La base de l'informatique : le binaire

Table des matières

1 Introduction: de quoi parle-t-on ?

Le terme *informatique* désigne le traitement automatique de l'information par des entités, qui sont généralement des ordinateurs.

1.1 L'information

Il n'est pas aisé de définir ce que peut être l'*information*. Intuitivement, on peut poser qu'elle est liée aux concepts de *pensée* et de *signification,* peut être échangée par des entités dotées d'une certaine forme d'intelligence, et transformée pour acquérir de l'utilité.

Wikipedia, dans son article « Information », explique en guise d'introduction:

> *« L'information désigne à la fois le message à communiquer et les symboles utilisés pour l'écrire ; elle utilise un code de signes porteurs de sens tels qu'un alphabet de lettres, une base de chiffres, des idéogrammes ou pictogrammes. »*[1]

L'information comporte donc deux aspects indissociables et complémentaires:

- Un message, qui désigne le **sens** de l'information, sa **signification**.
- Une **représentation**, liée au **langage** utilisé pour modéliser l'information.

La signification désigne en quelque sorte la capacité d'un message a être compris ou conscientisé par son récepteur. Il est possible de construire un message lisible mais non compréhensible.

Le site web « DanstonChat », qui recense les commentaires les plus amusants proférés dans des salons de discussion en ligne, apporte parfois quelques bijoux: *« on touche à ma brune aux planètes solides, un effet par hasard et à organiser les observe, qui les primates. »*[2]

Cet exemple est tout à fait caractéristique d'un message n'ayant aucune signification, bien qu'il soit rédigé dans un alphabet et avec des mots compréhensibles.

De même, sans langage compréhensible, une information ayant une signification peut très bien ne pas être comprise[3].

Observons le message suivant: Jeg elsker dig

Ce message possède un sens et est rédigé dans un langage compréhensible par une grande quantité d'entités (environ 5.500.000 personnes). Cependant, à moins que vous ayez étudié le danois, il est peu probable que vous puissiez en comprendre le sens. Le langage est donc une composante essentielle de la compréhension d'une information. Pour satisfaire votre curiosité, le message en français est « Je vous aime ».

1 http://fr.wikipedia.org/wiki/Information , version du 8 avril 2013
2 http://danstonchat.com/1215.html . Il y en a beaucoup d'autres qui valent leur pesant de cacahuètes.
3 Certains personnes brouillent intentionnellement leur message dans une langue incompréhensible par un non initié, comme par exemple la langue des oiseaux (http://fr.wikipedia.org/wiki/Langue_des_oiseaux)

1.2 La notion de codage

Comme nous l'avons évoqué au chapitre 1.1, toute information doit être représentée à l'aide d'un langage qui permet sa compréhension par son récepteur. Les langages sont eux-mêmes composés de symboles permettant leur écriture. Ainsi, les mots de la langue française sont-ils composés de lettres extraites de l'**alphabet latin**. Les nombres sont, quant à eux, composés de **chiffres** dits **arabes**. Les symboles et les langages sont essentiels pour représenter des informations.

Lorsqu'une information est représentée dans un langage composé de symboles, on parle de **codage de l'information**. Le codage peut prendre des formes variées. Il doit être maîtrisé pour appréhender le message délivré par l'information.

Observons l'information suivante:

Le codage devrait vous rappeler des souvenirs: il s'agit d'un code-barre (remarquez que le terme « code-barre » comprend le mot « code »). Sans la connaissance des symboles, vous ne pouvez déterminer la teneur du message, alors qu'il s'agit pourtant d'une représentation un peu différente des lettres classiques utilisées en français. Nous sommes donc en présence ici d'un langage qui nous est familier (il s'agit bel et bien d'une phrase en français) mais d'un codage différent de celui que nous utilisons habituellement. Essayez donc de dénicher un lecteur de code-barre si vous êtes curieux[4].

Observons maintenant le code suivant: 17701200101

Il s'agit d'un numéro de matricule typique du personnel de la Fédération Wallonie-Bruxelles.

Ici, les symboles sont connus, de même que le langage: nous sommes vraisemblablement en présence d'un nombre: dix-sept milliards sept cent un millions deux cent mille cent un. Pourtant, un petit élément supplémentaire doit être connu pour extraire vraiment le message véhiculé par ce nombre: il s'agit en fait d'un code représentant le sexe, suivi de la date de naissance et d'un numéro d'ordre.

Sexe (1 chiffres)	Année (2 chiffres)	Mois (2 chiffres)	Jour (2 chiffres)	Ordre (4 chiffres)
1	77	01	20	0101

L'employé qui a ce numéro est un homme né le 20 janvier 1977, numéroté 0101. L'ordre des symboles a ici toute son importance. Et on peut constater qu'il est possible d'utiliser des chiffres pour composer autre chose que des nombres: par exemple, le premier chiffre représente ici le sexe de la personne (1 pour homme, 2 pour femme); il aurait été possible aussi d'écrire M ou F, ou encore A ou B. Dans la vie courante, on utilise souvent des chiffres et nombres pour représenter bien d'autres choses que des valeurs strictement numériques (pensez aux codes postaux, ils ne portent aucun message numérique mais représentent une localité).

4 Bon, ça va, je vous le dis. Ce code-barre signifie « L'informatique, c'est chouette ! »

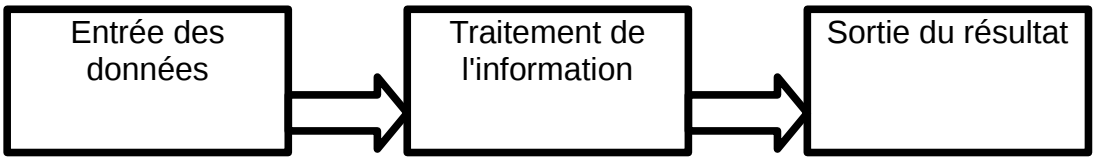
1.3 Le traitement de l'information

A l'origine, l'*informatique* désigne le traitement automatique de l'information. Plus généralement, on peut parler du développement pratique de la *théorie de l'information*.

Lorsqu'il s'agit du traitement de l'information, on peut constater qu'il existe un schéma universel du traitement réalisé par un dispositif: on lui présente une information et il en donne une autre en retour, considérée comme plus utile. La première est l'information d'*entrée* ou *donnée* et la seconde est l'information de *sortie* ou *résultat*, le produit fini du traitement. Les ordinateurs constituent des dispositifs de traitement de l'information et à ce titre, ils agissent exactement selon le schéma précité. Prenons quelques cas pratiques:

- La consultation de sites web: on donne à l'ordinateur une adresse à consulter (information d'entrée), il télécharge alors la page web correspondante (traitement de l'information) et fournit le contenu à l'utilisateur (résultat).
- Un jeu vidéo de style FPS[5]: on appuie sur une touche de direction (donnée d'entrée), l'ordinateur calcule la nouvelle disposition du monde autour du joueur (traitement de l'information) et fournit une nouvelle vue au joueur (résultat).

Dégageons le schéma général du traitement de l'information:



N'oublions pas que l'anglais est la langue principale de l'informatique, aussi traduisons les termes dans la langue de Shakespeare:

Prenons un exemple concret: mettons que vous souhaitez connaître la réponse à la question « combien font 12 x 43 ? ».

Quelles sont les informations d'entrée ? (vous avez 5 secondes).

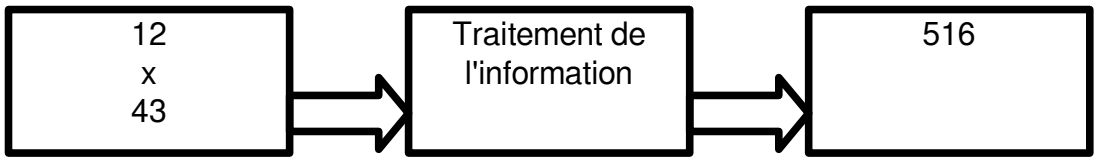
Avez-vous répondu 12 et 43 ? Vous avez partiellement raison. En effet, 12 et 43 constituent les données nécessaires pour effectuer une multiplication. Mais il ne faut pas oublier l'opération elle-même, qui détermine la manière dont le traitement va être effectué. En effet, si la question avait été « combien font 12 + 43 ? », les données 12 et 43 sont toujours présentes mais le traitement aurait été tout différent. On peut donc voir qu'il existe une grande variété de traitements possibles et que le choix de l'un d'eux constitue une donnée à part entière.

Quelle sera le résultat ? (non, ne sortez pas votre calculatrice, vous savez multiplier depuis vos études primaires).

Si vous avez répondu 516, vous savez encore calculer. Sinon, recommencez en étant plus attentif.

5 « First Person Shooter », un type de jeu dans lequel le joueur est immergé dans un monde en 3D et voit à travers les yeux du personnage. Vous ne le saviez pas ?

Si l'on adapte le schéma ci-dessus à ce cas particulier, nous pouvons écrire:



Lorsque vous avez effectué l'opération demandée ci-dessus, vous vous êtes comportés comme un ordinateur car vous avez appliqué un traitement aux informations d'entrée et en avez dégagé un résultat.

1.4 Conclusion: qu'est-ce qu'on fait avec un ordinateur ?

Un ordinateur est un dispositif du traitement de l'information. Il est donc nécessaire de lui présenter des informations qui aient à la fois un sens et une forme compréhensible par lui, afin qu'il puisse effectuer la transformation de l'information qui lui est demandée. Et de même, il conviendra de traduire les résultats de manière que nous puissions en prendre connaissance (sinon, les ordinateurs ne serviraient à rien).

2 Le codage de l'information dans l'ordinateur

2.1 Introduction

Les ordinateurs font partie d'une grande famille d'objets que l'on désigne par le nom générique de *machines*. Les machines ont généralement pour tâche d'amplifier les capacités humaines dans un domaine précis: une automobile permet de se déplacer plus vite qu'à pieds, un téléphone permet de porter un message sonore au lieu de devoir se déplacer pour le délivrer soi-même...

Un ordinateur est une machine électronique qui a pour tâche de transformer de l'information plus vite que ne pourrait le faire un être humain. En ce sens, il s'agit d'une machine tout à fait particulière: là où la plupart de ses « cousines » manipulent des objets concrets, l'ordinateur manipule, lui, des objets abstraits.

Nous avons vu dans le chapitre 1.1 que l'information était constituée d'un *message* présenté dans un certain *langage* et que chacun participait à la compréhension de l'information. Nous allons donc nous attarder à apprendre comment communiquer avec un ordinateur de manière qu'homme et machine puissent dialoguer.

Cela peut sembler étrange qu'il faille réapprendre cela, car les moyens de communication actuels sont connus et maîtrisés par la majorité des utilisateurs d'ordinateurs: le clavier, la souris et l'écran tactile en entrée, l'écran ou l'imprimante en sortie... Cependant, il ne s'agit pas ici de comprendre comment faire entrer les données ou sortir les résultats, mais **comment l'information est représentée et comprise par l'ordinateur.** Ceci nous permettra du même coup de comprendre comment formaliser les traitements afin de les adapter au **codage** de l'information par l'ordinateur.

2.2 Le codage électronique

2.2.1 Au départ est l'électricité

Le caractère électronique de l'ordinateur constitue la base du codage de l'information dans celui-ci: toute information doit être décrite ou portée par un courant électrique. Cela amène un grand nombre de contraintes dues aux pertes et incertitudes inévitables qui entourent la manipulation du courant.

Depuis que les phénomènes électriques sont connus et maîtrisés, on a réussi à mesurer nombre de caractéristiques intrinsèques des courants qui ont permis des avancées technologiques cruciales. Les plus triviales sont l'intensité et la tension. La première désigne grossièrement le nombre de charges électriques (en quelque sorte les unités fondamentales du courant) qui parcourent un conducteur en un temps donné, la deuxième désigne l'énergie véhiculée par ces charges. On pourrait donc supposer qu'il serait facile de coder des informations dans les variations de ces caractéristiques du courant: par exemple, une faible intensité de référence représenterait 1, une intensité double représenterait 2, une intensité triple représenterait 3 et ainsi de suite... On pourrait aussi imaginer faire de même avec la tension.

Malheureusement, les courants électriques sont soumis à diverses contraintes qui rendent ce codage assez aléatoire: la distance qu'un courant doit parcourir et des perturbations externes (notamment magnétiques) peuvent drastiquement modifier les caractéristiques du courant. Pour s'en convaincre, il suffit de mesurer par exemple la tension qui règne dans une banale prise électrique. Le réseau belge est supposé délivrer un courant de 220 volts. Or, cette valeur est très rarement atteinte dans les foyers: elle est très souvent dépassée. Il est donc peu commode d'utiliser des caractéristiques intrinsèques pour coder une information.

Une caractéristique qui peut se mesurer facilement et qui est peu sujette à des variations

aléatoires est tout simplement la présence ou l'absence de courant. Si vous introduisez vos doigts dans une prise électrique[6] vous savez immédiatement si du courant est présent ou pas, quelle que soit la variation de tension qu'il ait pu subir !

2.2.2 Ensuite vient le codage

Un fil peut donc être dans deux états: un fil qui véhicule du courant est dit « **chaud** », un fil qui n'en véhicule pas est dit « **froid** ». La présence ou l'absence de courant constitue une représentation compréhensible par une machine électronique comme un ordinateur. Pour notre propre compréhension, écrivons **C** quand l'état est chaud et **F** quand l'état est froid, afin de faire correspondre la représentation compréhensible par l'ordinateur avec une représentation compréhensible par un être humain (qui sait lire l'alphabet latin, mais c'est normalement votre cas si vous comprenez ceci).

Si on mesure l'état d'un fil, on peut donc découvrir deux états différents:

C	**F**

A partir de là, on peut aussi donner une autre signification à chaque état: blanc ou noir, féminin ou masculin, ouvert ou fermé... Tout dépend de ce que l'on veut représenter ! Mais pour le moment, continuons avec notre représentation la plus simple.

On peut aussi envisager de multiplier les fils. Prenons le cas de deux fils: chacun peut être dans l'état C ou l'état F. On a donc les combinaisons possibles suivantes:

FF	**FC**	**CF**	**CC**

Pour deux fils, on peut donc considérer qu'il existe 4 états possibles. Si l'on décide que chaque état correspond, par exemple, à un nombre, nous voilà en mesure d'exprimer 4 nombres différents à l'aide de deux fils, cela grâce à un **codage** adapté à la nature électrique de notre système de représentation. Par exemple, on pourrait décider de poser FF pour 0, FC pour 1, CF pour 2 et CC pour 3. Attention, l'ordinateur, quant à lui, n'a aucune conscience de la **signification** de chaque état, il s'agit ici d'une convention pour nous-mêmes. Pour lui, il s'agit toujours de fils chauds et de fils froids.

Essayons avec 4 fils. Prenez votre temps pour énumérer toutes les combinaisons possibles:

FFFF	FFFC	FFCF	FFCC	FCFF	FCFC	FCCF	FCCC	CFFF	CFFC	CFCF	CFCC	CCFF	CCFC	CCCF	CCCC

En voilà 16 ! Nous avons donc maintenant la possibilité de représenter 16 nombres différents, par exemple. Ou encore 16 mots, 16 pays, 16 marques de voitures... Encore une fois, tout dépend de la **signification** que l'on veut donner à chaque état. L'ordinateur n'en décide pas, c'est à nous de donner cette signification avec intelligence et en accord avec les informations que nous souhaitons traiter. Faut-il ensuite « expliquer » à l'ordinateur la signification de chaque représentation ? En fait non, nous verrons par la suite que l'ordinateur est parfaitement capable de nous rendre de grands services sans rien comprendre à ce qu'il fait et manipule.

6 C'est juste une image comique, ne le faites pas. Ça fait très mal et ça peut vous tuer. Vraiment !

En résumé:

Nombre de fils	Nombre d'états possibles
1	2
2	4
4	16

Il existe une relation entre le nombre de fils et le nombre d'états possibles. Nous allons tâcher de la découvrir.

On peut remarquer que plus le nombre de fils augmente, plus le nombre d'états possibles augmente lui aussi, et de manière assez rapide d'ailleurs ! Le cas de 3 fils est assez éclairant: en observant le tableau précédent, on peut éventuellement induire qu'il suffit d'élever le nombre de fils au carré pour obtenir le nombre d'états possibles, soit 9 états pour 3 fils.

Dès lors, essayez, en guise d'exercice, de découvrir les 9 états possibles... Vous avez 5 minutes !

Si si, cherchez encore un peu !

Bon, vous en avez trouvé 8 ? Quel peut bien être ce satané neuvième état...

FFF	FFC	FCF	FCC	CFF	CFC	CCF	CCC

Non, décidément, il ne semble pas y en avoir d'autre.

Remarquez que dans le cas d'un seul fil, le nombre d'états possibles n'est pas le carré du nombre de fils, mais son double. Dans le cas de 2 fils, le nombre d'état est aussi le double du nombre de fils. Mais pas dans le cas de 4 fils...

Donc, on peut vraisemblablement éliminer l'hypothèse selon laquelle le nombre d'état est le carré du nombre de fils. Traçons à nouveau le tableau en y intégrant le cas de 3 fils:

Nombre de fils	Nombre d'états possibles
1	2
2	4
3	8
4	16

Cela commence à devenir plus clair: à chaque fil supplémentaire, le nombre d'états possibles double. En fait, c'est tout à fait logique et normal. Prenons le passage de 2 à 3 fils et observons cela avec quelques petits tableaux:

Avec 2 fils, nous avons 4 états possibles résumés dans le tableau suivant (que vous avez déjà vu précédemment):

FF	**FC**	**CF**	**CC**

Ajouter un fil revient à passer à une configuration dans laquelle il y a 2 fils et un fil supplémentaire. Ce dernier peut être soit chaud, soit froid. Nous avons donc les deux configurations suivantes:

- Lorsque le fil supplémentaire est froid:

F FF	**F** FC	**F** CF	**F** CC

- Lorsque le fil supplémentaire est chaud:

C FF	**C** FC	**C** CF	**C** CC

Ajouter un fil double donc le nombre d'états possibles !

Si vous êtes un peu observateur, vous remarquerez qu'on peut facilement induire la formule générale permettant de calculer le nombre d'états possibles par rapport à un nombre de fils:

$$N_{états} = 2^{\text{nombre de fils}}$$

C'est un rapport exponentiel. Ainsi, il vous est très facile maintenant de déterminer combien d'états possibles existent lorsqu'on dispose de 10 fils: 2^{10} soit 1024 états possibles.
Pourriez-vous déterminer combien d'états possibles existent lorsqu'on dispose de 6, 8 ou 12 fils ?
Ne croyez pas que cela soit difficile et que vous deviez maintenant sortir votre calculatrice: les informaticiens ont depuis longtemps dressé des tables qu'ils ont pris soin d'étudier par cœur. Vous savez ce qu'il vous reste à faire...

Nombre de fils	Nombre d'états	Nombre de fils	Nombre d'états
1	2	**9**	512
2	4	**10**	1024
3	8	**11**	2048
4	16	**12**	4096
5	32	**13**	8192
6	64	**14**	16384
7	128	**15**	32768
8	256	**16**	65536

Avec seulement 8 fils, on peut déjà représenter 256 significations différentes ! Bien assez pour y établir toutes les lettres, tous les chiffres et même les signes de ponctuation que nous utilisons dans notre alphabet latin. Néanmoins, si on associe chaque état à un nombre, nous ne pouvons en représenter que 256. Dans la majorité des calculs que nous pourrons confier à un ordinateur, c'est bien trop peu. C'est pourquoi la majorité des ordinateurs actuels travaillent avec des associations de fils bien plus larges. Nous aurons l'occasion d'y revenir.

2.3 Donner du sens à cette représentation

2.3.1 Représenter un nombre de manière générale

Si nous souhaitons représenter des nombres à l'aide de fils chauds et froids, il serait sans doute commode de définir une bonne fois pour toutes à quoi vont ressembler les nombres dans ce

nouveau codage et quelles seront les significations des différents états.

Il s'agit donc de nous mettre d'accord sur un **codage** et une **représentation** afin d'exprimer différents **messages** (à savoir, des nombres) à l'aide des états possibles d'un ensemble de fils qui peuvent être chauds ou froids. Si vous avez correctement suivi et compris votre cours de mathématiques, vous savez qu'il existe une infinité de nombres[7]. Or, on ne peut multiplier le nombre de fils à l'infini. C'est pourquoi un ordinateur ne pourra, par conception, jamais manipuler de nombres aussi grands qu'on le voudrait dans ce système de représentation. Il faudra vous y faire... En pratique, il suffit de choisir le nombres de fils avec soin pour être sûr d'englober toutes les valeurs possibles qui peuvent apparaître dans les opérations que nous réalisons avec un ordinateur.

On pourrait par exemple poser arbitrairement des équivalences entre des nombres et des combinaisons de « chauds » et de « froids ». Il faudrait alors apprendre par cœur toutes les combinaisons, ce qui serait plutôt fastidieux. Il serait plus judicieux de définir une **manière simple** et rigoureuse qui permette d'**associer un nombre à chaque combinaison**, grâce à un ensemble de règles connues et immuables. Or, bien que vous ne le sachiez pas, vous manipulez tous les jours un système de représentation des nombres qui va nous permettre de nous en tirer.

Tout d'abord, afin de faciliter les choses, **cessons d'écrire C pour chaud et F pour froid** et choisissons un codage plus universel. A cela plusieurs raisons:

- nous n'avons décidément pas l'habitude d'écrire des nombres en utilisant des lettres. Nous utilisons des chiffres et les habitudes sont souvent difficiles à changer.
- C et F sont valables pour le français, mais deviendraient H et C en anglais (C signifiant même *cold*, donc froid !), W et K en néerlandais... Bref, si nous voulons communiquer avec une personne parlant une autre langue, cela risque de compliquer les choses.

Puisque nous voulons représenter des nombres, utilisons des chiffres ! Combien de fois y a-t-il du courant sur un fil froid ? Et sur un fil chaud ? Observez ceci:

0 = froid 1 = chaud

Nous y voilà ! 0 et 1 sont des chiffres universellement reconnus, lisibles aussi bien par un francophones que par un anglophone, un néerlandophone, un russophone... Remarquez qu'il ne s'agit pas ici des *nombres* 0 et 1, mais des *chiffres*. Pour rappel, l'ensemble des chiffres constitue le **codage** utilisé pour représenter les nombres.

Ainsi, en utilisant 0 et 1 au lieu de F et C, on peut écrire des états qui ressemblent un peu plus à des chiffres: 110010 ou 000001 sont plus parlants que CCFFCF ou FFFFFC lorsqu'il s'agit de représenter des nombres.

Reste maintenant à organiser tout cela pour que nous puissions donner une signification à chaque combinaison de 0 et de 1. Et sur ce point, vous possédez d'ores et déjà une méthode d'une redoutable efficacité pour organiser des chiffres et en faire des nombres.

2.3.2 Le système positionnel habituel: le système décimal

Observons, par exemple, le message suivant:

546

Si vous lisez à haute voix, il y a de fortes chances que vous prononciez « cinq cent quarante-six ». Grâce à notre système de codage (les chiffres), à notre système de représentation (on écrit les

7 Voire plus d'une infinité. Le mathématicien Georg Cantor a découvert qu'il existe des infinis dénombrables et des infinis indénombrables. C'est une découverte majeure du XX^{e} siècle.

chiffres l'un à côté de l'autre), nous pouvons dégager le message cinq cent quarante-six.

Observons maintenant ce nouveau message:

465

Vous conviendrez qu'il s'agit ici de quatre cent soixante-cinq, soit un nombre tout différent du précédent. Pourtant, sa représentation est constituée des **mêmes chiffres**, mais écrits dans un ordre différent. Notre système de représentation des chiffres accorde donc une importance capitale à **la position** de chaque chiffre dans le nombre. La position d'un nombre dans un chiffre s'appelle son **rang**. Vous le saviez, mais le rappeler est très important pour comprendre la suite.

Le RANG d'un chiffre désigne sa position dans un nombre

Au fond, comment notre système de représentation des nombres fonctionne-t-il réellement ? Cela fait longtemps que l'on vous a appris à manipuler les chiffres et les représentations des nombres, mais il est peu probable qu'on vous ait expliqué **pourquoi** on a choisi ce système. Essayons d'y voir plus clair.

Il convient d'abord de comprendre que nous disposons, pour le codage des nombres, de 10 symboles, 10 chiffres. Ce nombre de chiffres n'a pas été choisi au hasard: ouvrez vos mains, tendez les bras devant vous et pliez vos poignets comme pour pousser quelque chose qui serait devant vous: à moins d'une anomalie, vous possédez... 10 doigts ! Les enfants ont l'habitude de compter sur leurs doigts et on les en décourage souvent. Pourtant, tout notre système de représentation des nombres est basé sur cette manière simple de compter.

Énumérons les 10 chiffres dont nous disposons:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Nous pouvons, avec ces dix chiffres, dénombrer des objets: 3 pommes, 2 verres, 8 personnes... Les choses se compliquent néanmoins lorsque nous dénombrons 9 objets et qu'il est nécessaire d'en ajouter un: nous ne disposons pas d'un chiffre permettant d'exprimer la nouvelle quantité d'objets ! Ce codage pourrait paraître insuffisant, dès lors, mais heureusement, la représentation nous permet une petite astuce: en groupant les objets, nous comptons en fait que nous possédons maintenant une dizaine complète d'objets et aucun objet supplémentaire, que nous écrivons dans l'ordre: 10.

Quantité de dizaines	Quantité d'unités
1	0

Si on ajoute maintenant trois objets, nous écrivons que nous possédons une dizaine complète et trois objets: soit 13.

Quantité de dizaines	Quantité d'unités
1	3

Et si nous ajoutons maintenant 8 objets, nous pouvons créer une nouvelle dizaine et laisser un objet en plus, soit 21.

Quantité de dizaines	Quantité d'unités
2	1

Lorsqu'on atteint le nombre maximum de dizaines que l'on peut représenter, soit 9, on applique le même procédé aux dizaines: on groupe les dizaines en centaines et on écrit la quantité de centaines, puis de dizaines, puis d'unités... et ainsi de suite. Avec cette représentation, 10 chiffres suffisent à représenter n'importe quel nombre ![8]

Reprenons l'exemple précédent et inscrivons sous chaque chiffre le groupe qu'il dénombre:

5	4	6
100	10	1

Lisons: 5 centaines, 4 dizaines, 6 unités. Nous pouvons donc concevoir que ce nombre est issu de la somme de 5 x 100, 4 x 10 et 6 x 1.

Nous pouvons aussi noter tout cela en terme de puissances de dix:

5	4	6
10^2	10^1	10^0

Petit rappel: l'élévation de n'importe quel nombre à la puissance 0 donne 1.

Vous saviez tout cela ? Certainement... mais il est intéressant de conscientiser pleinement ces concepts car ils constituent la base de la représentation des nombres par les ordinateurs. Cette représentation des nombres à l'aide de 10 chiffre porte le nom de **système décimal**. Comme il utilise les puissances de 10, on dit qu'il utilise la **base 10**.

> La BASE d'un système numérique désigne la valeur dont les puissances successives interviennent dans la représentation d'un nombre dans ce système

8 Les Babyloniens utilisaient 60 chiffres pour représenter leurs nombres. Cela avait pas mal d'avantages, notamment lorsqu'il s'agissait de diviser: 60 est divisible par 1, 2, 3, 4, 5, 6,10,12,15, 20, 30 et 60, alors que 10 est divisible seulement par 1, 2, 5 et 10.

2.3.3 Le système positionnel avec deux chiffres: le système binaire

Revenons maintenant au codage de l'ordinateur: nous ne disposons maintenant que de deux chiffres, **0** et **1**. Il convient donc de construire un système de représentation similaire au système décimal, mais uniquement avec 2 chiffres.

Essayons d'abord de dénombrer un seul objet avec un chiffre: nous écrirons 1. Que se passe-t-il si nous ajoutons un objet ? Nous ne disposons plus d'un chiffre permettant de coder la nouvelle quantité, nous devons donc former un groupe. Ce groupe est constitué de deux objets, c'est donc une **paire.** À sa suite, il n'y a pas d'autre objet. Nous noterons donc 10 (prononcé un-zéro).

Quantité de paires	Quantité d'unités
1	0

Si on ajoute encore un objet, nous avons une paire et un objet: 11 (prononcé un-un).

Quantité de paires	Quantité d'unité
1	1

Ajoutons encore un objet: nous avons une nouvelle paire. Mais comme nous ne disposons pas d'un chiffre pour représenter la quantité de paires, nous devons grouper les paires par... paires ! Ainsi, pour représenter 4, nous noterons 1 paire de paires, aucune paire en plus et aucune unité en plus: 100.

Quantité de paires de paires	Quantité de paires	Quantité d'unités
1	0	0

Soyons fous, ajoutons encore un objet: nous avons une nouvelle unité, toute seule, à côté de la paire de paires: 101.

Quantité de paires de paires	Quantité de paires	Quantité d'unités
1	0	1

Et si l'on est assez dingue pour ajouter encore un objet, nous aurons une paire de paires, une paire et plus d'unité isolée: 110.

Quantité de paires de paires	Quantité de paires	Quantité d'unités
1	1	0

Et ainsi de suite... Le nombre 1101, écrit dans cette représentation, peut se lire:

- une paire de paires de paires (ça fait 8)
- une paire de paire (ça fait 4)
- pas de paire (donc rien)
- une unité (ça fait 1)

soit 8 + 4 + 1. 1101 exprime donc treize dans cette représentation.

On peut aussi s'amuser à écrire ce nombre et à placer sous chaque chiffre le groupe qu'il dénombre:

1	1	0	1
8	4	2	1

On peut aussi, comme précédemment, écrire les puissances qui entrent en jeu. Mais cette fois, il ne s'agit plus, bien sûr, de puissances de 10 mais de **puissances de 2**:

1	1	0	1
2^3	2^2	2^1	2^0

Merveilleux n'est-ce pas ? Peut-être entrevoyez-vous la portée de ce système de représentation: tout comme le système décimal, il permet de représenter n'importe quel nombre, mais cette fois uniquement avec 2 chiffres. Le **rang** de chaque chiffre désigne la puissance de 2 qu'il dénombre. Ce système de représentation s'appelle le **système binaire**. C'est grâce à lui qu'un ordinateur peut manipuler des nombres. Par commodité, on parle de **nombres binaires** lorsqu'on écrit ou lit un nombre codé dans le système binaire. Comme il utilise les puissances de 2, le système binaire utilise donc la **base 2**. Remarquez que 1201 est invalide. En effet, 2 ne fait pas partie des chiffres admissibles en base 2. Ce nombre n'est donc pas codable dans le système binaire. Afin de ne pas confondre (101 est-il trois ou cent et un ?), on peut **suffixer les nombres d'un indice** décimal exprimant sa base: 101_2 est un nombre binaire, alors que 101_{10} est un nombre décimal.

Voici les représentations binaires des quelques premiers nombres avec leurs équivalents décimaux:

Décimal	Binaire	Décimal	Binaire
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

Petit détail amusant: le système binaire dispose de deux chiffres pour représenter les nombres: 0 et 1 (vous le savez maintenant). 0 et 1 sont donc des **chiffres binaires**. En anglais, on traduit cela par *binary digit*. Par commodité (et probablement par fainéantise), les informaticiens ont contracté ces deux mots pour ne garder que les deux premières et la dernière lettre, ce qui donne le mot **bit**. Un chiffre binaire est donc un bit. Gageons que les anglophones n'ont certainement pas imaginé les conséquences hasardeuses de ce choix pour la langue française, qui dispose d'un mot très proche désignant une toute autre chose (salace, cela va sans dire). Insistons donc: le mot bit est masculin et ne comporte pas de E final.

Rappelons qu'en réalité, l'ordinateur n'a aucune idée de la valeur des nombres qu'il manipule: souvenez-vous que dans un ordinateur, la seule chose qui compte, c'est qu'un fil soit « chaud » ou « froid ». Mais grâce à la signification que nous attribuons à cette représentation, nous pouvons concevoir l'ordinateur pour qu'il manipule ces états d'une manière qui nous est utile.

2.4 Caractéristiques et conversions des nombres binaires

2.4.1 Convertir du binaire vers le décimal

Essayons de trouver le nombre décimal représenté par ce nombre binaire comportant 6 bits: 101011
Écrivons d'abord les puissances de 2 correspondant à chaque bit:

1	0	1	0	1	0
32	16	8	4	2	1

Ensuite, calculons la somme de toutes les puissances de 2 dont le bit correspondant vaut 1.
Calculons: 1x32+0x16+1x8+0x4+1x2+0x1, soit 42. Le nombre binaire 101010 s'écrit donc 42 en décimal[9].
Essayons maintenant avec 11010111.
Le nombre est plus long, il faut donc faire appel à des puissances de 2 plus élevées. Relisez encore une fois le tableau des puissances de 2 en page 9:

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

Pour construire la représentation décimale de ce nombre, additionnons toutes les puissances de 2 dont le bit correspondant vaut 1: 1x128+1x64+0x32+1x16+0x8+1x4+1x2+1x1, soit 215.

Pour construire la représentation décimale d'un nombre binaire, il faut:

- définir la puissance de 2 correspondant à chaque bit: noter 1 sous le bit le plus à droite, puis 2 sous son voisin de gauche, puis doubler ainsi successivement de bit en bit de droite à gauche
- calculer la somme de toutes les puissances de 2 dont le bit correspondant est 1

Notez qu'il existe d'autres méthodes pour représenter des valeurs à l'aide du codage binaire, comme par exemple le code Gray[10]. Son utilisation étant marginale, nous ne l'analyserons pas ici.

9 Selon le roman *Le Guide du Voyageur galactique* de D. Adams, des savants d'une race hyper-intelligente ont construit *Deep Thought*, le deuxième plus grand ordinateur de tous les temps, pour calculer la réponse à la *Grande Question sur la Vie, l'Univers et le Reste*. Au bout de 7 millions et demi d'années, l'ordinateur fournit la réponse « quarante-deux ». Une réponse jugée insignifiante, mais il semble, en fait, que ses concepteurs n'avaient pas très bien compris la question.

10 http://fr.wikipedia.org/wiki/Code_Gray

2.4.2 Quelques techniques futées pour convertir du binaire vers le décimal

Convertir un nombre binaire vers sa représentation décimale n'est donc pas très difficile pour peu qu'on connaisse les puissances de 2 et qu'on sache compter. Relisez encore une fois le tableau de la page 9 et refaites quelques additions pour vous entraîner.

Quelques cas particuliers nous permettent d'utiliser quelques raccourcis bien pratiques.

Facilement, on peut déduire qu'un nombre binaire constitué d'une quantité quelconque de bits à 0 vaut simplement... 0 !

> Un nombre binaire constitué uniquement d'une quantité quelconque de 0 vaut 0.

Observons ensuite le nombre binaire suivant:

1 0 0 0 0 0

Il s'agit d'un 1 suivi de 5 zéros. Si vous prenez le temps de noter les puissances de 2, vous remarquerez que le seul bit à 1 correspond à 32. Aucune autre puissance de 2 n'ayant de bit à 1, on peut déduire très facilement que ce nombre est 32 en décimal.

Comptez maintenant le nombre de zéros... Il y en a 5. Or, $32=2^5$. Cela fonctionne quelle que soit la quantité de zéros:

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

7 zéros, ce nombre est donc 2^7, soit 128.

> Pour déterminer la valeur d'un nombre binaire constitué d'un seul 1 suivi de N zéros, il suffit de calculer la valeur de 2^N.

Voyons maintenant un autre cas de figure:

1 1 1 1 1 1

Nous pouvons laborieusement placer les puissances de 2 sous chaque bit puis effectuer leur somme (tous les bits valent 1, il faut donc additionner toutes les puissances de 2). Cependant, un petit parallèle avec le système décimal va nous éclairer. Observez ce nombre décimal:

9 9 9 9 9 9

Il est constitué, pour chaque puissance de dix, du chiffre le plus élevé. Nous sommes donc en présence d'un nombre qui est très proche de la puissance de 10 suivante. En fait, si on y ajoute seulement 1, on passe à la puissance de 10 suivante: 999999+1=1000000. On peut encore écrire que 999999=1000000-1, ou que **$999999=10^6-1$**.

Cela tombe fort bien, 999999 est constitué de 6 chiffres 9, 6 fois le chiffre le plus élevé.

Si on généralise, on arrive à la conclusion suivante: lorsqu'un nombre décimal est constitué de N fois le chiffre 9, il vaut **10^N-1**.

Et bien, c'est pareil en binaire... enfin presque. En binaire, 1 est le chiffre le plus élevé, tout comme 9 l'est en décimal. Donc, quand un nombre binaire est constitué de N fois le chiffre 1, il vaut... attention, nous travaillons en base 2 ! Reprenons:

> Un nombre binaire constitué de N bits à 1 vaut **2^N-1**
> Corollaire: la plus grande valeur qu'on peut représenter avec un nombre binaire constitué de N bits est **2^N-1**

Fort de cette connaissance, vous pourrez déduire que le nombre binaire 111111, constitué de 6 bits à 1, vaut donc 2^6-1, soit 63. Relisez encore une fois le tableau en page 9 (ce tableau est VRAIMENT important... relisez-le encore une fois !).

Cette technique peut vous permettre de convertir bien plus vite certains nombres binaires vers leur équivalent décimal. Prenons 111101: il s'agit en fait de 111111 auquel le bit correspondant à 2 a été mis à 0. Il suffit donc de calculer 2^6-1 et de retrancher 2. 111101 est donc la représentation binaire de 61.

2.4.3 Un pré-requis: le décalage d'un nombre binaire

Nous avons vu comment passer de la base 2 (binaire) à la base 10 (décimal). Il nous suffit d'associer chaque bit à la puissance de 2 qu'il dénombre puis d'effectuer une simple somme. La conversion inverse est un peu plus compliquée, mais pas insurmontable pour peu qu'on connaisse certaines caractéristiques des nombres binaires qui vont nous être très utiles ici.

Tout d'abord, voyons ce qui se passe quand on ajoute ou ôte un bit à un nombre binaire:

Prenons un nombre binaire tout ce qu'il y a de plus classique:

1 1 0 1

Si vous avez suivi, vous pourrez certainement déterminer que ce nombre est la représentation binaire de 13. Essayons de mettre tout cela dans un petit tableau:

32	16	8	4	2	1
		1	1	0	1

C'est simplement une autre manière d'associer chaque bit à la puissance de 2 qu'il code. Que se passe-t-il si on décale tous les bits d'un rang vers la gauche ?

32	16	8	4	2	1
		1	1	0	1
	1	1	0	1	<--HOP

On ne peut évidemment pas laisser ce nombre dans cet état: il est incomplet car un bit est indéterminé. Complétons le tableau:

32	16	8	4	2	1
	1	1	0	1	0

Que voit-on ?

D'abord, observez que chaque bit correspond maintenant à une valeur **double** de celle qu'il désignait avant: le bit 8 est devenu 16, le 4 est devenu 8, le 2 est devenu 4 et le 1 est devenu 2. Or, en mathématiques, on peut appliquer le principe de factorisation: 2a+2b+2c+2d=2(a+b+c+d)

Donc, en décalant les bits d'un rang vers la gauche, on **double** la valeur du nombre binaire. Ce nombre là-dessus est la représentation binaire de 26 (si si, calculez). De même, si on décale de 2 rangs, on quadruple la valeur du nombre binaire, et ainsi de suite.

> Lorsqu'on décale un nombre binaire de N rangs vers la gauche, on multiplie sa valeur par 2^N
> Corollaire: pour multiplier la valeur d'un nombre binaire par 2^N, il suffit de le décaler de N bits vers la gauche.

Dit plus simplement, quand on « ajoute un zéro » à la droite d'un nombre binaire, on le double.

Et que se passe-t-il lorsqu'on décale d'un rang vers la droite ?

Reprenons nos petits tableaux, avec cette fois la représentation de 24 en binaire:

32	16	8	4	2	1
	1	1	0	0	0

Jusque-là, rien d'insurmontable. Décalons les bits vers la droite:

32	16	8	4	2	1
	HOP -->	1	1	0	0

Oups, nous avons perdu un zéro à droite du nombre. En effet, nous ne pouvons le conserver dans l'état de nos connaissances car on n'utilise pas les puissances de 2 inférieures à 0 dans la représentation des nombres que nous étudions[11]. Disons-lui adieu (provisoirement), et calculons la valeur du nombre binaire résultant: 8+4=12. Soit la **moitié** de la valeur du nombre original. Rien de surprenant: chaque bit (sauf le plus à droite) correspond maintenant à la moitié de la valeur qu'il représentait auparavant: le bit 16 est devenu 8, le 8 est devenu 4, le 4 est devenu 2, le 2 est devenu 1 et le 1... a disparu ! Or, toujours selon le principe de factorisation, on sait que ½ a + ½ b + ½ c + ½ d = ½ (a+b+c+d). On pouvait certainement « sentir » cela, car si, quand on décale d'un rang à gauche, on double, quand on décale d'un rang à droite, on divise de moitié.
C'est bien joli, mais que se passe-t-il si le bit le plus à droite est 1 ?
Prenons 25 en binaire:

32	16	8	4	2	1
	1	1	0	0	1

11 Rien n'interdit de noter des nombres binaires « à virgules », où les bits à droites de la virgule représentent des puissances négatives de 2, comme on le fait en décimal avec les puissances négatives de 10. Mais en pratique, cette représentation est très rarement utilisée et on préfère généralement d'autres représentations des nombres non entiers qui sont plus efficaces lorsqu'il s'agit de les manipuler.

Voyez-vous où le problème va se situer ? La moitié de 25 est 12,5, soit un nombre non entier. En binaire, cela risque de nous causer des ennuis. Essayons de décaler d'un rang vers la droite:

32	16	8	4	2	1
	HOP -->	1	1	0	0

Et vlan ! Le 1 à droite ayant disparu, le nombre binaire résultant vaut également 12. Est-ce normal ? Oui, dans la mesure où nous travaillons depuis le début sur des nombres **entiers**.

Nous manipulons déjà ce genre de souci avec efficacité en décimal. Divisez 865 par 8 en décimal, en utilisant la bonne vieille méthode de la potence:

```
 8 6 5 | 8
-8     |---
---    | 1 0 8
0 6 5  |
 - 0   |
  ---  |
   6 5 |
 - 6 4 |
   ----|
     1 |
```

Que faut-il lire ? Le quotient de 865 par 8 est 108, et il **reste** 1. Si l'on reprend le décalage du nombre binaire ci-dessus, on peut aussi affirmer qu'en décalant 11001 d'un rang vers la droite, on obtient 1100 et un **reste** qui vaut 1.

Puisqu'on travaille uniquement sur les puissances positives de 2, lorsqu'on décale un nombre binaire d'un rang vers la droite, on divise sa valeur de **moitié** mais on ne conserve que la **partie entière** du résultat et un **reste**. Les « moitiés », « quarts » et autres puissances négatives de 2 ne sont tout simplement pas prises en compte. Si l'on souhaite tout de même manipuler des valeurs non entières, il faut tenir compte de cet état de fait[12].

Lorsqu'on décale un nombre binaire de **N** rangs vers la droite, on divise sa valeur par $\mathbf{2^N}$.
Corollaire: pour diviser la valeur d'un nombre binaire par 2^N, il suffit de le décaler de N bits vers a droite.

2.4.4 Un autre pré-requis: la parité d'un nombre binaire

Vous rappelez-vous de ces petites phrases que vous avez dû apprendre en primaire ?

- La somme de deux nombres de même parité est paire.
- La somme de deux nombres de parités différentes est impaire.

Mais si, vous avez sûrement appris cela. Si vous ne vous en souveniez pas, c'est maintenant chose faite.

12 En fait, les ordinateurs ne sont pas aussi précis qu'on le croirait lorsqu'ils manipulent des nombres non entiers. Ce manque de précision est à l'origine d'un nombre incroyable de bugs dans de nombreux programmes trop peu testés et même dans certains microprocesseurs !

Par corollaire, on peut donc affirmer que

- La somme de deux nombres pairs est paire.
- La somme d'un nombre pair et d'un nombre impair est impaire.

A la lumière de ces connaissances pas si fraîches, observons un instant les puissances de 2 impliquées dans la constitution d'un nombre binaire: 1, 2, 4, 8, 16, 32, 64, ...

A partir de 2^1, chaque puissance est le double de la puissance inférieure... Donc, à partir de 2^1, toutes les puissances de 2 sont **paires**. La seule puissance de 2 qui soit impaire est 2^0, soit 1.

Cela implique qu'en fait, **seule 2^0 est en mesure de rendre un nombre binaire impair.** Et donc, pour déterminer la parité d'un nombre binaire, il suffit d'observer le bit le plus à droite, celui qui représente 2^0.

On peut déterminer la parité d'un nombre binaire en observant le bit correspondant à 2^0 (soit 1).

S'il est à 0, alors, **le nombre binaire est pair**. S'il est à 1, alors **le nombre binaire est impair.**

Ça marche à tous les coups. Il n'est même pas besoin de calculer la valeur du nombre binaire !

Ainsi, par exemple: 1110101101 est impair (non, pas besoin de calculer sa valeur, il suffit d'observer le bit le plus à droite), alors que 1110101100 est pair.

Si vous êtes un tantinet malin (mais il n'est pas permis d'en douter), vous pourrez sûrement déduire la propriété qui accompagne celle que nous venons de découvrir: on peut sans se tromper déterminer la valeur du bit qui correspond à 2^0 si on connaît la valeur du nombre binaire.

Lorsque la valeur d'un nombre binaire est paire, cela implique que le bit qui correspond à 2^0 est à 0. Sinon, lorsque la valeur est impaire, cela implique que le bit qui correspond à 2^0 est à 1.

Toujours avec un exemple: la représentation binaire de 42 se termine par un bit à 0, alors que celle de 43 se termine par un bit à 1. C'est aussi simple que cela !

Ceci va nous aider de manière considérable dans la conversion des nombres décimaux vers leur équivalent binaire.

2.4.5 Conversion du décimal vers le binaire

Essayons de trouver la représentation binaire du nombre décimal 75. Nous noterons les lettres de A à G pour désigner les bits dont nous ignorons la valeur.

64	32	16	8	4	2	1
A	B	C	D	E	F	G

En observant 75, il y a une chose que l'on peut instantanément déduire à propos de sa représentation binaire: 75 est un nombre **impair**, donc on sait que le **bit le plus à droite de sa représentation binaire (celui qui correspond à 1) est 1**. C'est un bon début, nous connaissons déjà un des bits !

Dans l'exemple ci-dessus, on peut affirmer que la valeur du bit G est 1.

Nous pouvons donc commencer à écrire la représentation binaire de 75:

\- - - - - - 1

Grâce à ce procédé, nous pouvons toujours déterminer la valeur du bit le plus à droite d'un nombre. Donc, si on parvient à « faire glisser » les bits de la représentation binaire de 75 d'**un rang vers la droite**, nous pourrons « observer » le bit correspondant à 2 (donc, le F) puisqu'il va alors occuper la place la plus à droite, celle du bit correspondant à 1. Nous savons comment faire glisser les bits d'un nombre d'un rang vers la droite: si **on divise la valeur d'un nombre par 2** en conservant la partie entière, **on décale du même coup les bits qui le constituent d'un rang vers la droite** (voir page 18).

Donc, divisions: 75 / 2 = 37,5. Gardons la partie entière: 37.

64	32	16	8	4	2	1
HOP -->	A	B	C	D	E	F

Dès ce moment, nous sommes en mesure de déterminer la valeur du bit le plus à droite, celui qui correspondait à 2 dans la représentation binaire de 75 (soit F). Ici, 37 étant un nombre impair, nous pouvons déterminer que le bit le plus à droite vaut 1.

Et donc, nous sommes en mesure d'affirmer que le bit F vaut 1.

Nous pouvons continuer à écrire la représentation binaire de 75:

\- - - - - 1 1

Voyez-vous la suite ? En continuant à **diviser successivement le nombre par 2** et en **observant la parité**, nous allons pouvoir passer en revue chacun des bits et déterminer s'il vaut 0 ou 1. C'est précisément ce que nous souhaitons faire: **déterminer la valeur de chaque bit**.

Divisons 37: 37 / 2 = 18,5. Gardons la partie entière: 18.

64	32	16	8	4	2	1
	HOP -->	A	B	C	D	E

18 étant pair, nous pouvons affirmer que le bit E vaut 0.

Écrivons ce que nous savons déjà de 75 en binaire:

- - - - 0 1 1

Continuons: 18 / 2 = 9.

64	32	16	8	4	2	1
		HOP -->	A	B	C	D

9 est impair, nous savons donc que le bit D vaut 1.
Écrivons:

- - - 1 0 1 1

Divisons 9: 9 / 2 = 4,5. Gardons la partie entière: 4.

64	32	16	8	4	2	1
			HOP -->	A	B	C

4 est pair, nous affirmons donc que C vaut 0.
Écrivons:

- - 0 1 0 1 1

Divisons 4: 4 / 2 = 2.

64	32	16	8	4	2	1
				HOP -->	A	B

2 est pair, nous sommes sûrs que B vaut 0.
Écrivons:

- 0 0 1 0 1 1

Plus qu'un, le suspens est à son comble !

Divisons 2: 2 / 2 = 1.

64	32	16	8	4	2	1
					HOP -->	A

1 est impair, A vaut donc 1.

Écrivons enfin:

1 0 0 1 0 1 1

Si on veut encore diviser 1, on obtient la partie entière 0. Dès ce moment, cela signifie que la conversion est terminée.

On peut vérifier que ce nombre binaire est bien la représentation de 75:

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

Calculons: 64+8+2+1 = 75. Ça tombe juste !

Cette méthode utilise des divisions successives, elle se nomme donc la **conversion par divisions successives** (c'est simple, non ?). Avez-vous remarqué qu'au fur et à mesure de la conversion, on écrit le nombre binaire **de droite à gauche** ?

Synthétisons tout cela:

> Conversion par divisions successives d'un nombre décimal en sa représentation binaire
>
> 1. Déterminer la parité du nombre décimal. S'il est pair, écrire 0 à gauche du nombre binaire, sinon écrire 1 à gauche du nombre binaire.
> 2. Diviser le nombre décimal par 2, conserver la partie entière. Si le résultat est 0, la conversion est terminée. Sinon, recommencer en 1.

Entraînez-vous à convertir des nombres décimaux en leurs équivalents binaires, c'est assez amusant.

Cette méthode a l'avantage d'être très facilement réalisée à l'aide d'un papier et d'un crayon. Néanmoins, il est parfois utile de réaliser la conversion mentalement. Dans ce cas, il faut bien s'entraîner et surtout, relire encore une fois le tableau des puissances de 2 de la page 9 car elles vont être très utiles pour la suite.

Prenons à nouveau 75. Quelle autre propriété peut-on utiliser pour déterminer la forme binaire de ce nombre ?

Rappelez-vous qu'en fin de compte, pour convertir un nombre binaire en décimal, il faut effectuer la **somme** de toutes les puissances de 2 dont le bit correspondant est à 1. Pourrait-on faire l'inverse et déterminer quelles sont les puissances de 2 qui participent à la somme ? Ou, en d'autres mots, peut-on déterminer quels sont les bits à 1 parmi ceux qui peuvent participer à la forme binaire du nombre ?

Pourquoi pas... Dans le cas de 75, on peut d'ores et déjà éliminer des puissances potentielles toutes celles qui dépassent 75. Donc, 128 et les puissances supérieures ne peuvent pas participer à la somme qui compose 75. C'est toujours ça de gagné...

Reste donc à déterminer, parmi toutes celles qui sont inférieures ou égales à 75, lesquelles sont impliquées.

Premier candidat: 64 ou 2^6. C'est la puissance de 2 la plus proche de 75 tout en étant inférieure ou égale.

Est-ce que 64 participe à la somme de puissances qui compose 75 ? Posons-nous la question à l'envers: est-il possible de trouver une somme de puissances de 2 hors 64 dont le résultat serait 75 ? Tout simplement: non.

En effet, rappelez-vous d'une des techniques futées vue à la page 16, où il était question de nombres binaires entièrement composés de bits à 1. Si l'on effectue la somme de toutes les puissances de 2 inférieures à 64, on ne peut obtenir que 63. C'est insuffisant. Donc, il est nécessaire de faire appel à 64 pour la représentation binaire de 75.

Partant de ce constat, nous pouvons écrire en toute confiance:

1 - - - - - -

Ce constat sera le même pour toutes les autres puissances de 2 inférieures à 64. Nous pouvons donc déduire que **la puissance de 2 la plus proche d'un nombre et qui lui est inférieure ou égale participe forcément à la somme de puissances qui le compose**.

Puisque nous avons déterminé que le bit représentant 64 participait à la forme binaire de 75, nous pouvons maintenant nous interroger sur le reste des puissances de 2.

Le cas de 64 étant réglé, nous avons maintenant à déterminer la valeur des 6 bits restants, dont aucun d'entre eux ne vaut 64. Nous ne pouvons, dès lors, poursuivre avec 75: nous devons en retrancher 64.

Effectuons 75 - 64 = 11

Est-ce que 32 peut participer à la somme qui compose 11 ? Certainement pas, il lui est supérieur ! Il est donc tout à faire évident que le bit correspondant à 32 ne peut être qu'à 0. Tant qu'on y est, le constat est pareil avec 16.

Nous pouvons donc écrire:

1 0 0 - - - -

Est-ce que 8 participe à la somme qui compose 11 ? Oui, c'est indéniable en vertu du principe déduit plus haut. Donc, nous pouvons écrire

1 0 0 1 - - -

Nous devons donc maintenant nous interroger sur les 3 derniers bits.

Effectuons 11 – 8 = 3

4 ne peut participer à la somme qui compose 3. Et un de plus !

1 0 0 1 0 - -

2 participe inévitablement à la somme qui compose 3:

1 0 0 1 0 1 -

Reste à effectuer 3 – 2 = 1.

1, c'est facile, c'est le bit le plus à droite ! Écrivons:

1 0 0 1 0 1 1

Ouf ! Le résultat est identique à celui de la méthode par divisions successives[13]. Cette méthode s'appelle la **conversion par soustractions successives**.

13 Il est heureux que les deux méthodes donnent des résultats identiques ! Sinon, toutes les mathématiques seraient à revoir...

Quelques remarques sur cette méthode:

- D'abord, observez que même si cela paraissait évident pour nombre d'entre elles, nous avons malgré tout **passé en revue l'ensemble des puissances de 2 inférieures ou égales au nombre décimal**, ceci afin de « n'oublier aucun 0 ». En effet, vous serez d'accord d'affirmer que 1111 et 1001011 sont bien différents ! Il est crucial que chaque puissance de 2 inférieure ou égale au nombre décimal soit « testée » afin d'écrire 0 ou 1.
- Ensuite, notez aussi que, dans cette méthode, le nombre binaire s'écrit **de gauche à droite**.
- Enfin, il est vrai qu'il est nécessaire de connaître les valeurs des puissances de 2 et d'effectuer des calculs de soustraction. Néanmoins, avec un peu d'entraînement, cette méthode permet de convertir mentalement des nombres décimaux vers leur représentation binaire plus rapidement qu'avec la méthode par divisions successives.

Synthétisons:

Conversion par soustractions successives d'un nombre décimal vers sa représentation binaire

1. Choisir la puissance de 2 inférieure ou égale la plus proche du nombre décimal

2. Si la puissance de 2 est inférieure au nombre décimal, noter 1 à la droite du nombre binaire et soustraire la puissance de 2 du nombre décimal. Sinon, noter 0 à la droite du nombre binaire.

3. Si la puissance de 2 est 2^0 (soit 1), la conversion est terminée. Sinon, choisir la puissance de 2 suivante dans l'ordre décroissant.

4. Recommencer en 2.

3 La manipulation des nombres binaires

Au fond, que peut faire l'ordinateur avec des nombres binaires ? A vrai dire, pas grand chose... mais si l'on conjugue une grande quantité de « pas grand chose », on obtient beaucoup de choses.

Jusqu'à présent, nous avons donc compris qu'un ordinateur compte, au mieux, jusque 1. Encore qu'il ne sache même pas ce que sont les nombres... Pour rappel, toute l'électronique qui anime l'ordinateur est basée sur la présence ou l'absence de courant: on note 1 pour un fil « chaud » et 0 pour un fil « froid ». Si l'on combine intelligemment les fils, on peut bâtir des circuits utiles.

Une des plus grandes avancées technologiques du XX^{e} siècle fut le *transistor*[14]. Il s'agit d'un petit composant électronique qui permet de contrôler les courants électriques. Aujourd'hui miniaturisés, ils se comptent par millions, voire milliards, dans les appareils que nous utilisons quotidiennement (dont l'ordinateur, cela va sans dire).

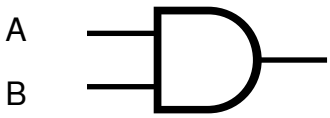
Un transistor joue en quelque sorte le rôle d'interrupteur électronique. Il accepte deux fils en entrée et un fil en sortie. La présence de courant sur le fil de sortie est conditionnée par la présence ou l'absence de courant sur les fils en entrée. Cela nous rappelle le schéma de la page 4 (traitement de l'information): ENTREE --> TRAITEMENT --> SORTIE.

Lorsqu'ils sont utilisés pour manipuler des informations binaires, on appelle souvent les transistors des **portes** ou *gates* en anglais. Ils sont en quelque sorte des portes que les courants peuvent ou non franchir en fonction de certaines conditions. Il existe très exactement **trois portes** de base qui, combinées, constituent l'ensemble des opérations possibles sur les nombres binaires. Dans la pratique, on les désigne souvent par leur nom anglais.

3.1 La porte ET (AND gate)

Une porte ET ne permet un courant de sortie que lorsque les deux fils en entrées portent eux-mêmes des courants. Donc, pour que le fil de sortie soit « chaud », il faut que chacun des fils en entrée soit « chaud ». Si ne fut-ce qu'un seul des deux fils en entrée est « froid », le fil de sortie est « froid ». Il faut donc que que le premier fil soit chaud ET que le deuxième soit chaud aussi pour que le fil de sortie soit chaud, d'où le nom donné à la porte.

Sa représentation dans les schémas est la suivante:



On voit les deux fils en entrée (notés A et B) et le fil en sortie. Puisqu'avec deux fils, on peut créer 4 états différents, nous pouvons dresser un petit tableau reprenant les différents états possibles en entrée afin de déterminer comment évolue l'état de sortie. Reprenons pour cela le représentation étudiée précédemment: un fil chaud est noté 1 et un fil froid est noté 0. Si le dispositif physique s'appelle une porte, sa représentation logique se nomme **fonction**, tout comme en algèbre. La fonction AND acceptant deux opérandes est notée A AND B (version longue), A $\wedge$ B (avec un opérateur logique) ou A.B (avec un opérateur algébrique). L'opérateur AND est commutatif et associatif.

14 De même que la glace au spéculoos, évolution majeure dans la technologie des papilles gustatives.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Ce genre de tableau est nommé **table de vérité** en référence à une représentation qui veut que 1 signifie « vrai » et 0 signifie « faux ».

Comment lire une table de vérité ? En fait, il faut lire par lignes. Par exemple, si on prend la dernière ligne de ce tableau, on peut lire: « lorsque A est 1 et lorsque B est 1, alors A AND B est 1 ». On peut aussi écrire la liste de tous les états possibles sous forme de petites égalités:

- 0 AND 0 = 0
- 0 AND 1 = 0
- 1 AND 0 = 0
- 1 AND 1 = 1

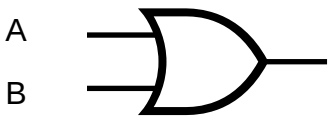
Cela nous amène à déduire que **A AND A vaut A** (cas 0 AND 0 et 1 AND 1), et que **A AND NOT A vaut 0** (cas 0 AND 1 et 1 AND 0). Et aussi cette phrase, que vous devriez essayer de retenir: « dès qu'un 0 se trouve en entrée d'une fonction AND, alors le résultat vaut forcément 0 ».

3.2 La porte OU (OR gate)

Une porte OU ne permet un courant en sortie que lorsqu'un des deux fils d'entrée au moins porte un courant. Donc, lorsqu'au moins un fil en entrée est chaud, le fil en sortie est chaud. Si les deux fils en entrée sont froids, le fil en sortie est froid. Si les deux fils en entrée sont chauds, le fil en sortie est chaud.

Il faut donc que le premier fil soit chaud OU que le deuxième fil soit chaud pour que le fil en sortie soit chaud, d'où le nom donné à la porte. Attention, ce OU est à prendre avec un sens supplémentaire: « c'est **l'un** ou **l'autre** ou **les deux** ».

Sa représentation dans les schémas est la suivante:



Comme pour la porte ET, il existe la fonction associée. Nous pouvons dresser la **table de vérité** reprenant les états possibles en fonction des opérandes A et B. Le résultat e la fonction OU est noté A OR B (version longue), A $\vee$ B (avec un opérateur logique) ou A+B (avec un opérateur algébrique). L'opérateur OR est commutatif et associatif.

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

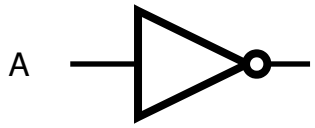
On peut aussi écrire la liste de tous les états possibles sous forme de petites égalités:

- 0 OR 0 = 0
- 0 OR 1 = 1
- 1 OR 0 = 1
- 1 OR 1 = 1

Cela nous amène à considérer aussi que **A OR A vaut A** (cas 0 OR 0 et 1 OR 1) et que **A OR NOT A vaut 1** (cas 0 OR 1 et 1 OR 0). N'oubliez pas cette phrase: « dès qu'un 1 est présent en entrée d'une fonction OU, alors le résultat vaut forcément 1 ».

3.3 La porte NON (NOT gate)

Dernière des portes de base, la porte NON se distingue par son côté *unaire*. En effet, elle n'accepte qu'un seul fil en entrée[15], dont elle va « inverser » la valeur: s'il est chaud, le fil de sortie sera froid; s'il est froid, le fil de sortie sera chaud. Sa représentation dans les schémas est la suivante:



Comme pour les portes ET et OU, il existe une fonction associée dont on peut dresser la **table de vérité** reprenant les états possibles. Le résultat d'une fonction NON est noté NOT A (version longue), ¬A (avec un opérateur logique) ou $\overline{A}$ (avec un opérateur algébrique).

A	NOT A
0	1
1	0

On peut aussi écrire la liste de tous les états possibles sous forme de petites égalités:

- NOT 0 = 1
- NOT 1 = 0

15 En pratique, quand on implémente une porte NOT via un transistor, il y a quand même deux fils en entrée dont l'un ne participe pas au traitement de donnée: il ne sert qu'à alimenter le circuit.

3.4 Combinaisons de portes, fonctions logiques

Le croirez-vous ? A partir des trois portes AND, OR et NOT, il est possible d'effectuer tous les calculs logiques et arithmétiques existants.

On peut évidemment placer plusieurs portes les unes derrières les autres, de manière que la sortie de l'une d'elle devienne une entrée de la suivante.

En combinant intelligemment les fonctions logiques de base, on peut concevoir de nouvelles **fonctions logiques**. Par exemple, on peut définir la fonction **XOR (OU exclusif)**, dont le fonctionnement peut être décrit comme suit: « l'un ou l'autre, mais pas les deux ». Plus sérieusement, cela signifie que le fil de sortie est chaud si un et un seul des deux fils d'entrée est chaud.

La représentation de la porte associée dans un schéma est la suivante:

Le résultat de la fonction XOR est noté A XOR B (version longue), A ⊕ B (avec un opérateur logique). L'opérateur XOR est commutatif et associatif.
Si l'on trace la table de vérité de l'opérateur XOR, on remarque qu'elle est assez semblable à celle de l'opérateur OR, à une ligne près:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

La dernière ligne, celle qui voit A et B à 1, change tout. Si l'on écrit les états possibles, nous obtenons:

- 0 XOR 0 = 0
- 0 XOR 1 = 1
- 1 XOR 0 = 1
- 1 XOR 1 = 0

L'opérateur XOR peut être décrit très facilement à partir des opérateurs de base AND, OR et NOT. Cela peut faire l'objet d'un exercice...[16]

Quelques remarques intéressantes: observez bien la table de vérité de la fonction XOR et vous remarquerez que A XOR B ne vaut **1** que lorsque **A et B sont dans un état différent** !
Donc, on peut en déduire que **A XOR A vaut 0** (cas 0 XOR 0 et 1 XOR 1) alors que **A XOR NOT A vaut 1** (cas 0 XOR 1 et 1 XOR 0), ou encore que **A XOR 1 vaut NOT A** (cas 0 XOR 1 et 1 XOR 1) et **A XOR 0 vaut A** (cas 0 XOR 0 et 1 XOR 0).

16 C'est un exercice amusant. Essayez simplement de traduire la phrase en français: A ET PAS B, OU B ET PAS A. C'est bien plus simple qu'on ne pourrait l'imaginer...

Si l'on est assez malin pour placer une fonction NOT à la sortie du XOR, on peut facilement créer une fonction qui vaut 1 lorsque ses deux entrées sont **dans le même état**. Il est alors facile de créer un petit circuit qui détermine si deux fils sont dans le même état ! Ce nouvel opérateur est noté EQU et se nomme simplement « équivalence ». Il est noté A EQU B (version longue), A ⇔ B (avec un opérateur logique) ou A ≡ B (avec un opérateur arithmétique).
Peut-être pouvez-vous tracer vous-mêmes sa table de vérité ?...

3.5 Fonctions arithmétiques

3.5.1 L'addition

Vous rappelez-vous comment effectuer une addition de deux nombres avec du papier et un crayon ? Il faut additionner les chiffres par colonne, écrire ensuite le chiffre des unités et reporter le chiffres des dizaines...

```
    1   1
    3   2   5
+   1   7   7
-------------
    5   0   2
```

Après avoir effectué 5+7, on écrit 2 et on reporte 1. Puis, on effectue 1+2+7, on écrit 0 et on reporte 1. Et enfin, on effectue 1+3+1. Facile non ?

Et bien, en binaire, cela se passe exactement de la même manière ! Pour rappel, en binaire, 2 s'écrit 10 et 3 s'écrit 11. Donc:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 1 0
- 1 + 1 + 1 = 1 1

Voyons cette petite addition:

```
  1   1   1   1
      1   0   1   1
+     1   1   0   1
    ---------------
  1   1   0   0   0
```

En décimal, nous aurions effectué ici 11 + 13 et nous aurions pu constater que la réponse est 24.

Comme dans le cas d'une addition en décimal, il faut calculer la somme des chiffres en colonnes, écrire le bit unité et reporter le bit de double. Mais si l'on prend la peine d'examiner finement l'addition de deux bits (nous verrons ensuite comment propager cela à plus de deux bits), nous pouvons observer qu'il existe une corrélation claire entre une addition et certains opérateurs logiques.

Prenons les quatre additions qu'il est possible d'effectuer avec deux bits:

- 0 + 0 = 0 0
- 0 + 1 = 0 1
- 1 + 0 = 0 1
- 1 + 1 = 1 0

Mettons tout cela dans un tableau: appelons A et B les bits que nous souhaitons additionner, S la somme et R le report. C'est donc ici la **table de vérité** de l'addition:

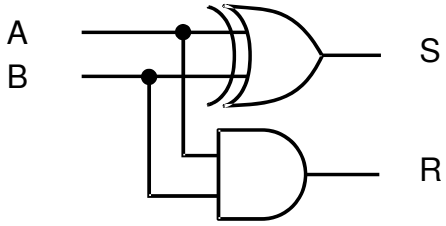
A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Comment lire ce tableau ? Très simplement: la somme de A et B donne S avec un report R. Donc, par exemple, pour la troisième ligne « 1 + 0 = 1, avec un report de 0 » ou encore pour la quatrième ligne « 1 + 1 = 0, avec un report de 1 ».

Prenez la peine de comparer ce tableau avec les tables de vérité des opérateurs AND (page 27) et XOR (page 29). Voyez-vous combien le calcul de S ressemble à un XOR et combien le calcul de R ressemble à un AND ? Bingo ! Nous pouvons donc **effectuer une somme à l'aide d'un XOR et d'un AND** !

Ce petit circuit simple permettant d'additionner deux bits se nomme **semi additionneur** et constitue la base des capacités de calcul des ordinateurs.

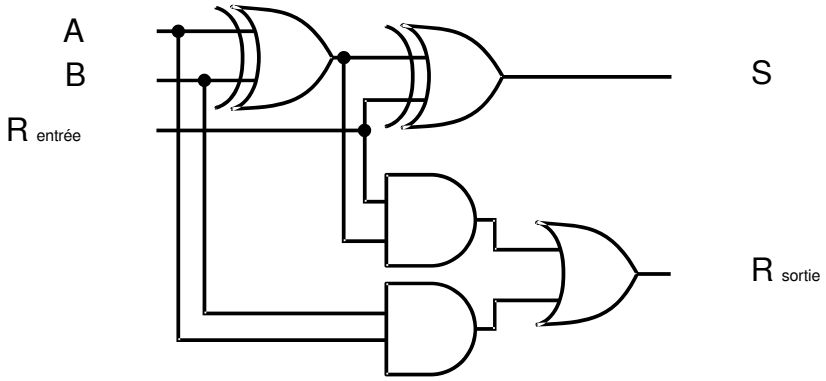
Sa représentation en schéma électronique est la suivante:



En entrée: A et B. En sortie, S (la somme) et R (le report ou **retenue**).

Petite remarque: en anglais, retenue se dit *carry* et il est très courant d'observer C au lieu de R sur ce schéma. Et toujours en anglais, semi additionneur se dit *half adder*.

Ce petit circuit est capable d'additionner deux nombres binaires. On peut en chaîner plusieurs pour additionner en parallèle plus de deux bits, constituant ainsi un **additionneur** complet. En effet, dans le calcul précédent, nous sommes parfois amenés à additionner trois bits (les deux bits à additionner et une retenue éventuelle). Il y a une porte en plus, la voyez-vous ?



Et enfin, en disposant intelligemment plusieurs additionneurs les uns à la suite des autres, on peut créer un circuit permettant l'addition de nombres binaires constitués de nombreux bits.

3.5.2 La soustraction

La soustraction est un peu plus difficile à réaliser que l'addition. En effet, il est nécessaire, pour effectuer une soustraction, de réaliser des comparaisons du genre « est-ce que 8 est plus grand que 2 ? Dans ce cas, je dois emprunter 1 au rang suivant »...

Afin de contourner ce problème, on peut aussi définir la soustraction comme l'**addition d'un opposé**. Ainsi, écrire 143 - 23 ou 143 + (-23) donne le même résultat. Néanmoins, nous ne savons pas comment représenter des nombres négatifs en binaire[17]. Les informaticiens savent tout cela, aussi ont-ils décidé de ruser un peu...

Revenons un instant dans notre bon vieux système décimal et voyons cette technique originale:

On vous demande d'effectuer 753 – 485. Vous pouvez bien entendu placer les nombres l'un au dessus de l'autre et effectuer une soustraction écrite comme on vous l'a appris lors de vos études primaires. Mais essayons une autre technique... Attention, suivez bien !

17 Il est amusant de constater combien il est facile de manipuler des nombres décimaux totalement inexistants dans la nature. Essayez de tenir -4 pommes dans vos mains pour voir...

D'abord, vous serez probablement d'accord pour affirmer que

753 – 485 = 753 – 485 + 1000 – 1000

ou encore que

753 – 485 = 753 – 485 + 999 + 1 – 1000

Intéressons-nous particulièrement à un groupe du deuxième terme de cette égalité:

753 – 485 = 753 **– 485 + 999** + 1 – 1000

Calculons simplement 999 – 485 = 514. Ce calcul est très facile: il suffit de déterminer le **complément** de 485, donc le nombre qui, additionné à 485, donne 999. Ou encore, de déterminer, pour chaque chiffre de 485 ce qui manque pour obtenir 999... Donc:

- à 4, il faut ajouter 5 pour obtenir 9
- à 8, il faut ajouter 1 pour obtenir 9
- à 5, il faut ajouter 4 pour obtenir 9

En prenant chaque chiffre individuellement, le calcul du complément est un jeu d'enfant.

Nous disposons maintenant de 514, qui remplace avantageusement -485+999 dans le calcul ci-dessus qui devient donc

753 – 485 = 753 **+ 514** + 1 – 1000

Nous avons donc transformé une soustraction ennuyeuse par une addition bien plus facile à réaliser !

```
    7  5  3
+   5  1  4
+        1
   ----------
1  2  6  8
```

Reste donc à soustraire 1000, ou encore à « laisser tomber le 1 en trop », ne conserver que les 3 chiffres significatifs, d'où 753 – 485 = **268**.

Cette méthode, appliquée au binaire, va nous permettre d'utiliser une addition (donc un circuit additionneur) pour réaliser une soustraction.

Tout d'abord, comment calculer un complément en binaire ? Et bien, exactement comme en décimal... ou presque. Prenons un exemple avec deux nombres binaires: 1101 et 0110. Remarquez tout d'abord que ces deux nombres contiennent le même nombre de bits, on dit qu'ils sont de **même longueur**. Ceci est absolument nécessaire, même s'il est vrai que le premier 0 du deuxième nombre n'est pas significatif.

Donc, si nous fonctionnons comme nous l'avons fait en décimal, nous pouvons affirmer que

1101 – 0110 = 1101 – 0110 + 10000 – 10000

ou encore que

1101 – 0110 = 1101 – 0110 + 1111 + 1 – 10000

Comme pour le décimal, isolons le groupe intéressant du deuxième terme de l'égalité:

-0110 + 1111

Il s'agit donc ici de calculer le complément de 0110, soit le nombre qu'il faut ajouter à 0110 pour obtenir 1111. C'est très facile: 1001.

Remarquez déjà que 0110 et 1001 ont un air de famille: **l'un est le résultat d'une fonction NOT appliquée à l'autre**. Autrement dit, on « remplace les 0 par des 1 et les 1 par des 0 ».

Le calcul devient donc:

1101 – 0110 = 1101 **+ 1001** + 1 – 10000

C'est une addition tout à fait classique qu'un circuit additionneur est capable de réaliser (et vous aussi d'ailleurs):

```
    1  1  0  1
+   1  0  0  1
+   0  0  0  1
   ---------------
1   0  1  1  1
```

Il faut ensuite soustraire 10000, autrement dit « laisser tomber le bit à 1 à gauche ». Le résultat nous apparaît alors: 1101 – 0110 = 0111.

Est-ce correct ? Voyons en décimal: 13 – 6 = 7. Oui, on dirait bien que ça tombe juste !

Cette technique est utilisée par la majorité des ordinateurs actuels pour manipuler les nombres négatifs. Elle consiste donc à **inverser les bits du nombre et à y ajouter 1**.

Simplement inverser les bits permet de calculer le **complément à 1**. Lorsqu'on ajoute encore 1, on obtient alors le **complément à 2**.

Notez encore qu'il ne s'agit en fait que d'une **représentation** d'un nombre négatif, intimement liée à une **convention**. Dans le cas précédent, 1010 représente -6, mais il pourrait tout aussi bien représenter 10. Comment savoir ? Et bien, si l'on ne dispose que du nombre binaire sans autre forme d'explication, on ne peut pas le déterminer... Cela peut amener à d'étranges résultats lors de certaines opérations arithmétiques, selon que le programme prenne en compte les compléments à 2 ou pas[18].

18 Il y a quelques années, il était possible d'encoder un nombre négatif comme montant d'un virement bancaire, simplement en entrant un grand nombre positif qui était mal interprété par le programme de gestion des comptes d'une grande banque (non, je ne vous dirai pas laquelle). Cela pouvait permettre de s'enrichir facilement, mais en laissant néanmoins une trace gênante.

Pour soustraire B de A en binaire, il faut:
- s'assurer que les deux nombres sont de même longueur et ajouter des 0 non significatifs si ce n'est pas le cas
- calculer le complément à 1 de B, soit inverser tous les bits qui le constituent
- ajouter 1 au complément à 1 de B pour obtenir son complément à 2
- additionner le complément à 2 de B à A
- conserver uniquement les bits significatifs, c'est à dire les bits correspondant à la longueur du plus long nombre parmi A ou B.

4 Conventions pour les nombres binaires

Un nombre binaire n'exprime, en fin de compte, que l'état d'un certain nombre de fils. Cela signifie donc que, par défaut, un ordinateur est capable par ce biais de manipuler des nombres compris entre certaines plages, correspondant au nombre de fils que les circuits sont capables de traiter simultanément en un temps donné.

Les ordinateurs ont donc été conçus pour les manipuler par paquets afin de définir clairement les limites des traitements applicables. On parle ainsi d'ordinateurs « 8 bits », « 16 bits », « 32 bits », « 64 bits »... Il existe aussi de petits circuits 4 bits.

Le croirez-vous ? La majorité des microprocesseurs vendus actuellement dans le monde sont des circuits 8 bits, donc capables d'effectuer des opérations de base sur des nombres compris entre 0 et 255. En effet, de nombreuses applications embarquées ne nécessitent pas de puissance de calcul énorme et se contentent très bien de ces circuits. C'est le cas, notamment, des puces contrôlant les moteurs de voitures, les feux de signalisation, les robots industriels, les calculatrices de poche, l'électroménager et même certains téléphones mobiles.

Un groupement de 8 bits est appelé un **octet** et il constitue l'unité de base de toute l'informatique moderne lorsqu'il s'agit de mesurer les capacités de stockage des mémoires. Remarquez qu'on y trouve la racine oct- , comme dans *octogone*, qui signifie huit.

Deux octets (soient 16 bits) constituent un **mot** et deux mots (soient 4 octets ou 32 bits) constituent un **double mot**. Ces deux terminologies sont plus marginalement utilisées et en pratique, on dit plus souvent « 16 bits » que « un mot ».

Il est à noter qu'il existe en anglais le mot *byte* (à ne pas confondre avec bit) qui désigne presque toujours un octet. Cependant, byte peut parfois désigner d'autres quantités de bits selon le contexte. Il convient donc d'être prudent lors de son utilisation.