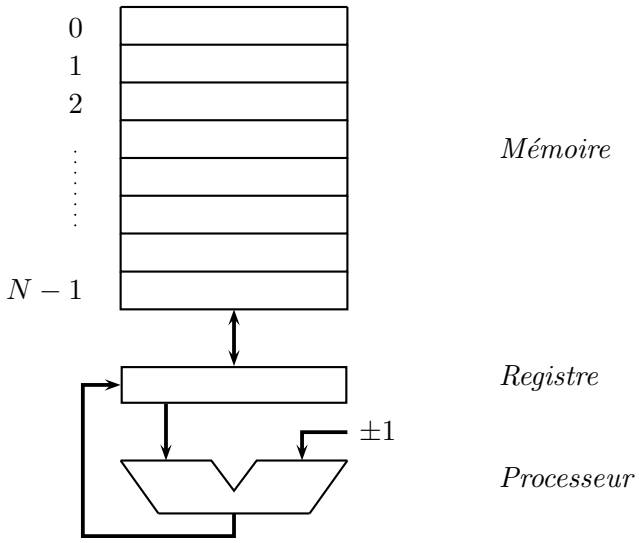
Architecture des Ordinateurs, corrigé TD 2

Exercice 1. On considère la machine RAM suivante, munie d'un registre unique :



Cette machine est munie du jeu d'intructions suivant :

`LOAD#` <valeur numérique> ;	chargement direct du registre
`LOAD@` <adresse mémoire> ;	chargement du registre depuis la mémoire
`STORE@` <adresse mémoire> ;	rangement du registre vers la mémoire
`INCR` ;	incrémentation du registre
`DECR` ;	décrémentation du registre
`JUMP` <étiquette> ;	saut inconditionnel à l'étiquette
`JZ` <étiquette> ;	saut à l'étiquette si (registre ≤ 0)
`HALT` ;	arrêt du programme

A l'aide de ce jeu d'instructions, et pour chacune des instructions C++ suivantes, écrire un programme qui la simule :

1. `A = 1;`
2. `A = A + 10;`
3. `A = A + B;`

Remarque : les variables `A` et `B` sont supposées rangées respectivement aux adresses 0 et 1 de la mémoire. Le contenu de la variable `B` n'est pas supposé être détruit après exécution de l'instruction `A = A + B;`

Correction. *Les programmes qui simulent les instructions C++ données sont respectivement :*

1. *Pour* `A = 1;` *:*

```
LOAD# 1 ;
STORE@ 0 ;
```

2. *Pour* `A = A + 10;` *: il sagit d'incrémenter dix fois l'accumulateur avant de ranger enfin la valeur obtenue à l'adresse* 0 *de la mémoire ; il est donc nécessaire de se munir d'un compteur, qu'on range (par exemple) à l'adresse* 2 *de la mémoire, et qui sera décrémenté d'autant de fois que l'accumulateur sera incrémenté dans une boucle pour obtenir la valeur finale de* `A`*.*

```
        LOAD# 10 ;
        STORE@ 2 ;
ADD :   LOAD@ 0 ;
        INCR ;
        STORE@ 0 ;
        LOAD@ 2 ;
        DECR ;
        JZ FIN ;
        STORE@ 2 ;
        JUMP ADD ;
FIN :   HALT ;
```

En remarquant qu'il est possible de factoriser l'instruction `STORE@ 2 ;` *présente simultanément avant l'entrée et en fin de boucle, on obtient un programme légèrement plus court (mais pas plus rapide en exécution) :*

```
        LOAD# 10 ;
ADD :   STORE@ 2 ;
        LOAD# 1 ;
        INCR ;
        STORE@ 0 ;
        LOAD@ 2 ;
        DECR ;
        JZ FIN ;
        JUMP ADD ;
FIN :   HALT ;
```

3. *Pour* `A = A + B;` *: On procède comme pour le programme précédent, à ceci près qu'on incrémente à partir de* `A`*, et que c'est la valeur de* `B` *qui sert de valeur initiale à un compteur indépendant (nécessaire pour ne pas détruire la donnée* `B`*), rangé à l'adresse* 2 *de la mémoire. Il est en outre nécessaire de tester si* `B` *contient la valeur* 0 *avant d'entrer dans la boucle.*

```
        LOAD@ 1 ;
        JZ FIN ;
ADD :   STORE@ 2 ;
        LOAD@ 0 ;
        INCR ;
        STORE@ 0 ;
        LOAD@ 2 ;
        DECR ;
        JZ FIN ;
        JUMP ADD ;
FIN :   HALT ;
```