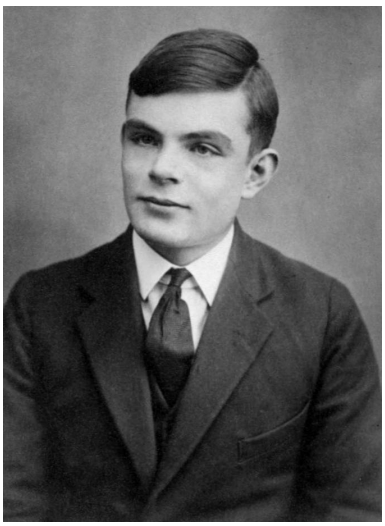
Chapitre 3

Déroulement d'un programme en langage machine

3.1 Un peu d'histoire

3.1.1 L'origine de l'ordinateur : la machine de Turing



Alan Turing est né le 23 juin 1912. C'est un scientifique britanique considéré aurjourd'hui comme le « père de l'informatique ».

Il rédige en 1936 la *Théorie des nombres calculables, suivie d'une application au problème de la décision*. Son idée eqt qu'une machine peut calculer différentes tâches toute seule si on lui indique précisément comment procéder. La **machine de Turing** est née.

C'est une machine abstraite, schématisée par un ruban infini sur lequel se déplace une tête de lecture qui peut lire, écriture se déplacer vers la droite ou la gauche et agir en fonction de ce qui est lu. La tête de lecture possède un nombre fini d'états et réagit en fonction de son état à ce qui est lu. C'est ce qui permet de programmer la machine de Turing.

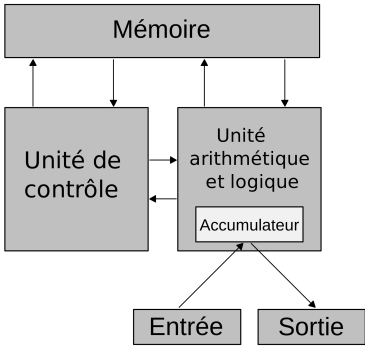
Pour découvrir quelques exemples, on pourra visiter le lien suivant :

`http://inriamecsci.github.io/#!/grains/machine-turing`

3.1.2 L'ordinateur concret : architecture de Von Neumann

On peut considérer que la machine de Turing est l'origine de l'ordinateur tel qu'on le connait aujourd'hui. C'est **John Von Neumann**, mathématicien et physicien américano-hongrois né en 1903, qui, en 1945, va donner concrètement naissance à l'ordinateur que nous connaissons aujourd'hui en décrivant un modèle : l'**architecture de Von Neumann**.

3.2 Architecture de Von Neumann



L'architecture de Von Neumann est un modèle structurel d'ordinateur dans lequel une unité de stockage (mémoire) unique sert à conserver à la fois les instructions et les données demandées ou produites par le calcul.
Les ordinateurs actuels sont tous basés sur des versions améliorées de cette architecture. Une telle architecture décompose l'ordinateur en 4 parties distinctes :

- l'**unité arithmétique et logique** (UAL ou ALU en anglais) ou **unité de traitement** : son rôle est d'effectuer les opérations de base
- l'**unité de contrôle**, chargée du séquençage des opérations
- la **mémoire** qui contient à la fois les données et le programme qui indiquera à l'unité de contrôle quels sont les calculs à faire sur ces données
- les **dispositifs d'entrée-sortie**, qui permettent de communiquer avec le monde extérieur

Les principaux composants de cette architecture, réalisés grâce aux **circuits logiques** que nous avons découvert dans le chapitre précédent, sont les suivants.

Le microprocesseur

Le **microprocesseur** (ou unité centrale de traitement, UCT, en anglais Central Processing Unit, **CPU**) est un composant essentiel qui exécute les instructions machine des programmes informatiques.
Il est constitué de trois parties :

- l'**unité arithmétique et logique** qui permet d'effectuer les calculs
- les **registres**, qui permettent de mémoriser de l'information (donnée ou instruction) au sein même du CPU, en très petite quantité, mais extrêmement rapidement
- l'**unité de contrôle** qui permet d'exécuter les instructions (les programmes)

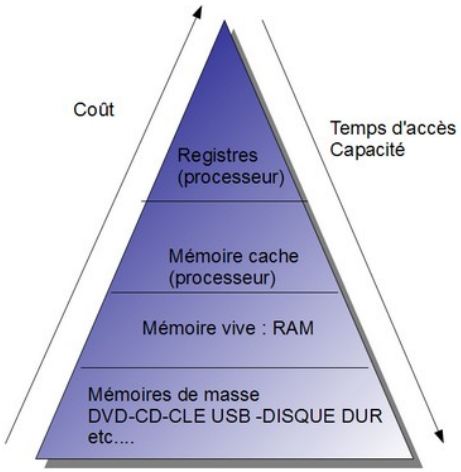
La mémoire

La **mémoire** permet de stocker des données et des programmes.
Les informations y sont classées par **adresses** : chaque octet est accessible par une adresse unique.
Il existe 4 grands types de mémoires :

- la **mémoire de masse** (ou de stockage) sert à stocker à long terme des grandes quantités d'informations.
  - Capacité : jusqu'à 10 To (Disque Dur)
  - Vitesse : jusqu'à 500 Mo/s (SSD)

- la **mémoire vive**, espace principal de stockage du microprocesseur, rapide, mais dont le contenu disparaît lors de la mise hors tension de l'ordinateur.
  - Capacité : jusqu'à 64 Go
  - Vitesse : jusqu'à 2 Go/s

- la **mémoire cache**, intégrée au processeur, qui sert à conserver un court instant des informations fréquemment consultées. C'est un type de mémoire très rapide, mais cher et relativement
  - Capacité : quelques ko (L1) à quelques Mo (L2)
  - Vitesse : jusqu'à 5 Go/s

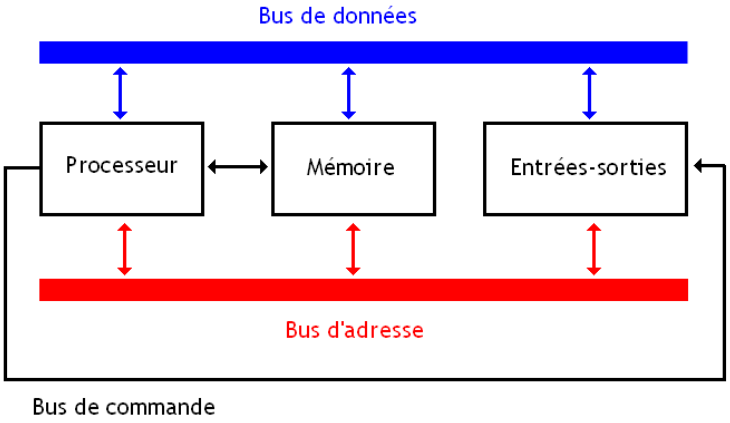


- les **registres**, également intégrés au processeurs. Ce sont les mémoires les plus rapides.
  - Capacité : quelques dizaines d'octets
  - Vitesse : jusqu'à 30 Go/s

Les bus de données

Pour que les données circulent entre les différentes parties d'un ordinateur (mémoire, CPU, entrées/sorties), il existe des systèmes de communication appelés **bus**. Il en existe de 3 grands types :

- le **bus d'adresse** permet de faire circuler des adresses mémoire (pour savoir quelle donnée prendre en mémoire ou pour savoir où écrire une donnée)
- le **bus de données** permet de faire circuler... les données
- le **bus de commande** permet de spécifier le type d'action à opérer (lecture, écriture...)



Une caractéristique importante des bus est leur taille ou leur **largeur**. Elle est définie en nombre de bits.

Par exemple, si un bus d'adresse est de largeur 32 bits, cela signifie qu'il est composé de 32 « lignes » et permet donc de gérer (on dit *adresser*) 2^{32} blocs mémoire, soit environ 4 Go.

Pour un bus de largeur 64 bits, on peut alors adresser 2^{64} blocs mémoires, soit pas loin de 10^{10} Go !

3.3 Langage machine

Le CPU ne gère que des grandeurs booléennes : les instructions exécutées au niveau du CPU sont donc codées en binaire. L'ensemble des instructions exécutables par le microprocesseur constitue ce que l'on appelle le **langage machine**.

Tout langage de programmation « évolué » (Python, C++, ...), destiné à être utilisé par des humains, se compose d'instructions complexes, opérant sur des types de données beaucoup plus complexes que des booléens. Il faudra donc passer par une étape de « conversion » du langage évolué vers le langage machine, chaque instruction du langage « évolué » donnant lieu à un grand nombre d'instructions élémentaires en langage machine.

Une instruction machine est une chaîne binaire composée principalement de 2 parties :

- le champ *code opération* (*opcode*) qui indique au processeur le type de traitement à réaliser.

  Par exemple le code *0010 0110* donne l'ordre au CPU d'effectuer une multiplication.

- le champ *opérandes* qui indique la nature des données sur lesquelles l'opération doit être effectuée.

Les instructions machines sont relativement basiques (on parle d'instructions de bas niveau), voici quelques exemples :

- les **instructions arithmétiques** (addition, soustraction, multiplication...)
- les **instructions de transfert de données** qui permettent de transférer une donnée d'un registre du CPU vers la mémoire vive et vice versa
- les **instructions de rupture de séquence**

Langage Assembleur

Pour additionner les nombres 37 et 29 et stocker le résultat dans un registre de notre processeur (par exemple le registre noté R1), il est donc nécessaire de coder cette instruction sous forme binaire, du style :

$$11100010100000100010000011111010$$

Afin de faciliter la lecture et l'écriture d'instructions machine par les informaticiens, on remplace les codes binaires par des symboles mnémoniques, en utilisant la syntaxe du langage appelé **assembleur**. Notre addition précédente s'écrirait sous la forme suivante, bien plus compréhensible par un humain, avant d'être transcrite en binaire :

$$ADD\ R1,\#37,\#29 \implies 11100010100000100010000011111010$$

Pour manipuler (avec parcimonie) le langage machine, nous pouvons utiliser un simulateur, disponible gratuitement à l'adresse suivante :

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

Le simulateur simule une architecture de Von Neumann de 32 bits.
Le processeur dispose de 13 registres, notés R0 à R12, plus 1 registre PC dit *compteur de programme* (qui s'inscrémente à chaque nouvelle instruction).
On y retrouve notamment l'ALU et l'unité de contrôle, la mémoire et les entrées-sorties.

Voici la liste des principales instructions disponibles :

- **LDR** *Rd, adresse* : charge la valeur de l'adresse dans le registre Rd
- **STR** *Rd, adresse* : place le contenu du registre Rd à l'adresse mémoire
- **ADD** *Rd, Rn, opérande* : additionne opérande et Rn et place le résultat dans Rd. Opérande peut être un nombre décimal (précédé de #) ou un registre
- **SUB** *Rd, Rn, opérande* : soustraction
- **MOV** *Rd, opérande* : copie la valeur de opérande dans Rd
- **CMP** *Op1, Op2* : compare Op1 et Op2. Le résultat est traité par une des commandes suivantes :
  - **BGT** *etiquette* : saute à *étiquette* si Op1 > Op2
  - **BLT** *etiquette* : saute à *étiquette* si Op1 < Op2
  - **BEQ** *etiquette* : saute à *étiquette* si Op1 = Op2
  - **BNE** *etiquette* : saute à *étiquette* si Op1 != Op2
- **INP** *Rd,2* : stocke un nombre donné en entrée dans Rd
- **OUT** *Rd,4* : affiche la valeur de Rd
- **HALT** : termine le programme

Quelques exemples

L'exemple *add* proposé dans le simulateur (ici commenté) est le suivant :

```
1 INP R0,2        # Stocke un nombre donné dans R0
2 INP R1,2        # Stocke un second nombre dans R1
3 ADD R2,R1,R0    # Additionne R1 et R0 et place le résultat dans R2
4 OUT R2,4        # Affiche la valeur de R2
5 HALT            # Fin du programme
```

On peut voir le déroulement pas à pas du programme dans le simulateur.

Le programme *max* affiche le plus grand des deux nombres donnés :

```
INP R0,2
INP R1,2
CMP R1,R0       # Compare R1 et R0
BGT HIGHER      # Si R1 > R0, on saute à l'étiquette HIGHER
OUT R0,4        # Sinon, on affiche R0
B DONE          # On saute à l'étiquette DONE
HIGHER:         # Étiquette HIGHER
OUT R1,4        # On affiche R1
DONE:           # Étiquette DONE
HALT            # Fin du programme
```

Et Python là dedans ?

On peut transcire un programme Python en langage assembleur. Par exemple :

```
x = 2
y = 3
print(x+y)
```

```
MOV R0, #2
MOV R1, #3
ADD R2, R1, R0
OUT R2,4
HALT
```

```
a = 10
b = 25
if a == 13:
  print(b)
else:
  print(a-b)
```

```
MOV R0, #10
MOV R1, #25
CMP R0, #13
BEQ EGALITE
SUB R2, R0, R1
OUT R2,4
B FIN
EGALITE:
OUT R1,4
FIN:
HALT
```

3.4 Exercices

Exercice 01 Qu'affichent les programmes suivants ?

```
MOV R0, #2
MOV R1, #17
MOV R2, #20
ADD R1, R2, R1
ADD R2, R1, R0
OUT R2,4
```

```
MOV R0, #10
MOV R1, #7
ADD R2, R1, R0
MOV R2, R1
OUT R2,4
```

```
MOV R4, #10
CMP R4, #18
BGT label
MOV R0, #14
B fin
label:
MOV R0,#18
fin:
OUT R0,4
HALT
```

Exercice 02 Quel est le rôle du programme suivant ?

```
INP R0,2
INP R1,2
CMP R1,R0
BGT saut
OUT R1,4
B DONE
saut:
OUT R0,4
DONE:
HALT
```

Exercice 03 Transcrire les programmes Python suivant en langage machine :

```
a = 10
b = a + 20
if b > 40:
    print(a)
else:
    print(b)
```

```
x = 1
y = x + 2
if x + y == 10:
    print(x)
else:
    print(y)
```

Exercice 04 Même exercice :

```
a = 0
while a < 5:
    a = a + 2
print(a)
```

```
x = 10
while x > 0:
    x = x - 4
print(x)
```

Exercice 05 Écrire un programme en langage machine qui demande un nombre et affiche le produit de ce nombre par 3.

Exercice 06 Écrire un programme en langage machine qui affiche le produit de deux nombres entiers positifs donnés.

Exercice 07 Écrire un programme en langage machine qui demande un nombre entier positif et affiche son carré.