1. Initiation à l'assembleur

Programmer en langage machine est extrêmement difficile (très longue suite de 0 et de 1), 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 à titre d'exemple quelques exemples d'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
```

Additionne la 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 (voir ci-dessous)

```
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

```
HALT
```

Arrête l'exécution du programme

Que permet ?

```
ADD R0, R1, #42
```

ou encore

```
CMP R4, #18
BGT 77
```

Les instructions assembleur B, BEQ, BNE, BGT et BLT n'utilisent pas directement l'adresse mémoire de la prochaine instruction à exécuter, mais des "labels". Un label correspond à une adresse en mémoire vive. L'utilisation d'un label évite donc d'avoir à manipuler des adresses mémoires en binaire ou en hexadécimale :

```
CMP R4, #18
   BGT monLabel
   MOV R0,#14
   HALT
monLabel:
   MOV R0,#18
   HALT
```

2. Comparaison Python vs Assembleur

Programme en python

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

Même chose en assembleur

```
   MOV R0, #4
   STR R0,30
   MOV R0, #8
   STR R0,75
   LDR R0,30
   CMP R0, #10
   BNE else
   MOV R0, #9
   STR R0,75
   B endif
else:
   LDR R0,30
   ADD R0, R0, #1
   STR R0,30
endif:
```

```
   MOV R0, #6
   STR R0,23
   HALT
```

On peut alors demander aux élèves à quoi correspond les adresses mémoires 23, 75 et 30. Pour en savoir plus sur l'architecture de Von Neuman, il est possible de consulter le livre de Claude Timsit [1].

3. Simulateur

Afin de mettre en pratique ce qui vient d'être étudié, on peut faire travailler les élèves sur le simulateur développé par Peter L Higginson [2].

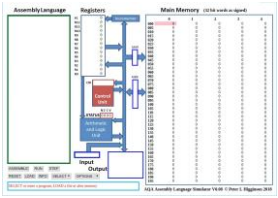

FIGURE 12 – Simulateur de Peter L Higginson

On utilise ce simulateur en le programmant en assembleur (le simulateur accepte l'assembleur étudié précédemment).

Références

[1] Claude Timsit. *Du transistor à l'ordinateur*. 2010.

[2] Peter L Higginson. Simulateur. http://www.peterhigginson.co.uk/AQA/.