Hardware - Le binaire

Difficulté : Débutant

Notions : Binaire, base 2

I. Introduction

> **Il y a 10 types de personnes dans le monde : celles qui savent compter en binaire et les autres.**
>
> *- Blague de geek*

Le langage binaire est le langage informatique le plus ***bas niveau***.

En effet un ordinateur fonctionnant avec du courant électrique, il n'y a que deux états possible pour un bit à l'instant T.

Circuit fermé, présence de courant : ***1***

Circuit ouvert, absence de courant : ***0***

De par ses deux valeurs possible, le binaire est donc un système de calcul en ***base 2***.

II. Les bases de calcul

> **Information** : Quel que soit le calcul, il est à noter que la première valeur possible est toujours 0 (Zero).

2.1 Base 10

La base 10 est la base naturelle pour l'humain et celle courrament utilisée dans le monde.

De par le fait que les mains humaines disposent de 10 doigts, c'est en effet plus pratique et instinctif pour compter.

on a donc les valeur suivantes possibles :

Valeurs base 10
0 =00
1 = 01
2 = 02
3 = 03
4 = 04
5 = 05
6 = 06
7 = 07
8 = 08

9 = 09

Une fois la valeur maximum atteinte (09) pour incrémenter au delà, on ajoute une dizaine devant l'unité et on continue ainsi à incrémenter.

10, 20, 30 etc... etc...

2.1 Base 2

En binaire le principe reste le même qu'en base 10 , on à donc 0, 1 et quand la valeur maximum est atteinte, on incrémente.

Valeurs base 2
0 =0
1 = 1
2 = 10
3 = 11
4 = 100
5 = 101
6 = 110
7 = 111
8 = 1000
9 = 1001
etc...

Si de prime abord, cela peut ne pas sembler naturel, il s'agit juste de reproduire la même logique de comptage que la base 10, mais avec seulement 2 valeurs disponible.

III. l'Octet

3.1 Construction

Heureusement, pour rendre tout cela plus lisible et car un bit seul ne fait pas sens dès lors qu'il s'agit de compter au delà de 1, on va regrouper ces bits par paquets. Ces packets sont appelés des ***registres***.

Le registre le plus connu et celui qui sera courrament utilisé est l'***octet*** (un paquet de 8 bits).

Basé sur ce principe on aura donc la représentation suivante : ***00000000***

Et là encore, si cela paraît compliqué, il existe un moyen simple et mnémotechnique de se représenter facilement les valeurs en binaire.

En effet, le binaire à une particularité, pour chaque bit que l'on rajoute devant, on double les possibilité, ce qui fait que chaque bit a une valeur différente. Cela s'appelle un ***Poid*** de bit.

Pour se donner une bonne idée, voici la valeur de chaque bit dans un octet :

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

3.2 Conversion base 10 / Base 2

Dès lors, pour convertir des nombres binaires en base 10 et de base 10 en binaire, cela deviens facile.

Il suffit d'additionner les valeurs de tous les bits égaux à 1. Ainsi,

01101011 par exemple se note :

128	64	32	16	8	4	2	1
0	**1**	**1**	0	**1**	0	**1**	**1**

soit : 64 + 32 + 8 + 2 + 1 = ***107***.

De la même manière, pour convertir un nombre en base 10 en binaire, il faudra additionner en partant de la droite les valeurs pour arriver à la somme du nombre que l'on veut et passer leurs valeurs à 1.

218 par exemple se note :

Détail de l'opération

128 < 218, donc on part avec 128 = 1.

128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0

128 + 64 = 192.

192 < 218, le bit avec une valeur de 64 est donc à 1.

128	64	32	16	8	4	2	1
1	**1**	0	0	0	0	0	0

192+32 = 224.

224 > 218 donc le bit avec une valeur de 32 est à 0. On passe à la valeur suivante.

128	64	32	16	8	4	2	1
1	**1**	**0**	0	0	0	0	0

192+16 = 208.

208 < 218, donc le bit avec une valeur de 16 est à 1.

128	64	32	16	8	4	2	1
1	**1**	**0**	**1**	0	0	0	0

208+8 = 216.

216 < 218, donc le bit avec une valeur de 8 est à 1.

128	64	32	16	8	4	2	1
1	**1**	**0**	**1**	**1**	0	0	0

216+4 = 220.

220 > 218, donc le bit avec une valeur de 4 est à 0.

128	64	32	16	8	4	2	1
1	**1**	**0**	**1**	**1**	**0**	0	0

216+2 = 218.

218 = 218. On passe donc notre dernier bit (celui dont la valeur est 2) à 1. La valeur des bits restant à droite sera donc 0.

128	64	32	16	8	4	2	1
1	**1**	**0**	**1**	**1**	**0**	**1**	**0**

128	64	32	16	8	4	2	1
1	1	0	1	1	0	1	0

> **Note** : Si l'on additionne toutes les valeurs, le résultat est de 255. Par conséquent, si l'on doit traiter des nombres supérieurs à 255, on ajoutera simplement un octet devant, et l'on poursuivra le tableau de la manière suivante.

32768	16384	8192	4096	2048	1024	512	256
octet 2							

128	64	32	16	8	4	2	1
octet 1							

3.3 Les Échelles de tailles

Partant du principe qu'un octet peut stocker toutes sortes de données et que les bits sont donc l'unité de mesure la plus petite, si l'on veut stocker plus de données, on sera donc obligé d'accoler des octets, ou utiliser des registres plus grands.

Au bout d'un moment, il faudra que nos échelles de mesures continuent d'avoir un sens.

Si on empile beaucoup d'octets, on ne va pas continuer à dire, la taille de ce bloc est de 1 228 283 219 233 274 272 octets.

Pour cela, sur le même principe que les tableau de conversion, on va utiliser d'autre échelles.

A l'instar des unités de mesures classiques (gramme, kilogramme) (centimètre, mètre, kilomètre) il va falloir établir un tableau de conversion pour faire la mise à l'échelle.

Les unités sont donc les suivante :

- **bit** (unité)
- **octet** (8 bits)
- **KiloOctet** ou **Ko** (1024 octets)
- **MegaOctet** ou **Mo** (1024 Ko)
- **GigaOctet** ou **Go** (1024 Mo)
- **TéraOctet** ou **To** (1024 Go)
- **PétaOctet** ou **Po** (1024 To)

> **Note** : A l'américaine, l'octet se dit ***Byte***. Un equivalent est donc, le GigaByte ou Mega byte, etc... Notés : Kb, Mb, Gb, etc...

> **Trivia** : au dessus du péta, il y a l'Exa, le Zeta, le Yotta, Ronna, Quetta, ...

IV. Les opérations en binaire

4.1 Addition

Cela fonctionne plus ou moins comme une addition standard, avec les retenues.

Selon les principes suivants :

0 + 0 = 0

(0 + 1 ou 1+0) = 1

1+1 = 10 (équivalent de 2) Donc dans ce cas, l'on pose 0 et on retient 1.

Par exemple : ***00100101 + 10110010*** (soit 37 + 178)

Détail de l'opération

retenue								
Nombre 1	0	0	1	0	0	1	0	1
Nombre 2	1	0	1	1	0	0	1	0
Total								**1**

On commence par la droite. 1+0 = 1, pas de retenue.

retenue								
Nombre 1	0	0	1	0	0	1	0	1
Nombre 2	1	0	1	1	0	0	1	0
Total				**1**	**0**	**1**	**1**	1

0 + 1 = 1, pas de retenue,
1 + 0 = 1, pas de retenue,
0 + 0 = 0, pas de retenue,
0 + 1 = 1, pas de retenue.

retenue		**1**						
Nombre 1	0	0	1	0	0	1	0	1
Nombre 2	1	0	1	1	0	0	1	0
Total			**0**	1	0	1	1	1

*1 + 1 = 10. je pose **0** et je retiens **1**.*

retenue		**1**						
Nombre 1	0	**0**	1	0	0	1	0	1
Nombre 2	1	**0**	1	1	0	0	1	0
Total		**1**	0	1	0	1	1	1

***1** + **0** = **1** puis **1** + **0** = **1**. Pas de retenue*

retenue		1						
Nombre 1	0	0	1	0	0	1	0	1
Nombre 2	1	0	1	1	0	0	1	0
Total	**1**	1	0	1	0	1	1	1

enfin, 0 + 1 = 1, pas de retenue.

1

Nombre 1	0	0	1	0	0	1	0	1
Nombre 2	1	0	1	1	0	0	1	0
TOTAL	1	1	0	1	0	1	1	1

on a donc ***00100101 + 10110010*** = ***11010111*** (soit 37 + 178 = 215)

4.2 Soustraction

La soustraction est un peu plus complexe que l'addition, en effet il va faloir jongler sur les chiffres.

Selon les principes suivants :

0 - 0 = 0

1 - 1 = 0

1 - 0 = 1

0 - 1 = Ne peux pas directement être résolu, nécessite un emprunt.

Par exemple : ***10110010 - 00100101*** (soit 178 - 37)

Détail de l'opération

retenue							*0*	*10*
Nombre 1	1	0	1	1	0	0	~~1~~	~~0~~
Nombre 2	0	0	1	0	0	1	0	1
Total								**1**

On commence par la droite. 0-1, emprunt à gauche.
*On barre le 1 à gauche on le remplace par un **0**. (car 1-1=0)*
On ajoute 2 soit $\mathbf{10}_2$ en remplacement du 0 ce qui donne le calcul 2-1 = 1.

retenue							*0*	*10*
Nombre 1	1	0	1	1	0	0	~~1~~	~~0~~
Nombre 2	0	0	1	0	0	1	0	1
Total							**0**	**1**

0 - 0 = 0 pas d'emprunt.

retenues								
				0	~~*10*~~		*0*	*10*
Nombre 1	1	0	1	~~1~~	~~0~~	0	~~1~~	~~0~~
Nombre 2	0	0	1	0	0	1	0	1
Total							**0**	**1**

0 - 1 non solvable sans emprunts.

Pas de 1 adjacent à gauche, on va donc devoir aller chercher le premier disponible pour l'emprunter.

*on barre le 1 que l'on remplace par **0** (1-1=0). Puis on barre le 0 à sa droite pour remplacer par 2 soit $\mathbf{10}_2$.*

retenues					**1**	*10*		
				0	~~*10*~~		*0*	*10*
Nombre 1	1	0	1	~~1~~	~~0~~	~~0~~	~~1~~	~~0~~
Nombre 2	0	0	1	0	0	1	0	1
Total						**1**	**0**	**1**

On continue à décaler. On soustrait 1 de 2 ($\mathbf{10}_2$), il reste **1**.
Enfin, on reporte le 1 à droite et on exécute le calcul.
2 - 1 = 1.

retenues					**1**	*10*		
				0	~~*10*~~		*0*	*10*
Nombre 1	1	0	1	~~1~~	~~0~~	~~0~~	~~1~~	~~0~~
Nombre 2	0	0	1	0	0	1	0	1
Total	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

On continue ainsi :
1 - 0 = 1
0 - 0 = 0
1 - 1 = 0
1 - 0 = 1

Nombre 1	1	0	1	1	0	0	1	0
Nombre 2	0	0	1	0	0	1	0	1
TOTAL	1	0	0	0	1	1	0	1

on a donc ***10110010 - 00100101*** = ***10001101*** (soit 178 - 37 = 141)

V. Les fonctions binaires

Les fonctions en binaire ne sont pas des opérations à proprement parler mais le résultat d'un test livré par ce que l'on appelle en informatique et en électronique une porte logique.

5.1 AND (et)

0	0	0
0	1	0
1	0	0
1	1	1

Le résultat de la condition est vrai uniquement si les deux termes sont vrais.

5.2 OR (ou)

0	0	0
0	1	1
1	0	1
1	1	1

Le résultat de la condition est vrai si au moins l'un des deux termes est vrai.

5.3 XOR (ou exclusif)

0	0	0
0	1	1
1	0	1
1	1	0

Le résultat de la condition est vrai seulement si l'un des deux terme est vrai.

> **Trivia** : c'est ce système là qui va être utilisé pour calculé des bits de parité. Dans les matrices RAID par exemple.

Pour aller plus loin : [Wikipedia | Tables de vérité]

VI. Conclusion

Au sein de l'ordinateur, tout est binaire. De la donnée écrite sur les supports de stockage, jusqu'à l'information transitant par le réseau, en passant par les opérations processeurs et le contenu de la mémoire.

Nous manipulons juste de moins en moins ce type de contenu au fil de l'évolution des langages informatiques et de programmation.

Mais il est cependant important de comprendre (au moins en théorie) comment fonctionne l'informatique sur sa couche la plus basique.

Revision #3
Created 2026-07-17 20:30:08 UTC by Olivier
Updated 2026-07-17 20:30:55 UTC by Olivier