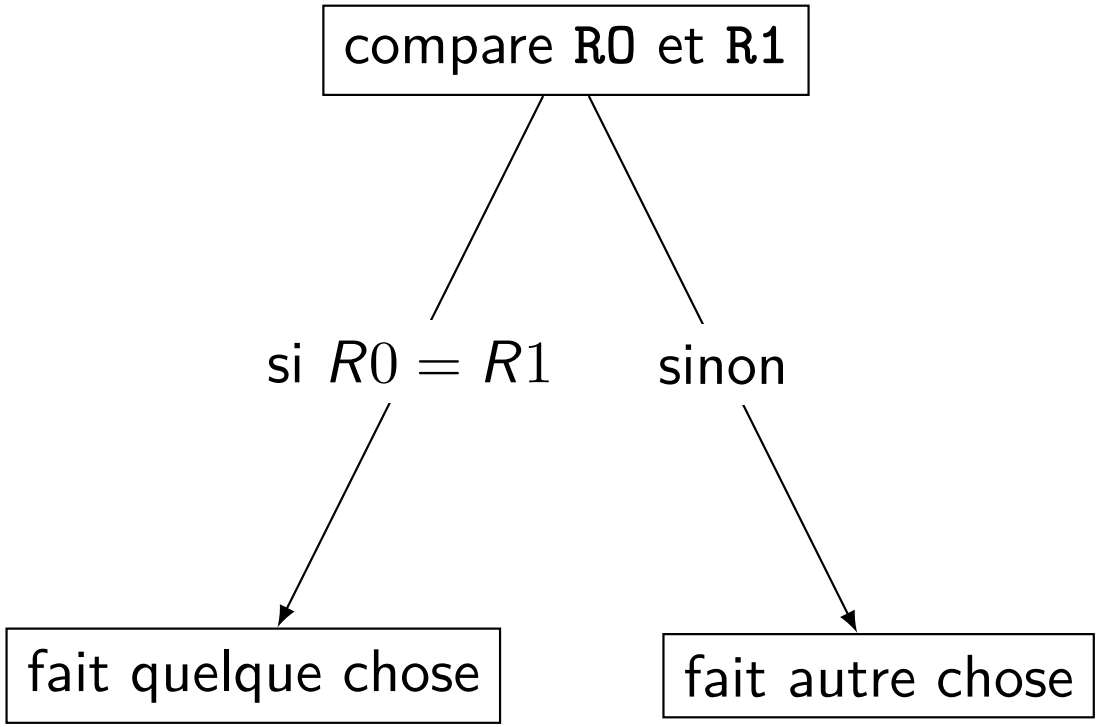


Figure 5 – Selon le résultat de la comparaison le programme réagit différemment.

Dans la documentation en pages 7-8 on trouve :

- `BEQ` : Branch if Equal,
- `B` : unconditional branch,
- `BNE` : Branch if Not Equal,
- `BLT` : Branch if Less Than,
- `BGT` : Branch if Greater Than.

À retenir

Un programme doit pouvoir agir de plusieurs manières différentes selon les résultats obtenus. On parle de **rupture de séquence** ou **structure conditionnelle**.

Activité 4 : Écrire un programme qui :

- demande un nombre à l'utilisateur,
- le compare à un nombre préalablement chargé dans le registre R1, par le programmeur,
- affiche *Bravo* si l'utilisateur trouve le bon nombre,
- affiche *Perdu* sinon.

```
1     LDR R0,.InputNum     // Copie d'une entrée dans R0
2     MOV R1,#10           // Charge 10 dans R1
3     CMP R0,R1            // Compare R0 et R1
4     BNE pasegal          // Si pas égalité, aller à pasegal
5     MOV R2,#gagne        // Copie du texte gagne dans R2
6     B fin               // Aller au label fin
7 pasegal:
8     MOV R2,#perdu
9 fin:
10    STR R2,.WriteString // Affiche R2 dans la sortie
11    HALT
12 gagne: .ASCIZ "Bravo"
13 perdu: .ASCIZ "Perdu"
```

Code 4 – Une réponse possible

Sommaire

Boucle

Activité 5 :

1. Que fait ce code ?
2. Que peut-on lui reprocher ?

```
    LDR R0,somme
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    STR R0,.WriteUnsignedNum
    HALT
somme:0
```

À retenir

Dans un programme, certaines actions peuvent être répétitives. Il est utile de disposer d'une construction qui répète des instructions tant qu'une condition est vérifiée : **la boucle.**

Activité 6 : Tester le code suivant.

```
1     MOV R11,#0          //initialise compteur
2 boucle:
3     ADD R11,R11,#1      //incrémente compteur
4     CMP R11,#3         //vérifie fin de la boucle
5     BNE boucle
6     //sortie de la boucle
7     HALT
```

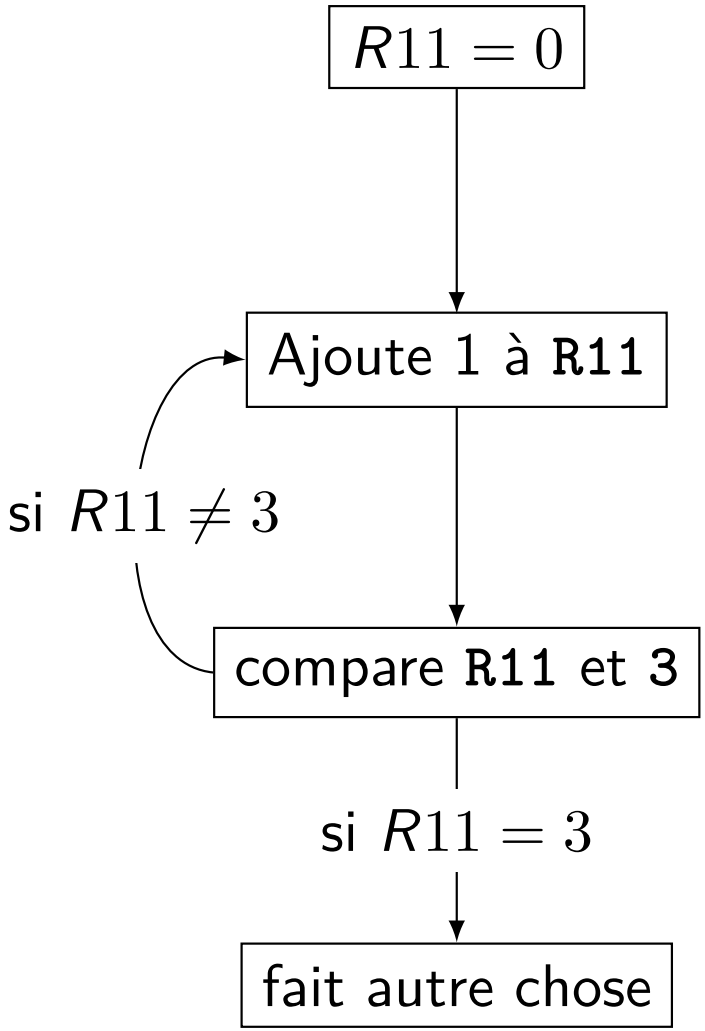


Figure 6 – Répéter une instruction

Activité 7 : Modifier alors le programme pour simplifier le code.

```
    LDR R0,somme
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    LDR R1, .InputNum
    ADD R0,R0,R1
    STR R0,.WriteUnsignedNum
    HALT
somme:0
```

```
1     LDR R0,somme
2     MOV R11,#0          //initialise compteur
3  boucle:
4     LDR R1, .InputNum
5     ADD R0,R0,R1
6     ADD R11,R11,#1      //incrémente compteur
7     CMP R11,#5          //vérifie fin de la boucle
8     BNE boucle
9     //sortie de la boucle
10    STR R0,.WriteUnsignedNum
11    HALT
12 somme:0
```