“

L'arithmétique, c'est être capable de compter jusqu'à vingt sans enlever ses chaussures.

”

Walt disney

REPRESENTATION DES DONNEES :

Types et valeurs de base (1)

- *Numération*
- *Représentation d’un entier positif dans une base*
- *Représentation binaire d’un entier relatif*

Numérique et Sciences Informatiques
1ère

Support de cours :
Jean-Christophe BONNEFOY

Objectifs :

- Passer de la représentation d’une base dans une autre :
- Maitriser la représentation binaire d’un entier relatif :
  - Utiliser le complément à 2
  - Evaluer le nombre de bits nécessaires à sa représentation
- Savoir additionner et multiplier en binaire
  - Evaluer le nombre de bits nécessaires à la somme et au produit

1. Bases de numération :

Les bases de calcul sont nombreuses même si peu sont réellement utilisées. On peut citer un certain nombre de bases qui ont eu ou qui ont toujours une très grande importance :

- La **base 2** ou **base binaire** très utilisée en A.I.I. (Automatisme et Informatique Industrielle) est la base de la logique booléenne ou algèbre de Boole. Cette base utilise le lien avec la physique : 0 représente un interrupteur ouvert et 1 représente un interrupteur fermé.

- la **base 4.** Cette base a été l'objet de nombreuses polémiques et aurait pu devenir la base universelle.

- la **base 8** (base octale). Utilisée il y a un certain temps en informatique lorsque les machines utilisées étaient peu gourmandes en puissance de bus.

- la **base 10** (base décimale). Elle est considérée comme la base universelle. De ce fait nous l'utilisons presque tout le temps.

- la **base 12** qui est une base religieuse établie sur les douze signes du zodiaque. Cette base a été utilisée principalement dans le commerce (c'est à cause de l'utilisation de cette base que l'on parle encore d'une douzaine d'œufs ou d'une douzaine d'huîtres, dans une journée il y a 24 heures divisées en deux fois douze heures, etc.)

- la **base 16** (base hexadécimale). Cette base est très utilisée dans le monde de la micro-informatique et des automates. Elle permet de coder un mot (16 bits) sur 4 variables hexadécimales. Cette base fait intervenir tous les chiffres de la base 10 complétée par les 6 premières lettres de l'alphabet.

- la **base 20** cette base a été construite à partir des dix doigts des mains et des pieds de l'homme. Cette base a été très utilisée et il en subsiste quelques traces dans notre numération actuelle (quatre-vingts, quatre-vingt-dix).

-la **base 60** cette base a été construite sur des concepts religieux, d'un maniement complexe, elle est toujours utilisée de nos jours. (Sur un cercle, il y a 360° qui sont divisés chacun en 60 minutes divisées chacune en 60 secondes. Il en va de même pour chacune des heures de la journée qui sont divisées chacune en 60 minutes, elles-mêmes divisées en 60 secondes. La division des secondes en soixantième de seconde n'ayant pas de sens la précision se fait en centième de seconde. Il subsiste quelques traces de cette base dans notre numération actuelle (soixante, soixante-dix).

Codage en base 2 ☞ 0 1.

Codage en base 8 ☞ 0 1 2 3 4 5 6 7.

Codage en base 10 ☞ 0 1 2 3 4 5 6 7 8 9.

Codage en base 16 ☞ 0 1 2 3 4 5 6 7 8 9 A B C D E F.

Attention !!! Le symbole (ou supérieur) de la base ne peut être un symbole utilisé dans la dite base.
Exemple : il n'y a pas de 8 ni de 9 dans la base 8

2. Relation entre les bases :

2.1 Compter en base 10

En base 10 on compte en décomposant en unité (10^0), dizaine (10^1), centaine (10^2), milliers (10^3), etc. On obtient ainsi l'écriture d'un nombre entier positif en base 10.

Exemple :

$$1345 = 1 * 1000 + 3 * 100 + 4 * 10 + 5 * 1 = 1 * 10^3 + 3 * 10^2 + 4 * 10^1 + 5 * 10^0$$

Pourquoi fait-on des paquets de 10^0, 10^1, 10^2, 10^3 etc **?**

Parce qu'on a que dix symboles à disposition : les chiffres de 0 **à** 9 donc on fait des paquets de puissances de dix. En effet, si on cherche par exemple le nombre de combinaisons possibles obtenues avec 3 chiffres, on a 10 possibilités pour chacun des 3 chiffres, soit 10×10×10=103=1 000 combinaisons possibles, de 0 à 999. Donc au-delà de 999 il faut … nécessairement rajouter un quatrième chiffre : le chiffre des milliers !

2.2 Formule de transformation d'une base B vers la base 10

On considère un nombre entier positif X écrit en base b

$(X)_b = (a_n \quad a_{n-1} \quad \ldots\ldots \quad a_1 \quad a_0)_b$ pour tout nombre entier $a_0 \ldots . a_n$ compris entre 0 et b-1

Alors il s'écrit en base 10

$$(X)_{10} = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b^1 + a_0 b^0$$

Avec $b^0 = 1$

Exemples : Convertir les nombres suivants dans la base décimale

$(534)_8 = 5*8^2 + 3*8^1 + 4*8^0 = 348$

$(5A4)_{12} = 5*12^2 + 10*12^1 + 4*12^0 = 844$

$(113)_4 = 1*4^2 + 1*4^1 + 3*4^0 = 23$

$(ABB)_{16} = 10*16^2 + 11*16^1 + 11*16^0 = 2747$

$(5A4)_{11} = 5*11^2 + 10*11^1 + 4*11^0 = 719$

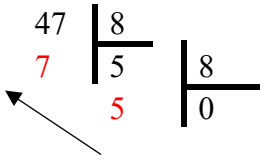
$(100110)_2 = 1*2^5 + 1*2^2 + 1*2^1 = 38$

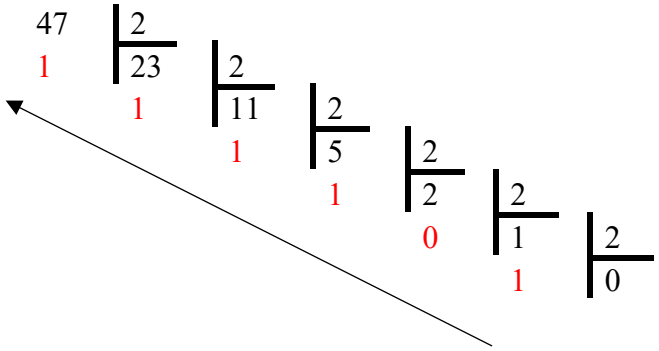
2.3 Formule de transformation d'une base 10 vers la base B

Méthode de la division euclidienne : Il faut prendre le nombre en base 10 et faire des divisions successives par la base B jusqu'à obtenir un quotient nul. Il suffit alors de récupérer les restes du bas vers le haut pour former le nombre en base B.

Exemple :

Exprimons $(47)_{10}$ dans le système octal et dans le système binaire. Nous obtenons:



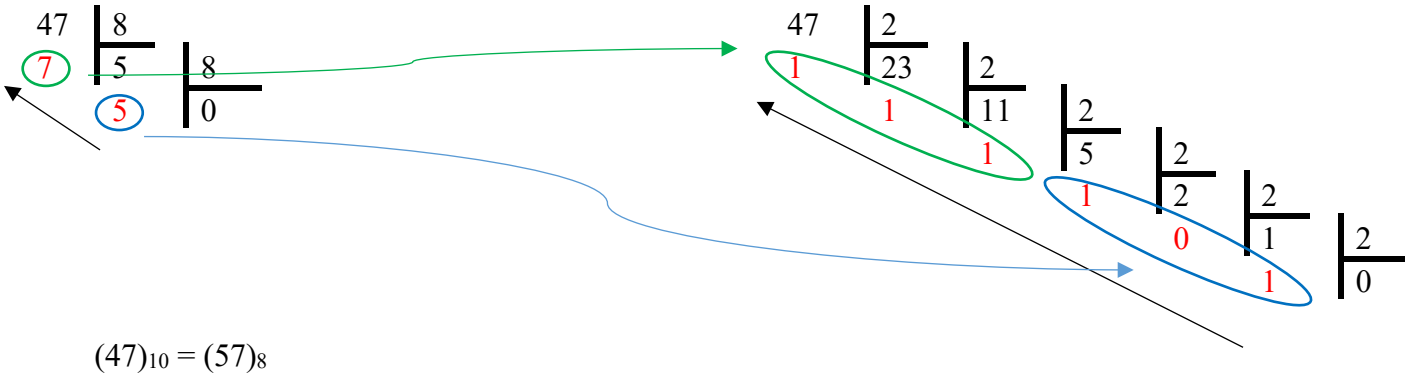


$(47)_{10} = (57)_8$
$(47)_{10} = (101111)_2$
Donc $(57)_8 = (101111)_2$

En base 2, les divisions sont assez évidentes pour les 16 premiers nombres décimaux (de 0 à 15). On peut déjà apprendre le tableau de correspondance suivant. Il va nous servir très vite

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

Dans l'exemple précédent, nous pouvons remarquer qu'après 3 divisions en binaire nous avons le même quotient qu'après une seule en octal. De plus le premier reste en octal obtenu peut être mis en relation directement avec les trois premiers restes en binaire:

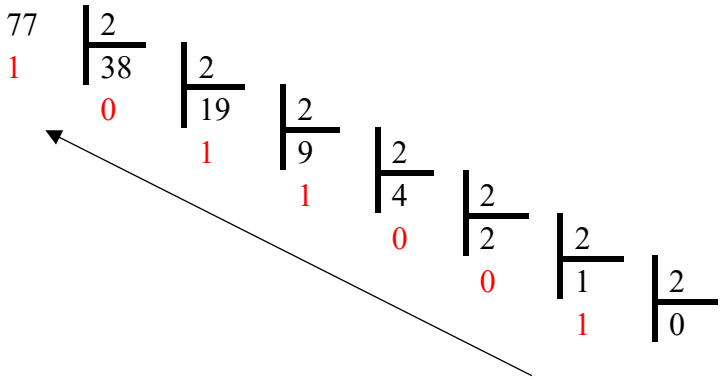


$(47)_{10} = (57)_8$
$(47)_{10} = (101111)_2$
Donc $(57)_8 = (\underline{101}\ \underline{111})_2$

Cette propriété d'équivalence entre chaque chiffre octal et chaque groupe de 3 chiffres binaires permet de passer facilement d'un système à base 8 à un système à base 2 et vice versa.

$8 = 2^3$ donc on réalise des groupements de 3 symboles binaires !

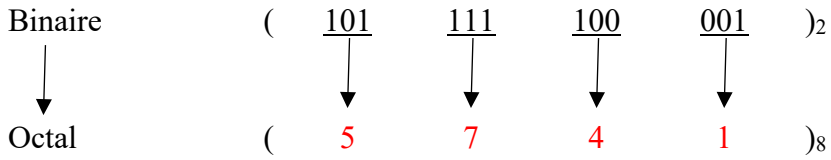
Exemple :
Convertir le nombre 77 (exprimé en base décimale) en base 2, puis en déduire la conversion en base 8

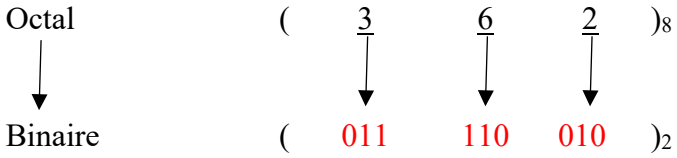


$(77)_{10} = (1001101)_2$
$(77)_{10} = (\underline{001}\ \underline{001}\ \underline{101})_2 = (1\ 1\ 5)_8$

2.4 Transformation d'une base 2^n vers la base B

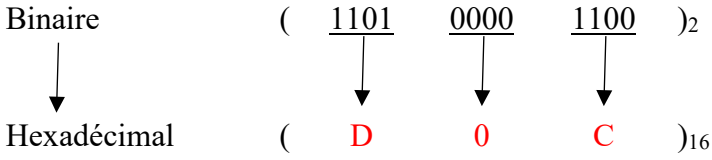
Exemple de conversion binaire octal et octal binaire :

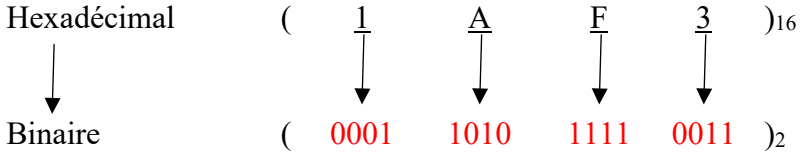




Relation entre les nombres binaires et les nombres hexadécimaux:

La propriété d'équivalence que nous venons de voir entre le binaire et l'octal existe aussi entre l'hexadécimal et le binaire. **16 = 2^4 donc on réalise des groupements de 4 symboles binaires !**





3. Définitions :

☞ **Bit :** Contraction des mots anglais BInary digiT. C'est l'unité élémentaire d'information qui ne peut prendre qu'une des deux valeurs suivantes : 0 ou 1. On aura tendance en algorithmie à faire l'analogie avec le type booléen : 0 : FAUX (FALSE) et 1 : VRAI (TRUE)

☞ **Mot :** C'est un ensemble de plusieurs bits.

☞ **Octet (byte):** C'est un mot de huit bits. On exprime généralement la capacité des mémoires en Kilo-octets (Ko), un Ko vaut 1024 octets (2^{10}).
Les disques durs peuvent stocker plusieurs dizaines, voire plusieurs Giga-octets de mémoire de masse. On exprime généralement la capacité des mémoires de masse en Mega-octets (Mo) et Giga-octets (Go). Un Mo vaut 1024 Ko. Un Giga-octet vaut 1024 Mo.

Attention donc à ne pas confondre un bit et un byte, un byte c'est 8 bits.

4. Codage binaire (appelé également code binaire pur) :

On associe à chaque bit un poids. Ce poids est fonction de la position du bit dans l'octet.

☞ **LSB** :(Least significant Bit) bit de poids faible, c'est le bit qui a le moins de signification dans un octet. Par convention, c'est le bit le plus à droite dans l'écriture d'un mot.

☞ **MSB** :(Most significant bit) bit de poids fort, c'est le bit qui a le plus de signification dans un octet. Par convention, c'est le bit le plus à gauche dans l'écriture d'un mot.

Explication par l'exemple : l'octet 1010 1101 représente le nombre 173 en base 10.

Décomposition :

Valeur décimale à décomposer en fonction du poids de chaque colonne : 173							
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	1	0	1

MSB LSB

En effet, on a

$$173 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$(173)_{10} = (10101101)_2$$

Avec n bits, on représente des nombres entiers positifs compris entre 0 et 2^n-1

Exemple : Avec 2 bits

B1	B0	nombre
0	0	0
0	1	1
1	0	2
1	1	3

On représente bien les nombres entiers compris entre 0 et 3 (2^2-1)

5. Représentation des nombres entiers relatifs en binaire

5.1 Codage par signe+module

Le signe d'un entier ne peut prendre que 2 valeurs (+ ou -), il suffit donc de prévoir un bit de plus (0 : positif et 1 : négatif) que l'on placera en position MSB

Avec n bits, on représente des nombres entiers relatifs compris entre $-(2^{n-1}-1)$ et $(2^{n-1}-1)$

Exemple : Avec 8 bits, on représente les nombres entiers compris entre -127 et 127
En effet, $-(2^{8-1}-1) = -(2^7-1) = -(128-1) = -127$ et $(2^{8-1}-1) = 127$

Remarque : Pour « zéro », il y a une double représentation, par exemple avec 8 bits

- $-0 \rightarrow (1000000)_2$
- $+0 \rightarrow (0000000)_2$

5.2 Codage par complément à 2

La représentation en complément à 2 permet un codage des entiers positifs ou négatifs en adoptant les règles suivantes

- **Si le nombre est positif, il est représenté par son code binaire affecté du bit de signe « 0 »**
- **Si le nombre est négatif, il est représenté par le complément à 2 du nombre positif et défini par $Y = 2^n - X$ avec X le nombre positif et n le nombre de bits de X**

Exemple : Cherchons le code du nombre -2 avec 3 bits
X = 2 et n=3 donc $Y = 2^3 - 2 = 8 - 2 = 6 = (110)_2$

Avec 3 bits, on peut représenter les nombres entiers compris entre -4 et 3

		100	-4
+3	011	101	-3
+2	010	110	-2
+1	001	111	-1
+0	000		

Avec n bits, on représente des nombres entiers relatifs compris entre $-(2^{n-1})$ et $(2^{n-1}-1)$

Remarques :

- La représentation pour « zéro » est unique
- En complément à 2, un nombre négatif est automatiquement affecté du bit de signe « 1 »

Synthèse

Les processeurs de nos ordinateurs sont des processeurs 32 bits ou 64 bits, cela veut dire qu'ils disposent de 32 bits ou 64 bits pour représenter (stocker) un nombre.

Représentation sur … bits	Plage des entiers positifs	Plage des entiers relatifs en complément à 2
8	0 à 255	-128 à 127
16	0 à 65 535	-32 768 à 32 767
32	0 à 4 294 967 295	- 2 147 483 648 à 2 147 483 647
64	0 à 18 446 744 073 709 551 616	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

6. Opérations en binaire

7.1. Addition

L'addition en binaire suit les mêmes règles qu'en décimal. On appelle également la retenue éventuelle CARRY. L'addition de deux nombres binaires se résume par la table suivante :

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

je pose 0 et je retiens 1

Exemple : On cherche à additionner 5+7 en binaire
$(5)_{10} = (101)_2$ et $(7)_{10} = (111)_2$
on pose l'opération :

	1	**1**	**1**	**Retenues**
+		1	0	1
+		1	1	1
	1	1	0	0

Vérifions $(1100)_2 = (12)_{10}$

Pour l'addition de deux nombres binaires (sur plusieurs bits), on peut donc considérer qu'il faut ajouter à chaque rang, 3 bits :

$$S_i = A_i + B_i + R_{i-1}$$

On peut donc définir une table que l'on appelle **table de vérité de l'additionneur**

R_{i-1}	A_i	B_i	S_i	R_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Exemples :

Effectuer l'addition binaire de 203 et 158 sur 8 bits

$(203)_{10} = ($ 1100 1011 $)_2$

$(158)_{10} = ($ 1001 1110 $)_2$

	1			1	1	1	1		
+		1	1	0	0	1	0	1	1
+		1	0	0	1	1	1	1	0
	1	0	1	1	0	1	0	0	1

Soit en décimal: **105**

Remarque :

Sur 8 bits le résultat est faux. Le calcul provoque un dépassement de capacité que l'on appelle OVERFLOW. Il faut donc prendre un 9^{eme} bit pour que le résultat soit juste.

Attention : pour avoir un résultat juste, il faut que le format soit respecté, c'est à dire que le résultat ne soit pas supérieur aux valeurs que l'on peut exprimer par le format choisi.

Pour représenter la somme de deux entiers représentés sur k bits, il faut au plus $(k + 1)$ bits

7.2. Soustraction

Une soustraction peut toujours se ramener à une addition avec un deuxième terme négatif.
Il faut simplement savoir dans quel codage, on s'exprime. Le plus souvent on s'exprimera avec la méthode du complément à 2.

Exemple :

Effectuer l'opération 195 – 96 en binaire (complément à 2) sur 8 bits

$(195)_{10} = (\ 1100\ 0011\ )_2$

Le nombre -96 est négatif, il est représenté par le complément à 2 du nombre positif et défini par

$Y = 2^n - X$ avec X = 96 et n = 8

donc $Y = 2^8 - 96 = 160 = (1010\ 0000)_2$ alors $(-96)_{10} = (\ 1010\ 0000\ )_2$

	1							
+		1	1	0	0	0	0	1	1
+		1	0	1	0	0	0	0	0
	1	0	1	1	0	0	0	1	1

Soit en décimal: 99

Remarque :

Il n'y a pas d'OVERFLOW dans ce cas vu que l'on additionne un nombre positif et un nombre négatif.

7.3. Multiplication

La multiplication en binaire suit les mêmes règles qu'en décimal. Elle se résume par la table suivante :

A	B	P
0	0	0
0	1	0
1	0	0
1	1	1

Exemple :
Effectuer le produit en binaire de 6 par 9

$(6)_{10}$ = (0110)$_2$

$(9)_{10}$ = (1001)$_2$

				0	1	1	0
*				1	0	0	1
				0	1	1	0
+			0	0	0	0	
+		0	0	0	0		
+	0	1	1	0			
	0	1	1	0	1	1	0

Soit en décimal: 54

Pour représenter le produit de deux entiers représentés respectivement sur p et q bits, il faut au plus $(p + q)$ bits