
LANGAGE MACHINE

objectifs :

- Savoir dérouler l'exécution d'une séquence d'instructions simples du type langage machine.

1 Langage machine

Un programme est une suite de nombres binaires placés en mémoire représentant des instructions exprimées en langage machine. C'est le seul langage que peut comprendre le processeur chargé d'exécuter ces différentes instructions.

Ainsi, un programme écrit dans un langage de programmation évolué (on dit de haut niveau) comme Python, Java, C, etc. ne peut pas être compris par le processeur. On doit donc utiliser un compilateur (pour le C, par exemple) ou un interpréteur (pour Python, par exemple) pour transformer le programme en langage machine.

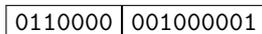
Instruction machine Une instruction machine est un mot binaire composé de deux parties :

- le champ *code opération* qui indique au processeur quelle opération il doit effectuer (charger une donnée en mémoire, faire une addition, une comparaison, etc.)
- le champ *opérandes* qui indique au processeurs la (les) donnée(s) sur laquelle (lesquelles) doit s'appliquer l'opération (l'adresse mémoire de la donnée à charger, les deux valeurs à additionner, les deux valeurs à comparer, etc.)

Une instruction machine possède le schéma suivant :



Par exemple, l'instruction machine pour additionner deux valeurs stockées en mémoire se formule comme :



Langage assembleur Il n'est pas facile (voire impossible) pour un humain d'écrire directement les mots binaires d'un programme en langage machine. On peut utiliser à la place un langage d'assemblage, appelé assembleur, qui est le langage de plus bas niveau d'un ordinateur lisible par un humain. Nous allons détailler un langage d'assemblage pour que tu puisses comprendre l'exécution d'une séquence d'instructions de type langage machine.

Chaque instruction écrite en assembleur possède les deux champs « code opération » et « opérandes ».

Instruction en assembleur	Signification
LDR R1,12	Charge dans le registre R1 la valeur stockée à l'adresse mémoire 12.
STR R3,125	Stocke la valeur du registre R3 en mémoire vive à l'adresse 125.
ADD R1,#128	Additionne le nombre 128 (une valeur immédiate est identifiée grâce au symbole #) et la valeur stockée dans le registre R1, stocke 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, stocke le résultat dans le registre R0.
SUB R1, #128	Soustrait le nombre 128 de la valeur stockée dans le registre R1, stocke 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, stocke 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.
HLT	Arrête l'exécution du programme.

Ainsi le programme Python

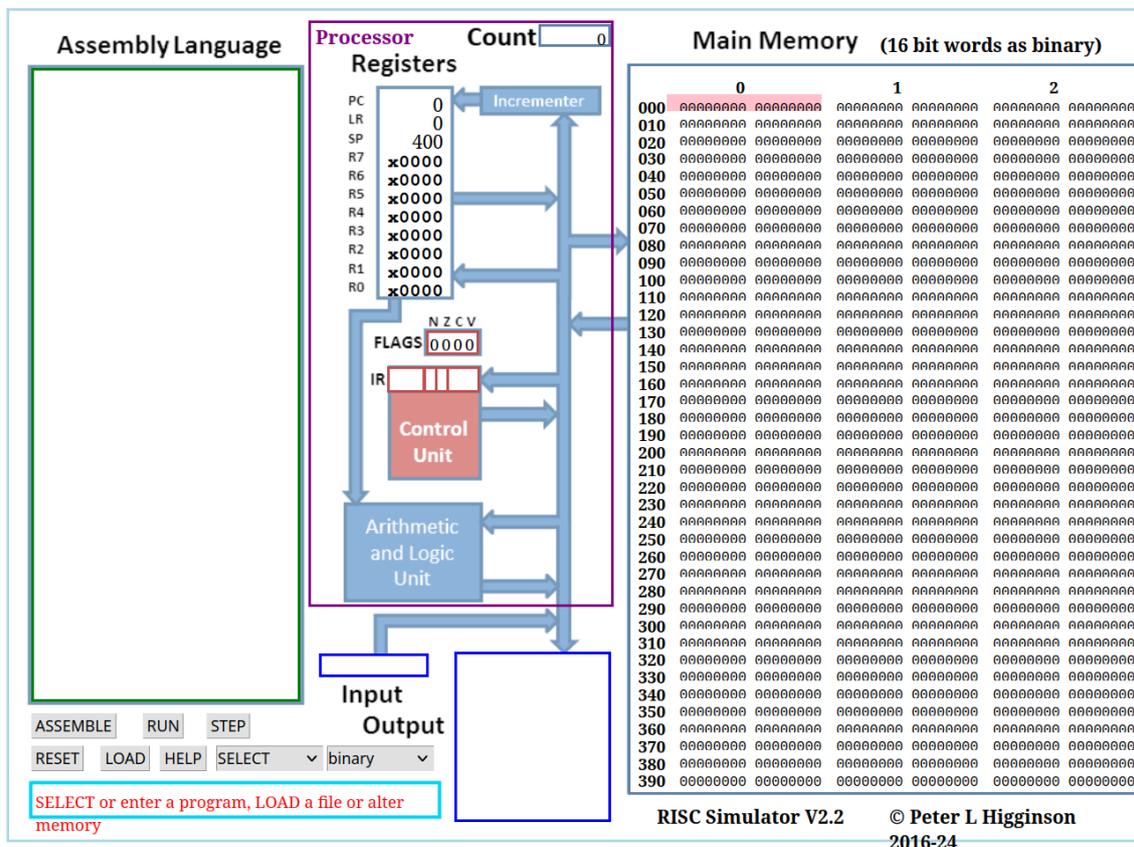
```
1 a = 2
2 b = 7
3 c = a + b
```

pourrait s'écrire en assembleur comme

```
1 MOV R0,#2
2 MOV R1,#7
3 ADD R1,R0,R1
4 STR R1,21
5 HLT
```

2 Simulateur RISC

Dans la suite on utilisera le jeu d'instructions du simulateur RISC développé par Peter Higginson et disponible à cette adresse : <https://peterhigginson.co.uk/RISC/>.



2.1 Comprendre l'exécution d'un programme



Travail à faire

1. Identifier, en les entourant, les 4 parties de l'architecture de von Neumann sur ce simulateur. Entoure également le CPU et localise les registres PC et IR.
2. À l'aide du tableau précédent, traduis par des phrases chacune des instructions du programme en langage d'assemblage suivant :
 - 1 MOV R0,#34
 - 2 STR R0,33
 - 3 HLT

Ouvre le simulateur en ligne et sélectionne « *binary* » dans le menu déroulant « *OPTIONS* » afin d'obtenir une visualisation de la mémoire en binaire (comme c'est le cas en réalité).

- Recopie dans la partie de gauche « *Assembly Language* » le programme ci-dessus et valide en cliquant sur le bouton « *Submit* ». Le programme a été traduit en langage machine et est stocké dans la mémoire à partir de l'adresse 0. Repère et recopie dans ton carnet de bord les mots binaires de ce programme en langage machine.
- Exécute le programme pas à pas en cliquant sur le bouton « *STEP* » à chaque étape en prenant soin d'observer et comprendre ce qu'il se passe. Pour chaque instruction, rédige de manière détaillée ce qu'il se passe.

On considère les instructions en assembleur suivantes :

```

1 MOV R0,#34
2 MOV R1,#5
3 SUB R0,#30
4 ADD R0,R0,R1
5 STR R0,12
6 HLT

```

- Écris un programme en Python qui pourrait être traduit par l'interprète comme cette séquence d'instructions.
- Quels sont les états des registres à la fin du programme et quelle est la valeur stockée dans la case mémoire 12 ?
- Recopie le programme dans le simulateur, lance l'exécution et observe ce qu'il se passe à chaque étape.

2.2 Cas des instructions conditionnelles

Instructions de comparaisons et de saut Il existe d'autres instructions que celles que l'on a vues (pour information, la liste complète est disponible à l'adresse http://www.peterhigginson.co.uk/RISC/instruction_set.pdf). En voici quelques unes importantes concernant les comparaisons et sauts (ruptures de séquence) qui permettent de faire des tests (instructions conditionnelles).

Instruction en assembleur	Signification
BRA 42	Il s'agit d'un saut inconditionnel (BRA pour branch que l'on peut traduire par "bifurcation") : indique que la prochaine instruction à exécuter se situe en mémoire à l'adresse 42.
CMP R0,#23	Compare (CMP pour compare) la valeur stockée dans le registre R0 et le nombre 23. Cette instruction CMP doit précéder une instruction de saut 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.
BEQ	signifie Branch if Equal (bifurcation si les deux opérandes sont égales).
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.
BNE	signifie <i>Branch if Not Equal</i> (bifurcation si les deux opérandes ne sont pas égales).
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 strictement supérieure à 23.
BGT	signifie <i>Branch if Greater Than</i> (bifurcation si la première opérande est strictement supérieure à la deuxième).
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 strictement inférieure à 23.
BLT	signifie <i>Branch if Less Than</i> (bifurcation si la première opérande est strictement inférieure à la deuxième).



Travail à faire

- En t'aidant du tableau ci-dessus, traduis chacune des instructions suivantes :

CMP R2,#100
BNE 36

CMP R1,R2
BGT 36

2. Donnez l'instruction en assembleur correspondant à la phrase suivante :

Si la valeur située dans le registre R3 est strictement inférieure à la valeur 5, l'instruction suivante est située à l'adresse mémoire 40.

Utilisation d'étiquettes En réalité, on va plutôt utiliser des étiquettes (ou label en anglais) avec les opérations BRA, BRA, BEQ, BNE, BGT, BLT, BRA : on remplace l'adresse mémoire qui suit ces opérations par le nom de l'étiquette. On peut alors dénir nous même les instructions de chaque étiquette et donc de chaque partie du programme correspondant à un saut.

Par exemple, voici un programme en assembleur dans lequel on utilise une étiquette appelée Sinon. On a traduit chaque ligne :

```
1      MOV R0,#5 // stocke le nombre 5 dans R0
2      MOV R2,#6 // stocke le nombre 6 dans R2
3      CMP R0,R2 // Compare R0 et R2
4      BNE Sinon // Si R0 != R2, saute à l'étiquette Sinon
5      ADD R0,R0,R2 // R0 <- R0 + R2
6      STR R0,42 // stocke la valeur de R0 en mémoire à l'adresse 42
7      HLT // arrête l'exécution du programme
8 Sinon SUB R0,R0,R2 // R0 <- R0 - R2
9      STR R0,42 // stocke la valeur de R0 en mémoire à l'adresse 42
10     HLT // arrête l'exécution du programme
```



Travail à faire

1. Propose un programme Python pouvant correspondre à ce programme en langage d'assemblage. On nommera **a** et **b** les variables correspondant aux registres R0 et R2. (Indication : il y a une instruction conditionnelle à bien formaliser).
2. Traduis en langage d'assemblage le programme Python suivant :

```
1 a = 3
2 b = 2
3 if a <= 5 :
4     | b = a + b
5 else :
6     | a = a + 3
```