Architecture matérielle

Un petit questionnaire vous attend en fin de travail sur l'architecture.

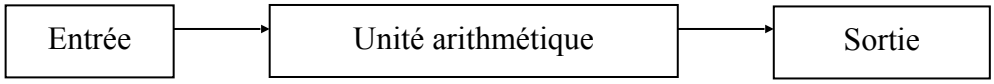
I. Genèse et histoire

1) Introduction

Entre le XIe et le XVIIe siècle, l'ordinateur désigne celui qui est chargé de « régler les affaires publiques ». La définition a énormément évolué au fil du temps et surtout ces derniers 60 ans.
Celle de Wikipedia est aboutie : « machine électronique qui fonctionne par la lecture séquentielle d'un ensemble d'instructions qui lui font exécuter des opérations logiques et arithmétiques sur des chiffres binaires ». Nous avons déjà vu l'aspect « lecture séquentielle d'instructions », en écrivant des programmes, et l'aspect binaire dans le cours sur le codage. Nous allons dans ce chapitre voir comment l'architecture permet d'effectuer les « opérations arithmétiques et logiques ».

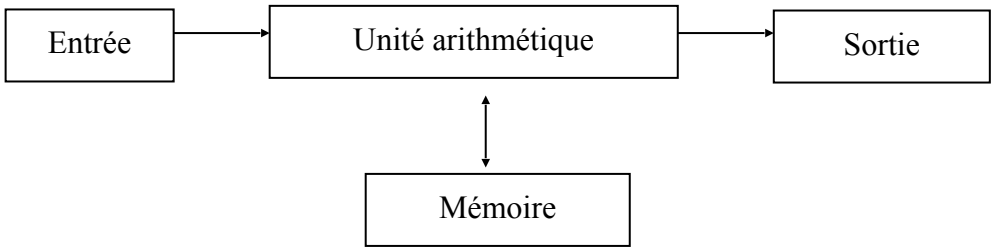
2) Un peu d'histoire

Les premières machines à calculer datent du XVIIe siècle. La pascaline est la plus connue, c'est en fait la seule dont on soit sûr de l'existence puisqu'il en existe toujours une dizaine. Elle ne pouvait faire que des additions, mais Blaise Pascal avait trouvé la méthode du complément à 10 (sur le principe du complément à 2 vu dans le dernier chapitre) pour faire des soustractions.
On a le schéma suivant :



Leibniz, en 1673, rajoute la mémorisation des résultats intermédiaires : sa machine pouvait faire ainsi des multiplications et des divisions.

D'où le schéma :



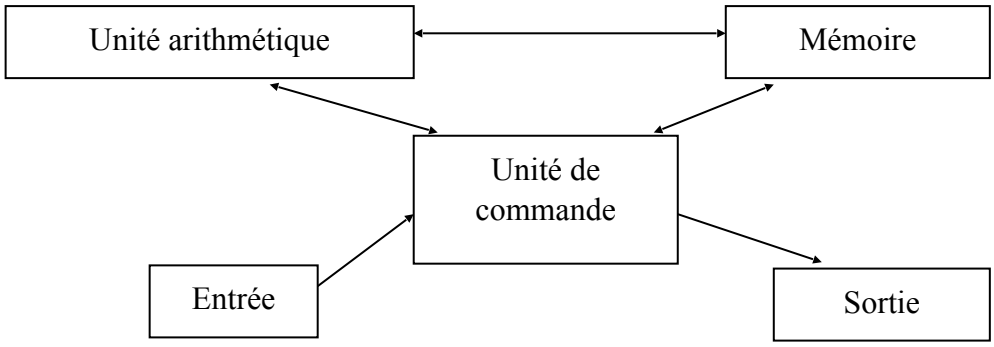
Leibniz évoque de plus la représentation binaire grâce… à la Chine. En effet, il connaît le « Yi-King », traité de divination, dont les tirages divinatoires se font en fait en binaire. Il communique sur le binaire à l'Académie des Sciences en 1703.

150 ans plus tard, en 1847, Georges Boole fonde l'algèbre de Boole, que l'on verra très prochainement. Cette algèbre est basée sur les valeurs de vérité Vrai ou Faux, et permet notamment de fabriquer des circuits électroniques compacts.

À peine plus tôt, en 1834, **Charles Babbage** à l'idée d'incorporer dans la machine à calculer des cartes perforées, pour donner la suite d'instructions à exécuter. Il est à noter que cette idée datant de 1834 n'a pas être réalisée matériellement qu'à la fin des années 1890 et est à la base de la notion d'entrée/sortie des ordinateurs actuels.

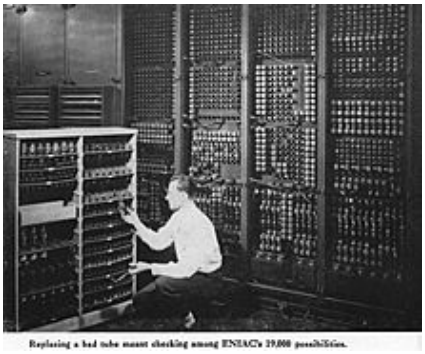
Babbage ne put jamais réaliser sa machine la plus performante, faute de fonds suffisants. Ada Lovelace, qui travailla avec Babbage, écrivit le premier véritable programme informatique de l'histoire. Son nom a été donné à un langage de programmation, et son portrait figure sur les hologrammes d'authentification des produits Microsoft.

Depuis cette époque, le schéma qui va servir de référence est celui-ci :



Le premier calculateur utilisant l'électricité et le binaire fut le Z1, conçu en 1936 par l'ingénieur allemand Konrad Zuse, qui crée aussi le premier langage de haut niveau. Ce langage fut oublié, faute de machine capable de le supporter, jusqu'en 1972. Il n'a été implémenté qu'en 2000, à titre historique. Quant à l'ordinateur, dont la troisième version, le Z3, fut détruite par les alliés à la fin de la guerre, il semblerait qu'il était bien plus avancé que ses concurrents directs, dont l'ENIAC (cf. ci-dessous).

C'est Claude Shannon, spécialiste de la théorie de l'information, qui expose en 1938 comment utiliser l'algèbre de Boole pour construire des machines à calculer basées sur des commutateurs et des relais.



Le premier ordinateur « moderne », l'ENIAC, est mis en service en 1946 sur une base de l'architecture de Von Neumann. Einstein et Gödel (vous connaissez le premier, le deuxième est un génie de la logique) estimaient que cette coûteuse réalisation n'apporterait aucune contribution à la science… Cet ordinateur de première génération (1946-1956) fonctionnait avec des lampes à vide (toujours utilisées dans les amplificateurs audio de grande qualité), de durée de vie très limitée. La durée moyenne entre deux pannes était de quelques heures. Une fausse origine du terme « bug » est l'une des causes de pannes : un insecte (bug) qui, se posant sur un tube, brûle et cause la rupture du tube. En fait ce terme est bien plus ancien, il désigne depuis le XIXe siècle les disfonctionnements dans des éléments mécaniques.

En 1947 apparaît le transistor, petit, fiable, et peu coûteux. Il remplace les lampes à vide dans les ordinateurs de deuxième génération (1956-1963).

Enfin les circuits intégrés formés de quelques dizaines à plusieurs centaines de millions de transistors voient le jour en 1958. Ils sont au cœur des ordinateurs de troisième génération (1963-1971).

Nous sommes dans la quatrième génération d'ordinateurs, celle qui utilise des microprocesseurs, qui ont permis la naissance des micro-ordinateurs. Selon certaines sources, le premier micro-ordinateur, serait français : le Micral, crée en 1972.

Selon la loi de Moore, la densité des composants électroniques sur une puce suit une croissance exponentielle (suite géométrique), en doublant

tous les deux ans. Le premier microprocesseur, un Intel était une unité de calcul sur 4 bits, tournant à 108 kHz et intégrant 2300 transistors. Un Intel actuel (un des i7) comporte 3,2 milliards de transistors, tourne à 3,7 Ghz, et calcule sur 64 bits. La loi de Moore s'est vérifiée jusqu'en 2010, mais la miniaturisation devient telle que des limites physiques infranchissables se dressent.
Les recherches se portent actuellement sur le long terme, avec des ordinateurs quantiques, où un bit peut superposer en même temps les valeurs 0 et 1…

3) L'architecture de la Machine de Von Neumann

L'architecture générale des ordinateurs a été fournie par John Von Neumann en 1945. Cette architecture est identique du nano-processeur qui équipe la machine à laver jusqu'au superordinateur.
On retiendra les principes suivants :

- La fonction de calcul est réalisée par *l'UAL* (*unité arithmétique et logique*) ;
- La fonction d'enregistrement est réalisée par la *mémoire* ;
- Le déroulement séquentiel est réalisé par *l'unité de commande*.
- Les *registres* stockent temporairement des données.

Pour simplifier un peu la situation, nous dirons que l'UAL et l'unité de commande sont regroupés dans le processeur, qui communique avec la mémoire par un bus.

Ces composants constituent l'unité centrale, située sur la carte mère de votre pc. L'ensemble est rythmé par une horloge, c'est la fréquence du microprocesseur. Exemple : 3,4 gHz, soit 3 milliards 400 000 000 cycles d'horloge par seconde pour un Core i7-2600 K.

II. Architecture

1) La mémoire

L'idée de la mémoire est simple : c'est un ensemble de cellules possédant deux caractéristiques.
Chaque cellule possède :

- Une adresse, qui est l'indice de la cellule dans le tableau : X
- Un contenu, qui est la valeur de l'élément dans le tableau, stocké sur un **octet** : M[X]

Les cellules servent à la fois à stocker les données (ou variables), et le programme.

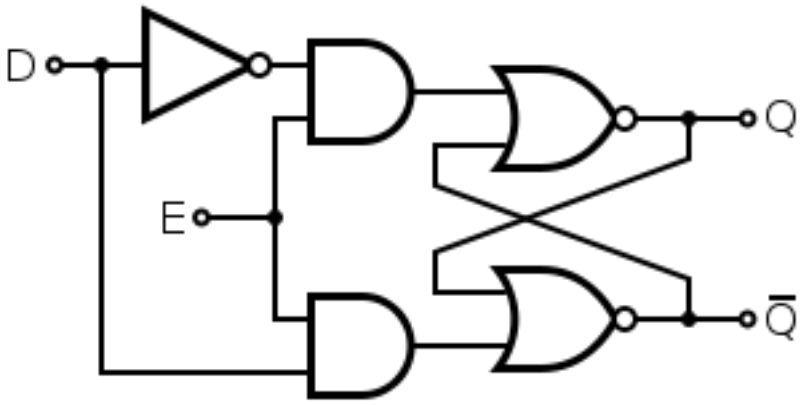
Adresse		Contenu (exemple)
Décimal	Binaire	
0	0	00101100
1	1	11001011
2	10	…
3	11	…

Globalement, on distingue plusieurs types de mémoire :

- Interne vive : la RAM, les barrettes que vous pouvez rajouter
- Interne morte : la ROM, indispensable pour le démarrage de l'ordinateur
- Externe (de masse) : disques durs, cartes flash des appareils photo (qui ne sont réinscriptibles que 100 000 fois au plus, ce sont un type particulier de ROM) , clefs USB, etc…

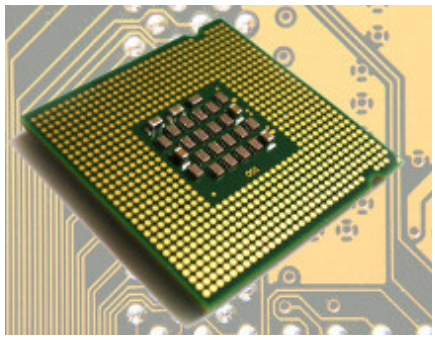
Technologiquement, un bit d'une cellule est l'association d'un transistor et d'un condensateur. Un condensateur est un composant électronique qui peut être soit chargé (on stocke alors un "1"), soit déchargé (on stocke alors un "0"). Le problème est qu'un condensateur n'est pas capable de conserver sa charge pendant très longtemps. Il doit donc être alimenté électriquement afin de conserver cette charge. Voilà pourquoi la mémoire vive est une mémoire volatile : toutes les données présentes en mémoire sont perdues en cas de coupure de courant. Pour conserver les données une fois l'ordinateur éteint, il faut faire appel à d'autres types de mémoire : les mémoires de stockage. Le disque dur est aujourd'hui la mémoire

de stockage la plus utilisé (au moins dans les usages "familiaux"). Un disque dur n'a pas besoin d'alimentation électrique pour conserver les données.
Pour terminer sur cet aspect technologique, il faut noter que l'on trouve aussi des mémoires vives qui stockent l'information grâce à un circuit dit de type "bascule". Ce circuit est une combinaison de plusieurs **portes logiques booléennes** qui "retiennent" le bit voulu.



Il n'est pas question pour nous d'étudier ce type de circuit. Nous n'étudierons que la base de la logique booléenne.

2) Le microprocesseur CPU (Central Processing Unit)



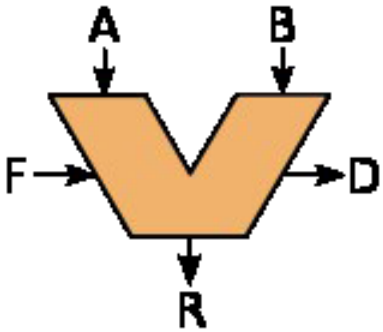
Le microprocesseur est le "coeur" d'un ordinateur : les instructions sont exécutées au niveau du CPU.

Il est schématiquement constitué de 3 parties :

- les **registres ou banc de registres** permettent de mémoriser de l'information (donnée ou instruction) au sein même du CPU. Leur nombre et leur taille sont variables en fonction du type de microprocesseur. Dans la suite on nommera ces registres R1, R2, R3...

- **L'unité de commande** (ou controle) permet d'exécuter les instructions (les programmes) : il contient en particulier un registre compteur de programme (cp) et un registre d'instruction (ri).

- **L'unité arithmétique et logique** (UAL ou ALU en anglais) est chargée de l'exécution de tous les calculs que peut réaliser le microprocesseur. Nous allons retrouver dans cette UAL des circuits comme l'additionneur.



Nous allons discuter un peu plus longtemps de l'UAL qui est le cœur de l'ordinateur.
On la représente habituellement par ce schéma :

- A et B sont les opérandes (les entrées)
- R est le résultat
- F est une fonction binaire (l'opération à effectuer)
- D est un drapeau indiquant un résultat secondaire de la fonction (signe, erreur, division par zéro, « supérieur »,…)

Les opérations que peuvent réaliser une UAL de base sont :

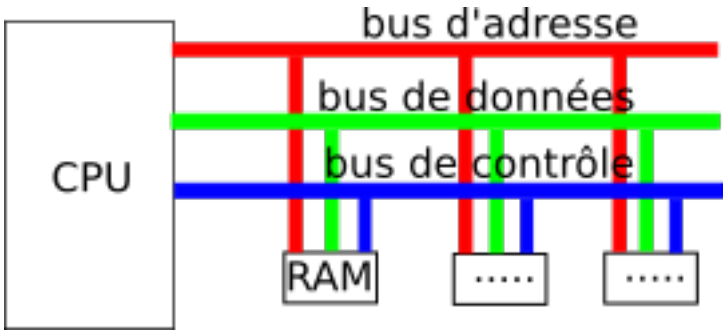
- Les opérations arithmétiques usuelles (+, - , ×, ÷) ;
- Les opérations logiques (ET, OU, NON) ;
- Les opérations de comparaison (> , < , =)

Certaines UAL sont spécialisées (calcul sur les nombres en virgule flottante, cartes graphiques, cartes sons, …). Un même processeur peut comporter plusieurs UAL. Dans certaines architectures, les UAL sont spécialisées et ne fonctionnent pas simultanément, dans d'autres les UAL fonctionnent en parallèle (architectures multi-cœur). Les architectures multi-cœur permettent d'augmenter la puissance de calcul sans se heurter aux problèmes de miniaturisation (cf. limites sur la loi de Moore).
Toues les fonctions réalisées par les UAL le sont grâce à des circuits électroniques effectuant des **fonctions binaires :** nous en verrons quelques exemples dans la partie sur le calcul booléen.

3) Le bus

Finalement, les données doivent circuler entre les différentes parties d'un ordinateur, notamment entre la mémoire vive et le CPU. Le système permettant cette circulation est appelé bus. Il existe, sans entrer dans les détails, 3 grands types de bus :

- Le bus d'adresse permet de faire circuler des adresses (par exemple l'adresse d'une donnée à aller chercher en mémoire)
- Le bus de données permet de faire circuler des données
- Le bus de contrôle permet de spécifier le type d'action (exemples : écriture d'une donnée en mémoire, lecture d'une donnée en mémoire).



III. Langage de base niveau : Assembleur

Pour faire fonctionner tout cela, nous avons besoin de langage de bas niveau. Les langages de programmation les plus proches du codage de « bas niveau », en 0 et 1, sont appelés « Assembleurs ». Un langage assembleur dépend d'un microprocesseur. Un programme écrit en Assembleur ne tournera que sur **une seule machine**, contrairement à un programme écrit dans un langage de haut niveau.

En Assembleur, dans sa forme la plus simple, une instruction est stockée dans **une cellule de mémoire désignée par une adresse**.

Elle est constituée de **deux parties** codée en binaire:

- Un code opération (CO) qui indique quelle opération doit être effectuée
- Une adresse (AD) désignant l'opérande de cette opération

Exemple:
un algorithme de calcul d'une addition décrit par l'expression arithmétique z = x + y donnera naissance à une suite de 3 instructions telles que :

- LDA X Chargement (load) du contenu de la cellule mémoire X dans un registre appelée accumulateur. On note ce registre A ici.
- ADD Y Addition du contenu de la cellule mémoire Y avec le contenu de l'accumulateur
- STO Z Stockage (sto pour… store) du contenu de l'accumulateur dans la cellule mémoire Z

LDA, ADD, STO étant des codes opération ; X, Y, Z les adresses des opérandes.

En binaire, si on imagine que LDA correspond à l'instruction 001, ADD à 010 et STO à 011, et que l'accumulateur a pour adresse 00100, la cellule Y a pour adresse 00101 et la cellule Z 01000.

La suite d'instructions se traduit alors en binaire par :

- **001**00100 : LDA X
- **010**00101 : ADD Y
- **011**01000 : STO Z

Toute expression arithmétique ou logique, quelle que soit sa complexité, va être décomposée en une suite d'instructions de cette nature. L'unité de commande va alors se charger d'enchaîner séquentiellement la suite de ces instructions calculant cette expression.

Cependant il est bien évident que ce seul mécanisme est insuffisant et ne peut réaliser les instructions itératives ou conditionnelles des langages de haut niveau. Pour cela, il est nécessaire d'introduire des instructions spécifiques appelées des **ruptures de séquence**.

Une instruction de rupture de séquence simple est de la forme :

- JMP X Saut Inconditionnel (jump) indique que le programme se poursuit à partir de l'adresse X

Une instruction de rupture de séquence conditionnelle est de la forme :

- JMPc X Saut Si c indique que le programme se poursuit à partir de l'adresse X si la condition c est réalisée.

Si la mémoire d'adresse X contient 10, MUN est l'adresse d'une cellule contenant la valeur -1, et « + » signifie le résultat est strictement positif, que va faire le programme suivant ?

```
        LDA   X
ADR     ADD   MUN
        JMP+  ADR
        STO   X
```

Remarque : il est particulièrement simple de tester si un entier est positif. Comme nous l'avons vu dans le chapitre précédent sur le codage, son premier bit vaut 0. Les opérations réalisables par les premières UAL étaient des opérations particulièrement simples. Rappelez-vous comment on fait pour multiplier par 2/ diviser par 2 par exemple…

Exercices architecture

Les exercices qui suivent s'appuient sur une structure ultra simplifiée d'un ordinateur totalement imaginaire dont voici les caractéristiques :

La mémoire centrale est constituée d'un tableau de 32 cellules
Chaque cellule est constituée d'un octet
Chaque instruction est constituée
- ❖ d'un code opération sur 3 bits
- ❖ d'une partie adresse sur 5 bits

Le jeu d'instructions est constitué des 8 instructions suivantes :

Instruction	Nom	Effet (version simple)	Effet (version moins simple)	Code opération
LD X	Charger	Charge dans l'accumulateur (l'entrée A de l'UAL) le contenu situé à l'adresse X	ACC ← M [X]	000
STO X	Stocker	Stocke à l'adresse X le contenu de l'accumulateur (le résultat du calcul précédent)	M [X] ← ACC	001
ADD X	Additionner	Ajoute à l'accumulateur (entrée A de l'UAL) le contenu situé à l'adresse X	ACC ← ACC+M [X]	010
SUB X	Soustraire	Soustrait à l'accumulateur (entrée A de l'UAL) le contenu situé à l'adresse X	ACC ← ACC-M [X]	011
JMP ADR	Saut	Saut à l'adresse ADR	CO ← A	100
JMPZ ADR	Saut si = 0	Idem, si résultat du calcul à la ligne d'avant (stocké dans l'accumulateur) est égal à 0	si CC = 0 alors CO ← A	101
JMPP ADR	Saut si > 0	Idem, si résultat du calcul dans l'accumulateur est supérieur à 0	si CC > 0 alors CO ← A	110
JMPN ADR	Saut si < 0	Idem, si résultat du calcul est inférieur à 0	si CC < 0 alors CO ← A	111

On considérera qu'un programme commence à l'adresse 8 (1000 en binaire), et que les adresses 0 à 7 seront utilisées pour stocker les données. On notera END pour la fin de programme

Ex 1.
Que font les programmes ci-dessous ? Version facile en code, version pénible en binaire.

Adresse	Contenu
0	41
1	54
…	
8	LD 0
9	ADD 1
10	STO 2

Adresse	Contenu
0	00100011
1	01000011
…	
1000	00000001
1001	01100000
1010	00100010

Ex 2.
Écrire un programme vérifiant les conditions suivantes :

- Soit *x* une valeur écrite dans la case mémoire 0.
- Lire le nombre rangé dans la case mémoire 1.
- Si ce nombre est plus grand ou égal à *x*, remplacer le contenu de la case 3 par ce nombre.

Ex 3.
Même exercice que le précédent, mais maintenant, on écrit dans la case mémoire 3 :

- 0 si le contenu de la case 1 est strictement inférieur à celui de la case mémoire 2
- 1 sinon.

Ex 4 : Écrire un programme qui effectue la multiplication de deux entiers positifs ou nuls.

Ex 5 : Écrire un programme qui effectue la division euclidienne d'un entier *a* par un entier *b*. On stockera le quotient et le reste.