Langage Machine et Assembleur

Mémoire

La mémoire de l'ordinateur peut être vue comme un tableau de cases mémoires élémentaires, appelées **mot mémoire**.

Selon les ordinateurs, la taille de ces mots varie de 8 à 64 bits.

Chaque case possède **une adresse unique** à laquelle on se réfère pour accéder à son contenu.

Cycle d'un processeur

Un **cycle d'horloge** ou **cycle d'exécution** d'une instruction par le processeur est constitué de trois étapes :

- le **chargement** : à l'adresse mémoire indiquée par son **registre IP** (adresse ou trouver la prochaine instruction à exécuter), l'unité de contrôle va récupérer le mot binaire qui contient la prochaine instruction à exécuter et le stocker dans le registre IR (celui contenant l'instruction courante a exécuter) ;

- le **décodage** : le mot binaire dans le **registre IR** est décodé pour savoir quelle instruction doit être exécutée et sur quelles données ; on va aussi charger les données (ou **opérandes**) nécessaires à l'exécution de l'instruction, depuis les registres ou la mémoire.

- l'**exécution** : l'instruction est exécutée, soit par l'ALU (opération arithmétique ou logique), soit par l'unité de contrôle, si il s'agit d'une opération de branchement qui va modifier le registre IP.

Langage Machine

Le processeur est uniquement capable d'interpréter le langage machine, constitué d'**instructions simples en binaire**.

Il comporte un jeu d'instructions spécifiques. Or chaque processeur à son langage, mais ces jeux ont des **structures communes** :

- Chaque instruction contient un code binaire correspondant à l'**opération à effectuer** et aux **opérandes** (données à utiliser) ;

- Les opérations sont : **transfert** de données, entre registres et mémoire par exemple ; **calcul** arithmétique ou logique (addition, comparaison...) ; **branchement** vers une certaine adresse mémoire selon le résultat de l'opération précédente (branchement conditionnel).

Un programme écrit en **langage de haut niveau** (ex : Python, Java, C++...) éloigné du langage machine dit de bas niveau dépend le **moins possible du processeur** et du **système d'exploitation**, car l'on souhaite que les programmes fonctionnent sur toutes les machines.

Entre les langages que l'on connait et que l'on implémente et le langage machine, il existe des langages intermédiaires.

Assembleur

Aujourd'hui plus personne n'écrit de programme directement en langage machine, en revanche l'écriture de programmes en un **langage de bas niveau** est encore chose relativement courante.

Alors on utilise l'**assembleur**, un programme qui permet de passer directement du langage machine à un langage dit d'assemblage plus lisible pour l'être humain :

- le *jeu d'instructions est exactement le même* que celui de la machine ;

- *les opérations sont désignées par des mnémoniques* (des mots qui facilitent la compréhension), comme par exemple **add** pour l'opération de l'addition ;

- les *registres ont des noms* ;

- les *constantes sont exprimables en hexadécimal, en binaire...*

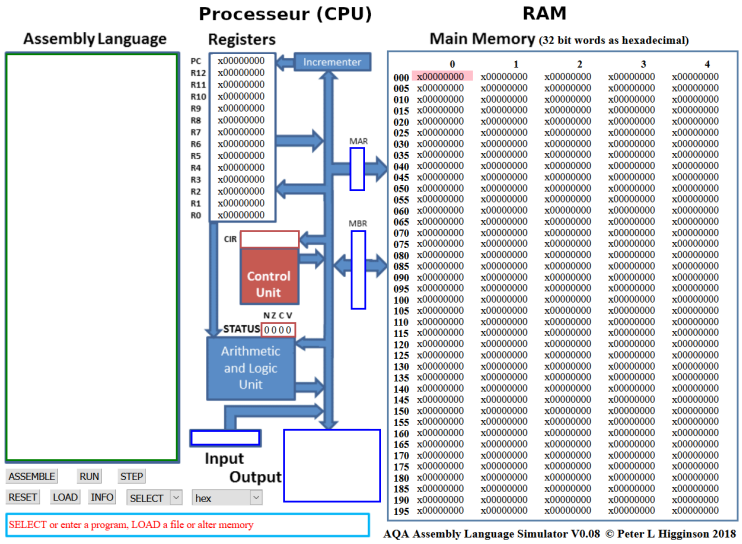
- les *branchements se font via des étiquettes* (labels) : si les contenus sont différents il faut aller à la ligne repérée par l'étiquette.

Par exemple : L'assembleur assurera le passage de l'instruction en langage assembleur "INP R0,2" à la même instruction en langage machine "11100010100000100001000001111101" (sur 32 bits).

Chaque type de processeur possède son langage assembleur, mais les **instructions de bases sont très similaires**. Le but ici n'est pas d'apprendre un langage en particulier, mais de comprendre ces instructions de base.

Pour cela, on va utiliser la syntaxe correspondant à un simulateur que l'on trouve sur internet, le simulateur **AQA**, qui nous permettra de visualiser le fonctionnement de la machine.

http ://www.peterhigginson.co.uk/AQA/



1) Instructions de bases pour AQA

∗ Il y a trois possibilités pour accéder à une donnée (opérande) :
  · elle est directement saisie
  · elle est dans un registre : il faut alors indiquer le nom de ce registre
  · elle est dans la mémoire vive : il faut indiquer son adresse.

∗ On désigne les registres du processeur où sont stockés les données (opérandes) par la notation R0, R1, R2...

∗ Pour désigner un nombre, il faut le précéder du symbole #, sinon, il s'agit d'une adresse de la mémoire vive.

∗ On donne ci-dessous les principales instructions du langage AQA.
(La documentation se trouve à l'adresse **http ://www.peterhigginson.co.uk/AQA/info.html**)

Syntaxe	*Signification*
Déplacements	
MOV R1,R0	(Move) Copie la valeur du registre R0 dans le registre R1
LDR R0,5	(Load) Charge la valeur stockée en mémoire à l'adresse 5 dans le registre R0
STR R1,2	(Store) Range la valeur contenue dans le registre R1 dans la mémoire à l'adresse 2

Opérations	
ADD R2,R1,R0	Additionne les valeurs contenues dans les registres R0 et R1 et stocke le résultat dans R2
SUB R2,R1,R0	(Substract) Soustrait la valeur contenue dans le registre R0 à celle de R1 et stocke le résultat dans R2
CMP R1,R0	(Compare) Compare les valeurs des registres R1 et R0
Branchement après la comparaison **CMP R1,R0**	**si la condition est vérifiée, on se déplace à la ligne indiquée par le label**
BEQ <label>	(Branch Equal) Si les valeurs des registres R0 et R1 sont égales on se déplace
BNE <label>	(Branch Nt Equal) Si les valeurs des registres R0 et R1 sont différentes on se déplace
BGT <label>	(Branch Greater Than) Si la valeur du registre R1 est supérieure à celle de R0 on se déplace
BLT <label>	(Branch Lower Than) Si la valeur du registre R1 est inférieure à celle de R0 on se déplace
Entrées-Sorties	
INP R0,2	(Input) Demande une valeur en entrée et la stocke dans le registre R0
OUT R0,4	(Output) Donne la valeur contenue dans le registre R0 en sortie
HALT	Stoppe le programme

Exemple :

Voici un premier exemple de programme écrit en langage assembleur :

```
INP R0, 2
INP R1, 2
ADD R2, R1, R0
OUT R2, 4
HALT
```

et une description de ce qu'il fait :

- on demande une valeur à l'utilisateur et on la place dans le registre R0 ;
- on recommence, pour placer une valeur dans le registre R1.
- on additionne les valeurs contenues dans les registres R0 et R1, et on place le résultat dans le registre R2.
- on retourne la valeur du registre R2 en sortie.
- le programme s'arrête.

Remarque : Lorsque l'on compare deux valeurs (CMP op1, op2), l'UAL stocke le résultat du calcul (op1 - op2) avec quatre bits spéciaux (NZCV) qui sont mis à jour de la manière suivante :

- N vaut 1 si le résultat est négatif et 0 sinon ;
- Z vaut 1 si le résultat est nul et 0 sinon ;
- C vaut 1 s'il y a une retenue et 0 sinon ;
- V vaut 1 s'il y a un dépassement (overflow) et 0 sinon.

2) Exercices

Exercice 1 : *Les questions sont indépendantes.*

1. Écrire en langage assembleur les instructions suivantes :
   (a) Additionner 18 avec la valeur du registre R_0 et stocker le résultat dans le registre R_1.
   (b) Place la valeur stockée à l'adresse mémoire 12 dans le registre R0
   (c) Place le contenu du registre R0 en mémoire vive à l'adresse 8

(d) Si la valeur stockée dans le registre R0 est égale 42 alors la prochaine instruction à exécuter se situe à la ligne désignée par le label "saut".

(e) Placer la valeur stockée à l'adresse 20 dans le registre R_0, lui soustraire 10, puis stocker le résultat dans le registre R_1.

2. Voici une suite d'instructions :

```
MOV R0, #4
STR R0, 30
MOV R0, #8
STR R0, 31
LDR R0, 31
LDR R1, 30
SUB R2, R1, R0
STR R2, 158
HALT
```

(a) Indiquer la signification de cette suite d'instructions.

(b) Quel est le contenu des registres R_0, R_1 et R_2 à la fin de ces instructions ?

Exercice 2 :

1. On donne une fonction en Python, compléter le code assembleur équivalent :

```
def compare(x,y):
    if (x == y):
        z = x - 4
    else:
        z = x + y
    return z
```

```
        INP ... ,2
        INP ... ,2
        CMP ........
        ... goto
        ............
        HALT
goto:
        ............
        ...
```

2. Tester le code assembleur obtenu sur le simulateur.

Exercice 3 :

Voici un programme Python :

```
x = 0
while x < 3:
    x = x + 1

```

Écrire et tester un programme en assembleur équivalent à ce programme.

Exercice 4 :

Voici un programme en assembleur.

1. Décrire pas à pas son fonctionnement avec les entrées 5 et 3.

2. Vérifier sur le simulateur.
3. Que représente la valeur retournée à la fin ?

```
        INP  R0,2
        STR  R0,25
        INP  R0,2
        STR  R0,26
        MOV  R0,#0
        LDR  R1,25
        LDR  R2,26
        SUB  R2,R2,#1
        MOV  R3,R1
retour:
        ADD  R3,R3,R1
        ADD  R0,R0,#1
        CMP  R0,R2
        BNE  retour
        OUT  R3,4
        HALT
```