5.4 Les différents langages

NSI 1ère - JB Duthoit

5.4.1 Le langage machine

Programmer en langage machine est extrêmement difficile (très longue suite de 0 et de 1).

5.4.2 Langage assembleur

Pour pallier cette difficulté, les informaticiens ont remplacé les codes binaires abscons par des symboles mnémoniques (plus facile à retenir qu'une suite de "1" et de "0"),cela donne l'assembleur. Par exemple un "ADD R1,R2,#125" sera équivalent à "11100010100000100001000001111101".
Le processeur est uniquement capable d'interpréter le langage machine, un programme appelé "assembleur" assure donc le passage de "ADD R1,R2, # 125" à "11100010100000100001000001111101".
Par extension, on dit que l'on programme en assembleur quand on écrit des programmes avec ces symboles mnémoniques à la place de suite de "0" et de "1"

Voici quelques instructions :

```
LDR R1,78
```

Place la valeur stockée à l'adresse mémoire 78 dans le registre R1 (par souci de simplification, nous continuons à utiliser des adresses mémoire codées en base 10)

```
STR R3,125
```

Place la valeur stockée dans le registre R3 en mémoire vive à l'adresse 125

```
ADD R1,R0,#128
```

Additionne le nombre 128 (une valeur immédiate est identifiée grâce au symbole #) et la valeur stockée dans le registre R0, place le résultat dans le registre R1

```
ADD R0,R1,R2
```

Additionnela valeur stockée dans le registre R1 et la valeur stockée dans le registre R2, place le résultat dans le registre R0

```
SUB R1,R0,#128
```

Soustrait le nombre 128 de la valeur stockée dans le registre R0, place le résultat dans le registre R1

```
SUB R0,R1,R2
```

Soustrait la valeur stockée dans le registre R2 de la valeur stockée dans le registre R1, place le résultat dans le registre R0

```
MOV R1,#23
```

Place le nombre 23 dans le registre R1

```
MOV R0,R3
```

Place la valeur stockée dans le registre R3 dans le registre R0

```
B 45
```

Nous avons une structure de rupture de séquence, la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 45

```
CMP R0,#23
```

Compare la valeur stockée dans le registre R0 et le nombre 23. Cette instruction CMP doit précéder une instruction de branchement conditionnel BEQ, BNE, BGT, BLT

```
CMP R0,R1
```

Compare la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1.

```
CMP R0,#23
BEQ 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est égale à 23

```
CMP R0,#23
BNE 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 n'est pas égale à 23

```
CMP R0,#23
BGT 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est plus grand que 23

```
CMP R0,#23
BLT 78
```

La prochaine instruction à exécuter se situe à l'adresse mémoire 78 si la valeur stockée dans le registre R0 est plus petit que 23

```
OUT R0,4
```

Affiche en sortie la valeur du registre R0.

⚠ Pour `OUT`, le '4' de l'exemple précédent est un paramètre. Vous pouvez retourner des nombres signés (paramètre 4), des nombres non signés (paramètre 5), des hexadécimaux (paramètre 6) ou des caractères (paramètre 7).

```
INP R0,2
```

Lit une valeur et l'affecte en R0.
⚠ Le '2' est un paramètre pour typer la valeur. La valeur 2 est utilisée pour un entier signé.

```
HALT
```

Arrête l'exécutiondu programme

Exercice 5.110
Ecrire en langage assembleur :
Additionne la valeur stockée dans le registre R0 et la valeur stockée dans le registre R1, le résultat est stocké dans le registre R5.

Exercice 5.111
Ecrire en langage assembleur :
la prochaine instruction à exécuter se situe en mémoire vive à l'adresse 478. Si la valeur stockée dans le registre R0 est égale 42 alors la prochaine instruction à exécuter se situe à l'adresse mémoire 85.

Exercice 5.112
Ecrire en langage assembleur le programme suivant :

```
x = 4
y = 8
if x == 10:
    y =9
else:
    x = x+1
z = 6
```

***

Exercice 5.113
Ecrire en langage assembleur le programme suivant :

```
x = 4
y = 8
if x == y:
    y = y + x
else:
    y = y - x
```

Exercice 5.114
Ecrire en langage assembleur le programme suivant :

```
x = 4
y = 5
p = 0
while x > 0:
    p = p + y
    x = x - 1
```

☛ A la fin du programme, on a p = 20, ce qui correspond à $p = 5 + 5 + 5 + 5 = 5 \times 4 = x \times y$

Exercice 5.115
Ecrire en langage assembleur un programme qui permet de calcule 2^5.
☛ Il n'y a pas d'instruction "multiplier", il va falloir se débrouiller avec des additions...
☛ Essayez de n'utiliser qu'un seul registre, R1 par exemple !

5.4.3 Langage de haut niveau

Même si coder en assembleur est plus rapide qu'en binaire, cela reste une étape fastidieuse : manipuler les données et les placer octet par octet dans la mémoire ou dans les registres, c'est long !

De plus que chaque processeur possède des noms et des numéros d'instructions différents ! Ainsi, un code en assembleur qui fonctionne sur un processeur Pentium ne marchera pas forcément sur un AMD 64 ou sur un processeur ARM. En gros, si vous écrivez un programme sur votre ordinateur, il ne marchera pas sur le mien ! C'est embêtant...

Si le programme ne marche que sur un processeur et non sur un autre, on dit que le programme n'est pas ***portable***.

Dès les débuts de l'informatique moderne on a donc inventé un moyen pour palier à ces deux inconvénients que sont la lenteur à coder et la non portabilité : les langages de haut niveau.

Parmi les langages de haut niveau, il y a une multitude de langages différents. Le C est un exemple, mais peut-être avez vous entendu parler des langages Java, Python, PHP ? Ce sont tous des langages de haut niveau ! Il en va des préférences du programmeur d'utiliser l'un ou l'autre des langages pour faire ses programmes.

Chaque langage possède sa syntaxe et ses objectifs. Par exemple, le langage Perl est très bon pour chercher et analyser dans des fichiers textes, le langage PHP est utilisé pour générer des sites web.

Il n'est pas rare non plus de voir un programme qui soit écrit en différents langages : on utilise le langage le plus convenable pour chaque tâche.

☞ Les instructions d'un programme écrit dans un langage de haut niveau sont traduite pour être comprise par la machine. Avec le module **dis**, nous pouvons avoir une idée des instructions passées à la machine lorsque l'on écrit un code Python :

```
import dis
dis.dis('x=1;x=x+2')
```

```
  0 LOAD_CONST               0 (1)
  2 STORE_NAME               0 (x)
  4 LOAD_NAME                0 (x)
  6 LOAD_CONST               1 (2)
  8 BINARY_ADD
 10 STORE_NAME               0 (x)
 12 LOAD_CONST               2 (None)
 14 RETURN_VALUE
```

Affichage du code machine désassemblée

- Le nombre 1 est copié dans le registre
- Le contenu du registe est copié en mémoire à l'adresse x
- la valeur de x est copié dans le registre
- Le nombre 2 est copié dans le registre
- On effectue l'addition binaire
- Le résultat est copié dans la mémoire à la case adressé par x
- La valeur None est copiée dans le registre
- Puis elle est renvoyée.

Exercice 5.116

utilisez et intérpréter le résulats du module dis avec le code Python suivant :

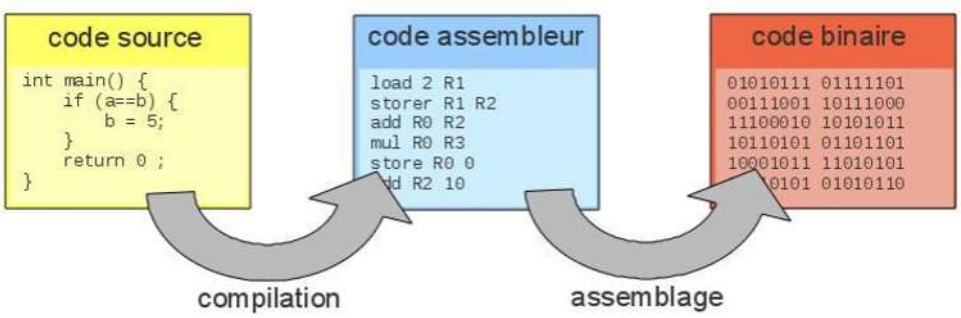
```
x = 3
if x < 0:
    y = -x
else:
    y = x
```

5.4.4 La compilation et l'assemblage

Le problème des langages de haut niveau c'est qu'ils ne sont absolument pas compréhensibles par le processeur ! Avant de pouvoir être utilisé, le code doit être traduit en langage assembleur.

Cette traduction, c'est ce qu'on appelle la compilation.

Une fois en langage assembleur, il faut ensuite l'assembler en langage binaire :



La compilation et l'assemblage