Langage machine

Un ordinateur ne peut fonctionner sans une suite d'instructions, le programme, qui lui dise ce qu'il doit faire. La programmation est une des tâches les plus importantes des informaticiens. Pour programmer, il faut un langage manipulable par l'homme et interprétable par la machine.

On connaît trois niveaux de langages de programmation :
— le **langage machine** directement interprétable par la machine ;
— le **langage assembleur**, qui est du langage machine sous forme symbolique ;
— les **langages extérieurs** de programmation, synthétiques et faciles à programmer car ils sont proches du pseudo code utilisé pour les algorithmes mais ils nécessitent d'être traduits en langage machine par un programme complexe appelé compilateur.

1 Langage machine

Les instructions et les données traitées par le processeur sont des suites de 0 et de 1. C'est le langage machine. Le langage machine est le seul langage qu'un processeur puisse exécuter. Le jeu d'instructions est spécifique au processeur. Les instructions du langage machine doivent être codées en binaire et implantées dans la mémoire. Pour leur manipulation par un humain, on les représente le plus souvent en hexadécimal.

Parmi les instructions, on distingue les genres suivants :
— manipulation de registres, notamment les registres de l'accumulateur ;
— opération entre registre et mémoire, par exemple charger un registre avec une donnée mémoire, stocker le contenu d'un registre à une certaine adresse mémoire ou ajouter à un registre la donnée qui est à telle ou telle adresse ;
— branchement à une certaine adresse selon certaines conditions.
Les deux principaux inconvénients de ce langage sont :
— il est très peu parlant, si on ne connaît pas par cœur le code binaire de chaque opération, il faut sans cesse se reporter au répertoire du jeu d'instructions ;
— le programmeur doit gérer lui-même la mémoire, il doit attribuer une adresse-mémoire à chaque donnée qu'il envisage de stocker et se souvenir de toutes les adresses.

2 Langage assembleur

Le code opération de chaque instruction du langage machine est remplacé par un mot appelé mnémonique qui rappelle le mieux possible l'opération accomplie. Les mnémoniques sont à base anglaise.

L'adresse est fournie sous la forme d'un nom symbolique attribué à la donnée. C'est en somme un nom de variable. Le programmeur doit encore attribuer les adresses, mais pour désigner les données il utilise des noms qu'il peut (et doit) choisir aussi parlants que possible.

La transformation de ce code en langage machine est accomplie par un programme nommé assembleur, d'où par raccourci son nom de langage assembleur. Le premier langage assembleur n'a été écrit qu'en 1954 par Nathaniel Rochester.

Exemple : additionner deux nombres N1 et N2 et mettre le résultat dans SOMME.

Load A N1	*mettre le nombre N1 dans le registre A de l'accumulateur*
Add N2	*ajouter le nombre N2 ; l'accumulateur contient N1+N2 dans le registre A*
Store A SOMME	*ranger le contenu du registre A dans un espace mémoire nommé SOMME*

3 Exercices

Pour les exercices suivants, on imagine un processeur qui contient 4 registres, appelés A, B, C et D, et qui dispose du jeu d'instructions suivant :

- LD X adr. charge le contenu de la case mémoire d'adresse adr. dans le registre X ;
- ST X adr. stocke le contenu du registre X dans la case mémoire d'adresse adr. ;
- ADD XYZ additionne le contenu des registres X et Y puis le place dans le registre Z ;
- DEC X décrémente (-1) le contenu du registre X ;
- JUMP n saute à l'instruction numéro n du programme ;
- JUMPZ X n saute à l'instruction numéro n du programme si le contenu du registre X vaut 0 ;
- END fin du programme.

Exercice 1. Décrire ligne par ligne ce que fait ce programme. Quelle valeur se trouve dans la case mémoire 11 à la fin de ce programme ?

1 : LD A 7		
2 : LD B 8	mémoire	
3 : ADD ABA	adr.	donnée
4 : LD B 9	7	42
5 : ADD ABA	8	68
6 : LD B 10	9	47
7 : ADD ABA	10	33
8 : ST A 11		
9 : END		

Exercice 2. Expliquer ce que fait le programme ci-dessous en supposant que le nombre x contenu dans la case mémoire d'adresse 10 est strictement positif.

1 : LD A 10		
2 : LD B 10	mémoire	
3 : JUMPZ A 7	adr.	donnée
4 : DEC A	10	x
5 : ADD ABB	11	y
6 : JMP 3		
7 : ST B 11		
8 : ST A 10		

Exercice 3. Écrire un programme en assembleur équivalent aux programmes Python ci-dessous.

```
x = 10
while x > 0 :
    x -= 1
```

```
x = 5
y = 0
while x > 0 :
    x -= 1
    y -= 1
```

Exercice 4. Écrire une séquence d'instructions qui multiplie par 5 le nombre contenu dans la case mémoire d'adresse 10 et stocke le résultat dans la case mémoire d'adresse 11.

Exercice 5. Écrire un programme qui lit une valeur x dans la case mémoire 10 et calcule son opposé puis stocke le résultat à l'adresse 13. On suppose que cette valeur est un entier positif.

Exercice 6. Écrire un programme qui lit deux valeurs x et y contenues respectivement dans les cases mémoires 11 et 12, calcule la différence y-x et stocke le résultat à l'adresse 13. On suppose que ces valeurs sont des nombres entiers positifs.

Exercice 7. Modifier le programme de l'exercice 5 pour qu'il stocke 0 à l'adresse 15 si x est égal à y, sinon on y stockera la valeur de x.

Exercice 8. Écrire un programme en assembleur équivalent au programme Python ci-dessous.

```
x = 0
while x < 5 :
    x += 1
```