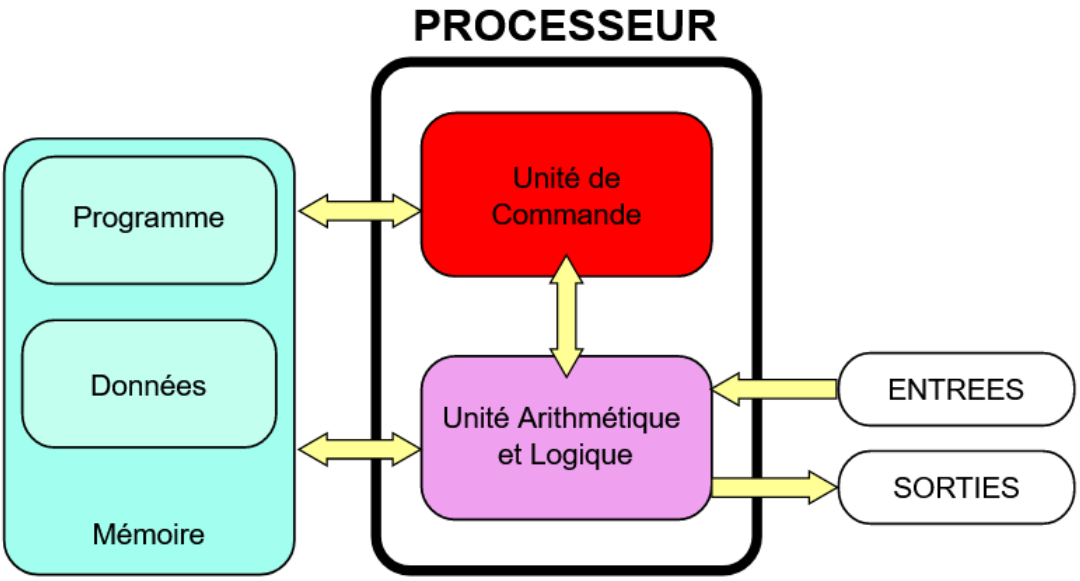
Chapitre 1 : Architectures matérielles

1. Le modèle de Neumann

1.1. Présentation

Le premier ordinateur, L'ENIAC (*Electronic Numerical Integrator And Computer*), entièrement électronique conçu sur le modèle de la machine de Turing[1] , est réalisé en 1945. Son architecture, est décrite en 1945 par le mathématicien John von Neumann et donna naissance au modèle de Neumann :



L'architecture de von Neumann est un modèle pour un ordinateur qui utilise une structure de stockage unique pour conserver à la fois les instructions et les données demandées ou produites par le calcul [Wikipedia.org]

Cette architecture est centrée autour de deux composants principaux :

- Le **processeur** ou CPU *(Central Processing Unit)* ;
- La **mémoire**.

→ Ces deux composants sont reliés par des fils (circuits électriques) appelés *bus de communication* ; le processeur dispose de *bus d'entrée* et *de sortie* qui le relient à d'autres parties de l'ordinateur appelées les *périphériques* comme le disque dur, l'écran, le clavier...

PRINCIPALES CARACTÉRISTIQUES D'UN PROCESSEUR :

- Le jeu d'instructions : permet d'additionner, de multiplier, de comparer deux nombres... Un processeur peut exécuter plusieurs dizaines, voire centaines ou milliers, d'instructions différentes par seconde ;
- La complexité : plus le microprocesseur contient de transistors, plus il pourra effectuer des opérations complexes ;
- Le nombre de bits pouvant être traités simultanément : les premiers microprocesseurs ne pouvaient pas traiter simultanément plus de 4 bits. À partir de 2007 les microprocesseurs peuvent traiter des nombres sur 64 bits. Le nombre de bits des bus, de la mémoire et du processeur permet de manipuler rapidement des grands nombres, ou des nombres d'une grande précision ;
- La vitesse de l'horloge : la fréquence d'exécution du processeur est contrôlée par un signal d'horloge, plus la vitesse de l'horloge est grande, plus le microprocesseur effectue d'instructions en une seconde ; les cadences des processeurs usuels sont de l'ordre du GHz à l'heure actuelle.

→ La combinaison des caractéristiques précédentes détermine la puissance du microprocesseur qui s'exprime en « millions d'instructions par seconde » (MIPS).

[1] Modèle abstrait du fonctionnement des appareils mécaniques de calcul, comme un ordinateur, pouvant être programmés.

1.2. L'ordinateur personnel

On entend souvent dire qu'un ordinateur « utilise uniquement des 0 et des 1 ».

À la base de la plupart des composants d'un ordinateur, on trouve des transistors (ci-contre) qui sont regroupés au sein de ce que l'on appelle des circuits intégrés, qui sont gravés sur des plaques de silicium. Les connexions entre ces millions de transistors qui composent un circuit intégré sont, elles aussi, gravées directement dans le silicium.

Les transistors (inventé en 1947) se comportent comme des interrupteurs : soit ils laissent passer le courant électrique (comme un interrupteur fermé), soit ils ne le laissent pas passer (comme un interrupteur ouvert). Ainsi, l'ordinateur fonctionne uniquement avec deux états : un état « haut » et un état « bas » que l'on symbolise souvent par le chiffre « 1 » (état haut) et le chiffre « 0 » (état bas). C'est donc juste une histoire de « courant qui passe » (état « haut » ou « 1 ») ou de « courant qui ne passe pas » (état « bas » ou « 0 »). Un ordinateur réalise donc des opérations avec uniquement « 2 chiffres », voilà pourquoi on dit qu'il travaille en base 2 (en binaire) et non pas en base 10 comme dans la vie courante.

Le transistor est l'élément de base des circuits logiques qui permettent de réaliser des opérations booléennes qui sont directement liées à l'algèbre de Boole (Georges Boole, mathématicien Britannique 1815-1864) : un circuit logique prend en entrée un ou des signaux électriques (chaque entrée est dans un état « haut » (symbolisé par un « 1 ») ou à un état « bas » (symbolisé par un « 0 »)) et donne en sortie un ou des signaux électriques (chaque sortie est aussi dans un état « haut » ou à un état « bas »).

Il existe deux catégories de circuit logique :

- Les circuits combinatoires (les états en sortie dépendent uniquement des états en entrée) ;
- Les circuits séquentiels (les états en sortie dépendent des états en entrée ainsi que du temps et des états antérieurs). [Hors programme]

QUELQUES EXEMPLES DE CIRCUITS COMBINATOIRES :

❖ LA PORTE « NON » :

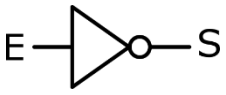
Le plus simple des circuits combinatoires est la porte « **NON** » (« NOT » en anglais) qui inverse l'état en entrée : si l'entrée de la porte est dans un état « bas » alors la sortie sera dans un état « haut » et vice versa.

Si on symbolise l'état « haut » par un « 1 » et l'état « bas » pour un « 0 », on peut obtenir ce que l'on appelle la table de vérité de la porte « NON » :

Entrée	Sortie
E	S (NON E)
1	0
0	1

Table de vérité « NON »

La porte « NON » est symbolisée par le schéma suivant :



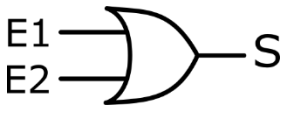
❖ LA PORTE « OU » :

La porte « **OU** » ou « **OU inclusif** » (« OR » en anglais) associe à la sortie un résultat qui a la valeur « 1 » seulement si au moins une des deux entrées a la valeur « 1 ».

Entrée		Sortie
E1	E2	S (E1 OU E2)
1	0	1
1	1	1
0	0	0
0	1	1

Table de vérité « OU »

La porte « OU », a deux entrées (E1 et E2) et une sortie (S), est symbolisée par le schéma suivant :



❖ **LA PORTE « ET » :**

La porte « **ET** » (« AND » en anglais) associe à la sortie un résultat qui a la valeur « 1 » seulement si les deux entrées ont la valeur « 1 ».

Entrée		Sortie
E1	E2	S (E1 ET E2)
1	0	0
1	1	1
0	0	0
0	1	0

Table de vérité « OU »

La porte « ET », a deux entrées (E1 et E2) et une sortie (S), est symbolisée par le schéma suivant :

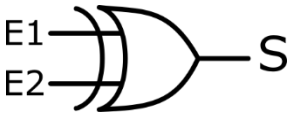
❖ **LA PORTE « OU EXCLUSIF » :**

La porte « **OU Exclusif** » (« XOR » en anglais) associe à la sortie un résultat qui a la valeur « 1 » seulement si les deux entrées ont des valeurs distinctes (= différentes).

Entrée		Sortie
E1	E2	S (E1 XOR E2)
1	0	1
1	1	0
0	0	0
0	1	1

Table de vérité « OU EXCLUSIF »

La porte « OU EXCLUSIF », a deux entrées (E1 et E2) et une sortie (S), est symbolisée par le schéma suivant :

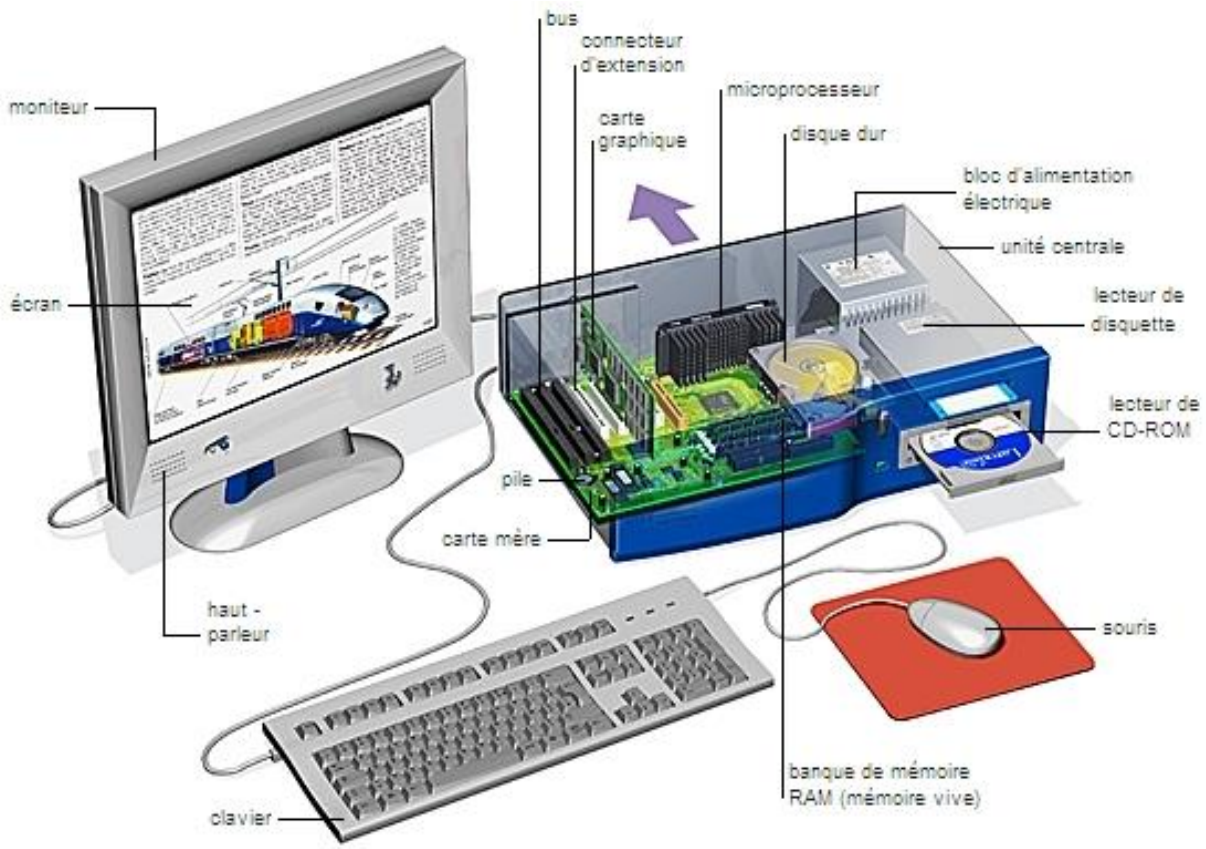


→ En combinant les portes logiques, on obtient des circuits plus complexes.

Pour aller plus loin :

- Fonction logique (Wikipedia.org)

1.2.1. Composants et périphériques

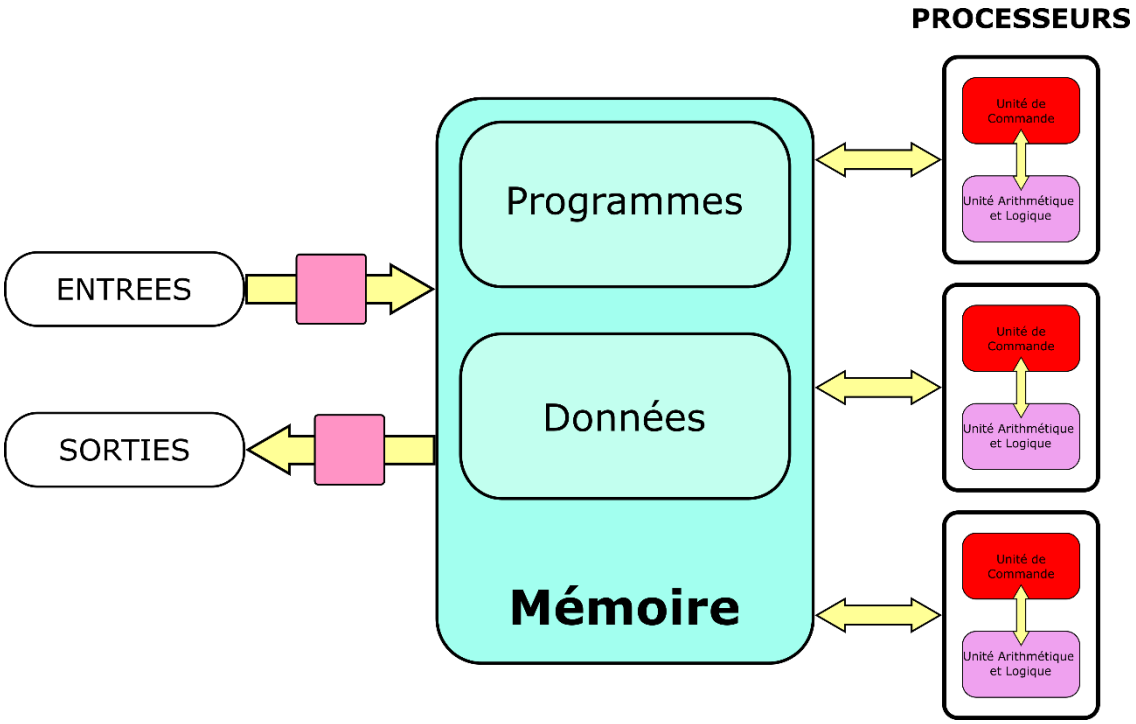


Le **microprocesseur** (ou **processeur**) est le « cœur » d'un ordinateur : les instructions sont exécutées au niveau du CPU (*Central Processing Unit*). Il est schématiquement constitué de 3 parties :

- Les *registres*, qui permettent de mémoriser (stocker) de l'information (donnée ou instruction) au sein même du CPU. Leur nombre et leur taille sont variables en fonction du type de microprocesseur ;
- *L'unité arithmétique et logique* (UAL ou ALU en anglais), c'est un circuit numérique combinatoire qui est chargée de l'exécution de tous les calculs que peut réaliser le microprocesseur ;
- *L'unité de commande*, elle dirige le fonctionnement du processeur et permet d'exécuter les instructions (les programmes). Elle indique à la mémoire de l'ordinateur, à l'unité arithmétique et logique et aux périphériques d'entrée et de sortie comment répondre aux instructions qui ont été envoyées au processeur.

→ Lorsque l'opération est terminée, l'unité de commande passe à l'instruction suivante du programme.

Remarque : les ordinateurs multiprocesseurs sont capables d'effectuer plusieurs tâches en parallèle, pour obtenir une plus grande puissance de calcul.



La **mémoire vive** ou **RAM** (*Random Access Memory*) est un espace de stockage temporaire des données traitées par le microprocesseur. Elle est volatile : son contenu disparait lorsqu'elle n'est plus alimentée en électricité.

La **mémoire morte** ou **ROM** (*Read-Only Memory*) est un espace de stockage (mémoire) non volatil, de faible capacité, dont le contenu est fixé lors de sa programmation. Ce contenu peut être lu mais n'est pas prévu pour être modifié. C'est l'exemple du BIOS (Basic Input Output System), qui contient des instructions permettant d'effectuer des opérations de base, lors de la mise sous tension de la machine.

La **mémoire de masse** est l'autre nom donnée à la mémoire non volatile sur laquelle des données peuvent être lues et écrites. C'est l'exemple des disques durs, des SSD, des clés USB, des carte SD/ microSD, etc. Elle offre de grandes capacités de stockage.

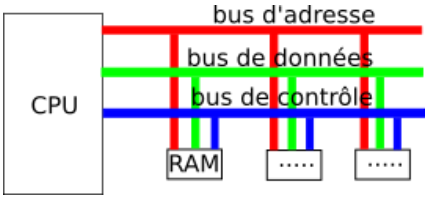
La **mémoire cache** est un espace de stockage (mémoire) volatil permettant d'accélérer l'accès aux données contenues dans la mémoire vive ou la mémoire de masse. Elle contient les informations fréquemment utilisées par le microprocesseur ou le contrôleur (de la mémoire de masse).

→ La mémoire ne gère pas les bits 1 par 1, mais 8 par 8, la mémoire gère donc des octets (1 octet = 8 bits) : on peut la schématiser comme une série de cellules, chaque cellule étant capable de stocker 1 octet et chacune de ces cellules possède une adresse.

→ Les opérations sur la mémoire sont de 2 types : lecture et écriture. Une opération de lecture consiste à aller lire l'octet situé à l'adresse mémoire XXXXX (codées en binaire) et une opération d'écriture consiste à écrire un octet donné à l'adresse mémoire YYYYY (codée en binaire).

La **carte mère** est le circuit imprimé sur lequel sont rassemblés la plupart des composants électroniques et les ports de connexion (**connecteurs**) permettant d'assurer la liaison entre les différents éléments (processeur, mémoire vive, chipsets, BIOS, …) et périphériques de l'ordinateur.

Le **bus informatique** est un câble de liaison ou un dispositif de transmission de données, partagé entre les différents composants de la machine.



- 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).

Pour pouvoir utiliser un ordinateur, un utilisateur doit interagir avec le système d'exploitation grâce à des **périphériques** :

- Les **périphériques d'entrée** servent à donner des informations ou des données au système d'exploitation ;
- Les **périphériques de sortie** servent à faire sortir des informations ou des données du système d'exploitation ;
- Les **périphériques d'entrée/sortie** permettent aussi bien de lire que d'écrire des données.

Exemples :

Périphériques d'entrée	Périphériques de sortie	Périphériques d'entrée/sortie
Clavier, souris, scanner,	Écran, imprimante,	Disques durs internes et externes, clé USB,
micro, webcam,	haut-parleurs,	lecteur/graveur de CD/DVD, tablette, autre
touchpad	enceintes	ordinateur, console de jeux, routeur, modem,
		commutateur réseau, écran tactile, casque
		de réalité virtuelle, multicopieur

1.2.2. L'élément binaire

Le « **Bit** », ou élément binaire, signifie « *binary digit* », c'est-à-dire « 0 » ou « 1 » en numérotation binaire. C'est la plus petite unité d'information numérique manipulable par un ordinateur :

- Avec **1** bit, il est possible d'obtenir deux états : soit 1, soit 0 ;
- Avec **2** bits il est possible d'obtenir 4 états différents ($4 = 2 \times 2 = \mathbf{2^2}$) : **00**, **01**, **10** et **11** ;
- Avec **3** bits on peut obtenir 8 états différents ($8 = 2 \times 2 \times 2 = \mathbf{2^3}$) : **000**, **001**, **010**, **011**, **100**, **101**, **110**, **111** ;
- Avec **4** bits, il est possible d'obtenir $\mathbf{2^4}$ ($= 2 \times 2 \times 2 \times 2$), soit 16 états différents ;

0000, **0001**, **0010**, **0011**, **0100**, **0101**, **0110**, **0111**, …, **1111**

- Avec **8** bits, il est possible d'obtenir $\mathbf{2^8}$ ($= 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$), soit 256 états différents :

00000000, **00000001**, **00000010**, **00000011**, **00000100**, **000000101**, **000000110**, **000000111**, …, **11111111**

Cette information binaire est représentée physiquement :

- Par un signal électrique ou magnétique, qui, au-delà d'un certain seuil, correspond à la valeur 1 (0 en dessous) ;
- Par des aspérités géométriques dans une surface (ex : surface d'un CD / DVD / Blu-ray) ;
- Grâce à des bistables, c'est-à-dire des composants électroniques ayant deux états logiques stables « 0 » et « 1 ». Le passage d'un état à l'autre ne peut s'opérer qu'à la suite d'une action extérieure.

L'**octet** est une unité d'information composée de **8 bits**. Il permet de stocker une information, un caractère tel qu'une lettre, un chiffre …

1 octet = 8 bits

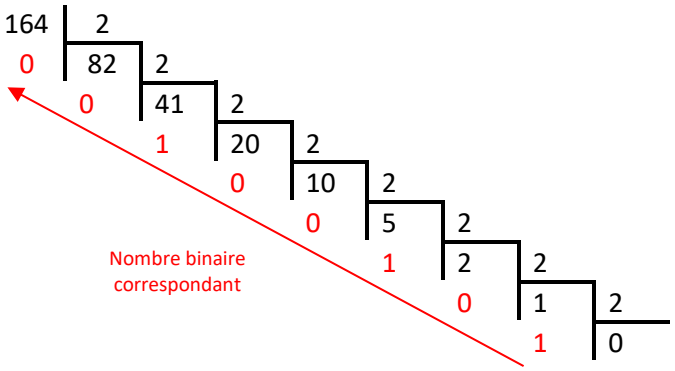
Remarques :

- Une unité d'information composée de 16 bits est généralement appelée « **mot** » (*word* en anglais) ;
- Une unité d'information de 32 bits de longueur est appelée « **mot double** » (*double word* en anglais, d'où l'appellation *dword*).

1.2.2.1. Conversions binaire ↔ décimal

- Conversion **décimal → binaire** : cela revient à convertir un nombre de la base 10 en base 2, en réalisant des divisions successives. On réalise une suite de divisions par 2 en divisant par 2 le quotient obtenu, jusqu'à obtenir un quotient nul. On lit les restes en remontant pour obtenir le nombre en binaire.

Exemple : Que vaut le nombre décimal **164** en binaire ?



164 ÷ 2 = 82 + 0
82 ÷ 2 = 41 + 0
41 ÷ 2 = 20 + 1
20 ÷ 2 = 10 + 0
10 ÷ 2 = 5 + 0
5 ÷ 2 = 2 + 1
2 ÷ 2 = 1 + 0
1 ÷ 2 = **0** + 1

$\Rightarrow (164)_{dec} = (10100100)_{bin}$

$164_{10} = 10100100_2$

- Conversion **binaire → décimal** : on multiplie chaque élément du nombre binaire (bit) par le chiffre 2 élevé à une puissance croissante par pas de 1, comptée à partir de 0 en partant de la droite et on effectue la somme des résultats obtenus pour obtenir sa valeur en décimal.

Exemple : Que vaut l'**octet** (ensemble de 8 bits) **11011010** en décimal ?

	2^7 (= 128)	2^6 (= 64)	2^5 (= 32)	2^4 (= 16)	2^3 (= 8)	2^2 (= 4)	2^1 (= 2)	2^0 (= 1)
Octet	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**
Conversion en décimal	**1** × 128	**1** × 64	**0** × 32	**1** × 16	**1** × 8	**0** × 4	**1** × 2	**0** × 1

Donc : **11011010** = **1** × 128 + **1** × 64 + **0** × 32 + **1** × 16 + **1** × 8 + **0** × 4 + **1** × 2 + **0** × 1 = 218_{10}

Astuce : **11011010** → $1^{128}1^{64}0^{32}1^{16}1^{8}0^{4}1^{2}0^{1}$ → 128 64 0 16 8 0 2 0 = 128 + 64 + 0 + 16 + 8 + 0 + 2 + 0 = 218

Exercice :

Décimal	Binaire
512	1000000000
413	110011101
371	101110011
21	10101

1.2.2.2. Conversions binaire ↔ hexadécimal ↔ décimal

Le **système hexadécimal** est un système de numération positionnel en **base 16** et utilise 16 symboles appelés chiffres hexadécimaux :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F

Un entier s'écrit comme la concaténation de ces chiffres, et sa lecture s'effectue de droite à gauche. Sa valeur vaut la somme des chiffres affectés de poids correspondant aux puissances successives du nombre 16.

- Conversion **hexadécimal → décimal** :

Exemple : que vaut le nombre hexadécimal **04D5** en décimal ?

	16^3 (= 4096)	16^2 (= 256)	16^1 (= 16)	16^0 (= 1)
Hexadécimal	**0**	**4**	**D**	**5**
Conversion en décimal	**0** × 4096	**4** × 256	**13** × 16	**5** × 1

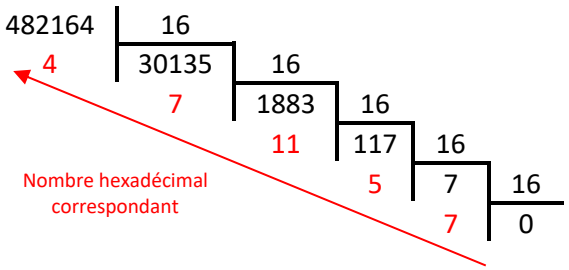
Donc : $04D5_{16} = (0 \times 16^3 + 4 \times 16^2 + 13 \times 16^1 + 5 \times 16^0)_{10} = 1\,237_{10}$

Astuce : 04D5 → $0^{4096}4^{256}D^{16}5^{1}$ → 0×4096 + 4×256 + 13×16 + 5×1 = 1 237

Décimale (base 10) : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexa décimale (base 16) : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Binaire (base 2) : 0, 1

- Conversion **décimal → hexadécimal** : cela revient à convertir un nombre de la base 10 en base 16, en réalisant des divisions successives. On réalise une suite de divisions par 16 en divisant par 16 le quotient obtenu, jusqu'à obtenir un quotient nul. On lit les restes en remontant pour obtenir le nombre en hexadécimal.

Exemple : Que vaut le nombre décimal **482164** en hexadécimal ?



482164 ÷ 16 = 30135 + 4
30135 ÷ 16 = 1883 + 7
1883 ÷ 16 = 117 + 11
117 ÷ 16 = 7 + 5
7 ÷ 16 = 0 + 7

$\Rightarrow (482164)_{dec} = (75B74)_{hex}$

$482164_{10} = 75B74_{16}$

- Conversion **binaire ↔ hexadécimal** : elle se fait en regroupant les chiffres (les bits) quatre par quatre, ou inversement en remplaçant chaque chiffre hexadécimal par 4 chiffres binaires :

Binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Hexadécimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Exemple : que vaut le nombre binaire **101011110101001001111 0111** en hexadécimal ?

Binaire	1 0101 1110 1010 0100 1111 0111						
Conversion en blocs de 4 chiffres	1	0101	1110	1010	0100	1111	0111
Conversion en hexadécimal	1	5	E	A	4	F	7

Donc : $1010111101010010011110111_2 = 15EA4F7_{16}$

Exemple : que vaut le nombre binaire **18FD3E0** en binaire ?

Hexadécimal	1	8	F	D	3	E	0
Binaire	0001	1000	1111	1101	0011	1110	0000
	000110001111111010011111100000 ou 110001111111010011111100000						

Donc : $15EA4F7_{16} = 1010111101010010011110111_2$

Exercice :

Décimal	Binaire	Hexadécimal
22 719 735	110011101	15AACF7
500	111110100	1f4
1 237	10011010101	4D5

Pour aller plus loin :

- Conversion Binaire - Décimal - Hexadécimal

2. Initiation à l'assembleur

2.1. Le langage machine

L'unité de commande du microprocesseur exécute des « instructions machines » (suites d'instructions définies dans des programmes). Ces instructions sont donc, pour cela, codées en binaire puisque le CPU ne gère uniquement que 2 états (symbolisés par un « 0 » et un « 1 ») et constituent ce que l'on appelle le « langage machine » : le processeur est uniquement capable d'interpréter le langage machine.

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

Code opération	Code opérandes

- Un champ « code opération » qui indique au processeur le type de traitement à réaliser. Par exemple le code « 00100110 » donne l'ordre au CPU d'effectuer une multiplication ;

- Un champ « opérandes » qui indique la nature des données sur lesquelles l'opération désignée par le « code opération » doit être effectuée.

2.2. Les instructions machines

Elles sont relativement basiques : on parle d'instructions de bas niveau.

Quelques exemples :

- Instructions arithmétiques : addition, soustraction, multiplication...

Exemple : « *Additionne la valeur contenue dans le registre R1 et le nombre 789 et ranger le résultat dans le registre R0* »

- Instructions de transfert de données qui permettent de transférer une donnée d'un registre du CPU vers la mémoire vive et vice versa.

Exemples :

« *Prendre la valeur située à l'adresse mémoire 111100111 et la placer dans le registre R2* »

« *Prendre la valeur située dans le registre R1 et la placer à l'adresse mémoire 1000001* »

- Instructions de rupture de séquence : les instructions machines sont situées en mémoire vive donc, si, par exemple, l'instruction n°1 est située à l'adresse mémoire 101010111 (343_{10}), alors l'instruction n°2 sera située à l'adresse mémoire 101011000 (344_{10}), l'instruction n°3 sera située à l'adresse mémoire 101011001 (345_{10}) ...

→ Au cours de l'exécution d'un programme, le CPU passe d'une instruction à une autre en passant d'une adresse mémoire à l'adresse mémoire immédiatement supérieure : après avoir exécuté l'instruction n°2 (situé à l'adresse mémoire 344_{10}), le CPU « va chercher » l'instruction suivante à l'adresse mémoire $344_{10} + 1 = 345_{10}$. Les instructions de rupture de séquence d'exécution encore appelées « instructions de saut » ou « de branchement » permettent d'interrompre l'ordre initial sous certaines conditions en passant à une instruction située une adresse mémoire donnée.

Exemple :

Imaginons qu'à l'adresse mémoire 354_{10} nous avons l'instruction :

« *Si la valeur contenue dans le registre R1 est strictement supérieure à 0 alors exécuter l'instruction située à l'adresse mémoire* 4521_{10} ».

⇒ Si la valeur contenue dans le registre R1 est strictement supérieure à 0 alors la prochaine instruction à exécuter est celle située à l'adresse mémoire 4521_{10}, dans le contraire, la prochaine instruction à exécuter est celle qui est située à l'adresse mémoire 355_{10}.

2.3. Les opérandes

Un opérande peut être de trois natures différentes :

- L'opérande est une valeur immédiate : l'opération est effectuée directement sur la valeur donnée dans l'opérande ;
- L'opérande est un registre du CPU : l'opération est effectuée sur la valeur située dans un des registres (R0, R1, R2, ...), l'opérande indique de quel registre il s'agit ;
- L'opérande est une donnée située en mémoire vive : l'opération est effectuée sur la valeur située en mémoire vive à l'adresse XXXXX (codée en binaire). Cette adresse est indiquée dans l'opérande.

Exemple n°1 :

L'instruction machine :

« *Additionne le nombre 125 et la valeur située dans le registre R2, range le résultat dans le registre R1* »

S'interprète ainsi :

- Le « *nombre 125* » est une valeur immédiate ;
- « *la valeur située dans le registre R2* » fait référence à la valeur d'un registre (R2 ici).

Exemple n°2 :

L'instruction machine :

« *Prendre la valeur située dans le registre R1 et la placer à l'adresse mémoire* 512_{10} »

S'interprète ainsi :

- « *à l'adresse mémoire* 512_{10} » fait référence à une adresse mémoire en mémoire vive ;
- « *la valeur située dans le registre R1* » fait référence à la valeur d'un registre (R1 ici).

2.4. L'assembleur

Les instructions machines sont codées sous forme binaire, un programme en langage machine est donc une suite très très longue de « 0 » et de « 1 ». Concevoir un programme codé directement sous forme binaire nécessiterait des dizaines de milliers de « 0 » et de « 1 » et serait ainsi fastidieux et source d'erreur. Pour pallier à ces inconvénients, les informaticiens ont remplacé les codes binaires par des symboles mnémoniques (plus lisibles et plus faciles à retenir qu'une suite de « 0 » et de « 1 »).
→ Les instructions sont inchangées mais au lieu d'avoir une série de « 0 » et de « 1 », on utilise des symboles. Le **programme assembleur** convertit ces symboles mnémoniques en langage machine, ainsi que les valeurs (écrites en décimal) en binaire et les libellés d'emplacements en adresses, en vue de créer, par exemple, un fichier exécutable.
Exemple : l'instruction « additionne le nombre 125 et la valeur située dans le registre R2 , range le résultat dans le registre R1 », qui se traduira en binaire par « 11100010100000100001000000111101 » donnera « `ADD R1,R2,#125` ».

« Additionne le nombre 125 et la valeur située dans le registre R2, range le résultat dans le registre R1 »

⇑
⇓

« `ADD R1,R2,#125` »

Remarque : dans la pratique courante, le même terme *assembleur* est utilisé à la fois pour désigner le langage d'assemblage et le programme assembleur qui le traduit. On parle ainsi de « programmation en assembleur ».

Bibliographie :

- ENIAC (Wikipedia.org) ;
- Machine de Turing (Wikipedia.org) ;
- Architecture de von Neumann (Wikipedia.org) ;
- EDVAC (Wikipedia.org) ;
- Le transistor (Wikipedia.org)
- L'assembleur (Wikipedia.org)

SOURCES

Programme :

Contenus	Capacités attendues	Commentaires
Écriture d'un entier positif dans une base $b \geqslant 2$	Passer de la représentation d'une base dans une autre.	Les bases 2, 10 et 16 sont privilégiées.

Contenus	Capacités attendues	Commentaires
Modèle d'architecture séquentielle (von Neumann)	Distinguer les rôles et les caractéristiques des différents constituants d'une machine. Dérouler l'exécution d'une séquence d'instructions simples du type langage machine.	La présentation se limite aux concepts généraux. On distingue les architectures monoprocesseur et les architectures multiprocesseur. Des activités débranchées sont proposées. Les circuits combinatoires réalisent des fonctions booléennes.

https://www.lyceum.fr/1g/nsi/6-architectures-materielles-et-systemes-dexploitation/1-architecture-dun-ordinateur
https://www.info-nsi.fr/nsi-assembleur/Cours_Programmation_assembleur.pdf
https://pixees.fr/informatiquelycee/n_site/nsi_prem_von_neu.html
https://cache.media.eduscol.education.fr/file/NSI/76/9/RA_Lycee_G_NSI_arch_von_neu_1170769.pdf