Codage de l'information

Laurent Noé-Léopold Weinberg
paternité : Francesco De Comité

Licence 1 Mathématiques Informatique Semestre 1
Université de Lille
Faculté des Sciences et Technologies

10 septembre 2025

Ecriture des nombres dans d'autres bases

Histoire et morphologie humaine

Les humains ont naturellement utilisé leurs doigts pour compter.

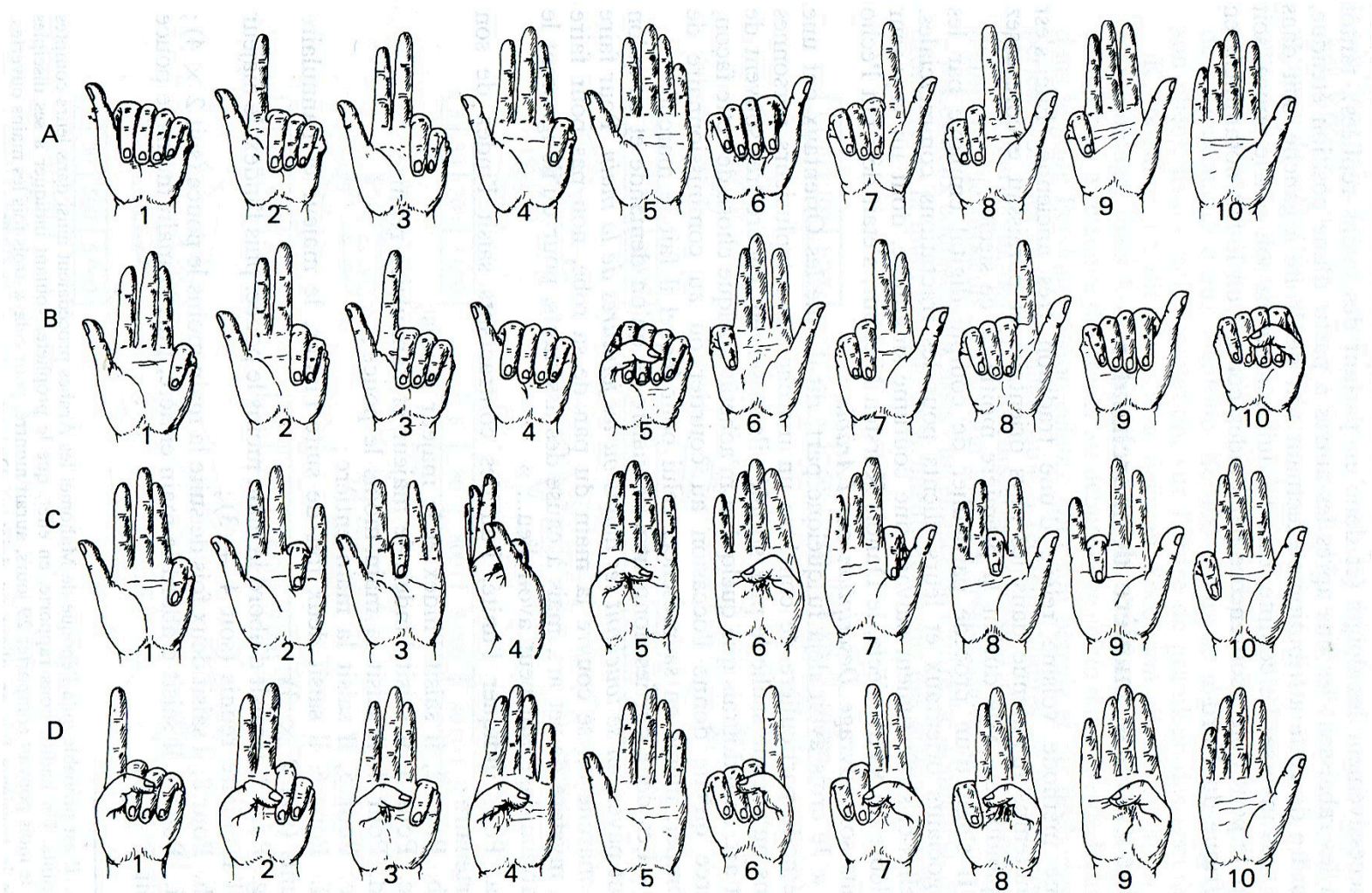


Fig. 3.3. – *Variantes du compte digital élémentaire.*

crédit : G. Ifrah Histoire universelle des chiffres

Histoire et morphologie humaine

Considérations

- C'est sans doute pourquoi on utilise 10 chiffres pour représenter les nombres.
- Digital : *relatif aux doigts*
- Ok, les Babyloniens travaillaient aussi avec les multiples de 60 (voir cours 1). Mais ils n'avaient pas soixante symboles, seulement 10 pour les unités, et 5 pour les dizaines.

J.L.Borges : Irénée Funes ou la mémoire

On pourrait imaginer un système de numération où chaque nombre aurait un nom différent. Mais J.L. Borgès l'a déjà fait :

> *Il me dit que vers 1886, il avait imaginé un système original de numération et qu'en très peu de jours il avait dépassé le nombre vingt-quatre mille. Il ne l'avait pas écrit, car ce qu'il avait pensé une seule fois ne pouvait plus s'effacer de sa mémoire. Il fut d'abord, je crois, conduit à cette recherche par le mécontentement que lui procura le fait que les Trente-Trois Orientaux exigeaient deux signes, et trois mots, au lieu d'un seul mot et d'un seul signe. Il appliqua ensuite ce principe extravagant aux autres nombres. Au lieu de sept mille treize, il disait (par exemple), Maxime Pérez ; au lieu de sept mille quatorze, Le chemin de fer ; d'autres nombres étaient Luis Melain Lafinur, Olimar, soufre, le bât, la baleine, le gaz, la chaudière, Napoléon, Augustin de Vedia. Au lieu de cinq cents il disait neuf. Chaque mot avait un signe particulier, une sorte de marque ; les derniers étaient très compliqués...*

Restriction du domaine

Contraintes

- En quelle bases compterions-nous si nous avions six doigts à chaque main ?
- Y a-t-il des bases plus pratiques que les autres ?
  - Pour le calcul mental ?
  - Pour représenter les nombres dans les mémoires d'ordinateurs ?
- On ne considérera dans la suite que les bases différentes de 10, mais en se basant toujours sur la numération positionnelle.
- Des bases plus exotiques seront éventuellement abordées durant les travaux dirigés.

La numération de position en base 10 (rappel du cours 1)

Définition

- On définit un ensemble $\mathcal{C}$ de 10 caractères, les chiffres.
- Chacun de ces chiffres est associé à une valeur unique comprise entre 0 et 9.
- Un nombre est représenté par une suite de chiffres

$$a_{t-1}a_{t-2}\cdots a_1 a_0, \forall i \in [\![0, t-1]\!], a_i \in \mathcal{C}$$

- La valeur représentée par ce nombre est :

$$n = \sum_{i=0}^{t-1} a_i \times 10^i$$

- Exemple :

$$1214 = 1 \times 10^3 + 2 \times 10^2 + 1 \times 10^1 + 4 \times 10^0$$

Rappels et généralisation

Notations et définitions (rappel du cours 1)

- La taille d'un nombre est définie par le nombre de chiffres qui le composent.
- Le chiffre représentant les unités, le plus à droite dans l'écriture du nombre est appelé le *chiffre de poids faible*.
- Réciproquement, le chiffre le plus à gauche est appelé le *chiffre de poids fort*.
- Le chiffre de poids fort est celui qui a le plus d'influence sur la valeur finale du nombre.
- Le chiffre de poids fort est forcément différent de zéro.
- On dira que des zéros éventuels à gauche du chiffre de poids fort sont *non significatifs*.

La numération de position en base b

Définition

- On définit un ensemble $\mathcal{C}$ de b caractères, les chiffres.
- Chacun de ces chiffres est associé à une valeur unique comprise entre 0 et $b-1$ grâce à la fonction *val* de $\mathcal{C}$ dans $\mathbb{N}$
- Un nombre est représenté par une suite de chiffres

$$a_{t-1}a_{t-2}\cdots a_1 a_0, \forall i \in [\![0, t-1]\!], a_i \in \mathcal{C}$$

- La valeur représentée par ce nombre est :

$$n = \sum_{i=0}^{t-1} val(a_i) \times b^i$$

Définition : la numération positionnelle en base b

Propriétés, contraintes et conventions

- b doit être entier et strictement supérieur à 1.
- La décomposition d'un nombre en base b est unique.
- Lorsque $b \leq 10$, on utilise les chiffres standards pour représenter les chiffres en base b
- Lorsque $b > 10$, on utilisera les lettres majuscules, en commençant par A,B,...
- Comme pour la base 10, il est facile de comparer deux nombres.
- On peut ici aussi se poser la question : combien de chiffres aura la représentation d'un nombre donné dans une base b ?

Quelques exemples

En base 7 (sept chiffres : $\{0, 1, 2, 3, 4, 5, 6\}$)

$$346 = 3 \times 7^2 + 4 \times 7^1 + 6 \times 7^0 = 181$$

En base 18 (dix-huit chiffres : $\{0, 1, \cdots, 9, A, B, C, D, E, F, G, H\}$)

$$3A6 = 3 \times 18^2 + 10 \times 18^1 + 6 \times 18^0 = 1158$$

En base 3 (trois chiffres : $\{0, 1, 2\}$)

$$212 = 2 \times 3^2 + 1 \times 3^1 + 2 \times 3^0 = 23$$

la numération positionnelle en base b

Bases particulières

- Certains prétendent que la base 12 aurait été plus pratique à utiliser que la base 10 : 12 a en effet plus de diviseurs que 10, ce qui simplifierait la manipulation des nombres (multiplication, division, caractère de divisibilité...)
- En informatique, où les informations de base (les *bits*) ne peuvent prendre que deux valeurs, il est pratique d'utiliser la base 2.
- Par souci de concision (longueur des nombres), on utilise aussi les bases où b est une puissance de 2, principalement les bases 8 et 16.

Notation et exemples

Notation

- Puisqu'à partir de maintenant, on manipulera des représentations de nombres dans des bases différentes, on doit connaître la base b dans laquelle on travaille.
- Par exemple : quelle est la quantité représentée par 215 ?
  - Est-ce le 215 *standard* en base 10 ?
  - Est-ce qu'on travaille en base 7, et on a alors affaire à la quantité :

$$2 \times 7^2 + 1 \times 7^1 + 5 \times 7^0 = 110$$

(Attention, la valeur '110' est en base 10)

Notation et exemples

Définition

Soit la quantité A, qui se décompose dans une base b sous la forme :

$$A = a_n \times b^n + a_{n-1} \times b^{n-1} \cdots a_1 \times b^1 + a_0 \times b^0$$

On notera sa représentation sous la forme suivante :

$$A = \overline{\mathtt{a_n a_{n-1} \cdots a_1 a_0}}_b$$

Exemples :

$$\overline{123}_7, \overline{100101001010}_2, \overline{123}_{10}$$

Changements de base

D'une base b à la base 10

Pour passer d'une écriture en base b à la valeur en base 10 :

- On peut utiliser la formule :

$$A = a_n \times b^n + a_{n-1} \times b^{n-1} \cdots a_1 \times b^1 + a_0 \times b^0$$

- Mais on peut faire mieux, si on considère le nombre d'additions/multiplications :
  - Soit le nombre $A = \overline{\mathtt{a_n a_{n-1}} \cdots \mathtt{a_1 a_0}}_b$
  - Poser resu $= 0$
  - Pour i de n à 0, faire resu $= b \times$ resu $+ a_i$
  - Retourner resu
- C'est une application de l'algorithme de Horner.

Changements de base

Exemple

$$A = \overline{236}_7 \text{ (Spoiler : 125 en base 10)}$$

$$resu = 0$$

$$resu = 0 \times 7 + 2 = 2$$

$$resu = 2 \times 7 + 3 = 17$$

$$resu = 17 \times 7 + 6 = 125$$

De la base 10 à la base b

Procédure

- Soit à transcrire le nombre A s'écrivant $\overline{\mathtt{a_n a_{n-1} \cdots a_1 a_0}}_{10}$ dans son écriture en base b : $\overline{\mathtt{b_m b_{m-1} \cdots b_1 b_0}}_{b}$
- Le chiffre b_0 est le reste du nombre A dans sa division par b, donc $b_0 = A$ modulo b
- Il reste la partie de A qui n'a pas été traitée : A div b (division entière). On pose $A' = A$ div b
- Son reste dans la division par b donne b_1, il faut maintenant traiter $A'' = A'$ div b
- On continue tant que le nombre à traiter est différent de zéro.

De la base 10 à la base b

Exemple

Soit à transcrire le nombre s'écrivant 2567 en base 10 vers la base 7.

$$
\begin{array}{ccccccccc}
2567 & \text{mod} & 7 & = & 5 & \longrightarrow & b_0 & = & 5 \\
2567 & \div & 7 & = & 366 & \longrightarrow & A' & = & 366 \\
366 & \text{mod} & 7 & = & 2 & \longrightarrow & b_1 & = & 2 \\
366 & \div & 7 & = & 52 & \longrightarrow & A'' & = & 52 \\
2 & \text{mod} & 7 & = & 3 & \longrightarrow & b_2 & = & 3 \\
52 & \div & 7 & = & 7 & \longrightarrow & A''' & = & 7 \\
7 & \text{mod} & 7 & = & 0 & \longrightarrow & b_3 & = & 0 \\
7 & \div & 7 & = & 1 & \longrightarrow & A'''' & = & 1 \\
1 & \text{mod} & 7 & = & 1 & \longrightarrow & b_4 & = & 1
\end{array}
$$

Finalement : $A = \overline{10325}_7$

La base 2, le binaire

Un peu d'histoire

- L'utilisation de la base 2 s'est vite imposée lors de la conception des premiers ordinateurs.
  - Dans une machine électro-mécanique ou électronique, un état du système se code facilement par la présence ou l'absence de courant.
  - Détecter et manipuler cette présence ou absence de courant se fait avec des composants de base (relais, lampes, puis transistors).
  - On peut se servir du codage binaire à la fois pour les commandes système et pour la représentation des données, ce qui simplifie la conception et la fabrication des ordinateurs.
  - Les opérations de base deviennent plus faciles à définir.

Numération binaire de position

Propriétés

- La base 2 est la plus petite possible.
- Elle ne connaît que deux chiffres : 0 et 1.
- Les opérations arithmétiques élémentaires deviennent plus faciles à décrire que dans toute autre base.

Lire en binaire

Lire la base 2 dans le texte

- Savoir lire directement des nombres en base 2 est un petit talent qui peut se révéler bien utile, et qui fait gagner du temps.
- Connaître la valeur des premières puissances de 2 simplifie la vie.

Puissance	Valeur	Puissance	Valeur
2^0	1	2^6	64
2^1	2	2^7	128
2^2	4	2^8	256
2^3	8	2^9	512
2^4	16	2^{10}	1024
2^5	32	2^{11}	2048

Lire en binaire

Quelques puissances utiles

Puissance	nom	Valeur	Proche de ...	Nom approché
2^{10}	kibi	1024	1000	Kilo
2^{20}	mébi	1,048,576	10^6	Méga
2^{30}	gibi	1,073,741,824	10^9	Giga

Lire en binaire

Lire directement un nombre binaire

- Soit le nombre binaire $\overline{1101101_2}$
- Le chiffre en $i$ème position à partir de la droite correspond à 2^i (on commence à numéroter à partir de zéro).
- Pour traduire $\overline{1101101_2}$, pas besoin de multiplication, on additionne les puissances présentes (les “1”).
- Donc $\overline{1101101_2}$ représente $1 + 4 + 8 + 32 + 64 = 109$

L'addition en binaire

Procédure

- La mise en place des nombres et l'addition colonne par colonne sont les mêmes que pour l'addition en base 10.
- Les tables d'additions sont simplifiées.

Add	0	1
0	0	1
1	1	10

Table d'addition

Unité	0	1
0	0	1
1	1	0

Chiffre des unités

Retenue	0	1
0	0	0
1	0	1

Retenue

Definitions

Les tables de "Unités" et de "Retenue" sont respectivement celles des fonctions **Ou Exclusif** et **Et Logique**.

Comment ça marche ? Un exemple : 11+14=25

		1	0	1	1
+		1	1	1	0

		1	0	1	1
+		1	1	1	0
					1

		1	0	1	1
+		1	1	1	0
				10	1

			1		
		1	0	1	1
+		1	1	1	0
			10	0	1

		1			
		1	0	1	1
+		1	1	1	0
		11	0	0	1

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

La multiplication en binaire

- On va utiliser une version un peu modifiée de l'algorithme standard de multiplication : on va commencer par multiplier par le poids fort du multiplicateur...
- ...ce qui donnera un algorithme plus adapté à l'implémentation physique dans un ordinateur.
- Soit à multiplier $\overline{101110_2}$ par $\overline{11010_2}$

						1	0	1	1	1	0
	$\times$						1	1	0	1	0

La multiplication en binaire

- Etape 1 : Produit du multiplicande par le chiffre de poids fort du multiplicateur.
- On remarque que les deux seuls résultats possibles sont 0 ou bien le multiplicande lui-même.

					1	0	1	1	1	0
						1	1	0	1	0
×										
					1	0	1	1	1	0

- Etape 2 : décaler le résultat d'un rang vers la gauche.

					1	0	1	1	1	0
						1	1	0	1	0
×										
				1	0	1	1	1	0	0

La multiplication en binaire

- Produit du multiplicande par le chiffre suivant.

					1	0	1	1	1	0
$\times$						1	1	0	1	0
				1	0	1	1	1	0	0
					1	0	1	1	1	0

- Additionner le résultat partiel et la nouvelle ligne.

					1	0	1	1	1	0
$\times$						1	1	0	1	0
			1	0	0	0	1	0	1	0

- Décaler le résultat partiel.

					1	0	1	1	1	0
$\times$						1	1	0	1	0
		1	0	0	0	1	0	1	0	0

La multiplication en binaire

- Le chiffre suivant du multiplicateur est zéro : on n'a pas de nouveau résultat partiel à ajouter, on ne fait que le décalage.

					1	0	1	1	1	0
×						1	1	0	1	0
	1	0	0	0	1	0	1	0	0	0

- Chiffre suivant.

					1	0	1	1	1	0
×						1	1	0	1	0
	1	0	0	0	1	0	1	0	0	0
					1	0	1	1	1	0

La multiplication en binaire

- Additionner les deux résultats partiels.

					1	0	1	1	1	0
$\times$						1	1	0	1	0
	1	0	0	1	0	1	0	1	1	0

- Dernier chiffre du multiplicateur : zéro, la seule chose à faire est de décaler. (comme on fait en base 10 lorsqu'on multiplie par un multiple de 10)

					1	0	1	1	1	0
$\times$						1	1	0	1	0
1	0	0	1	0	1	0	1	1	0	0

La multiplication en binaire

Vérification

- Multiplicande :

$$\overline{101110_2} = 2 + 4 + 8 + 32 = 46$$

- Multiplicateur :

$$\overline{11010_2} = 2 + 8 + 16 = 26$$

- Résultat :

$$\overline{10010101100_2} = 4 + 8 + 32 + 128 + 1024 = 1196$$

- Vérification : $46 \times 26 = 1196$

La division en binaire

En base 10, la division est complexe parce qu'on doit *deviner* combien de fois le diviseur est présent dans une partie du dividende.

2 5 6 4 2	1 1 7
2 2 4	2 1 9
1 0 7 2	
1 9	

2 5 6 4 2	4 7
2 1 4	5 4 5
2 6 2	
2 7	

- Sur le premier exemple, on considère immédiatement les trois chiffres à gauche du dividende, qui forment un nombre supérieur au diviseur, et on cherche le chiffre du quotient correspondant (ici un 2).
- Sur le deuxième exemple, on compare 47 à 25, la tranche du dividende est inférieure au diviseur, on prend alors une tranche de trois chiffres. Il faut encore trouver *en 256 combien de fois 47...*

La division en binaire

Tout est plus simple en binaire :

- Un chiffre du quotient ne peut être que 0 ou bien 1.
- Il suffit de voir si le dividende est supérieur au diviseur (on obtient un 1) ou non (on obtient un 0).

Un exemple

- On veut diviser $\overline{1101101_2}$ par $\overline{1011_2}$
- On aligne les deux nombres à gauche :

1	1	0	1	1	0	1	Quotient
1	0	1	1				1

- Diviseur inférieur à la tranche du dividende de même taille : concaténer un '1' au quotient.

La division en binaire

Un exemple

- Soustraire le diviseur de la tranche de dividende (résultat : 0010), et décaler le diviseur vers la droite.

0	0	1	0	1	0	1	Quotient
	1	0	1	1			1

- Tranche de dividende inférieure au diviseur : mettre 0 dans le quotient.

0	0	1	0	1	0	1	Quotient
	1	0	1	1			1 0

La division en binaire

Un exemple

- Pas de soustraction, seulement le décalage du diviseur vers la droite :

0	0	1	0	1	0	1	Quotient
		1	0	1	1		1 0				

- Tranche de dividende inférieure au diviseur : mettre 0 dans le quotient.

0	0	1	0	1	0	1	Quotient
		1	0	1	1		1 0 0			

- Pas de soustraction, seulement le décalage du diviseur vers la droite :

0	0	1	0	1	0	1	Quotient
			1	0	1	1	1 0 0			

La division en binaire

Un exemple

- Tranche de dividende supérieure au diviseur : soutraction, et mettre 1 dans le quotient.

0	0	1	0	1	0	1	Quotient				
			1	0	1	1	1	0	0	1	

Conclusion, vérification

- Est-ce que $\overline{1101101_2} \div \overline{1011_2} = \overline{1001_2}$, reste $\overline{1010_2}$?
- $\overline{1101101_2} = 1 + 4 + 8 + 32 + 64 = 109$
- $\overline{1011_2} = 1 + 2 + 8 = 11$
- $\overline{1001_2} = 1 + 8 = 9$
- $\overline{1010_2} = 2 + 8 = 10$
- $109 = 9 \times 11 + 10$

La division en binaire

Conclusion

- On a besoin des actions primitives suivantes :
  - Comparaison de deux nombres en binaire.
  - Soustraction.
  - Décalage et alignement de nombres.
- L'algorithme est simple à programmer ou à câbler *"en dur"*.

Les entiers des ordinateurs

Un cadre plus strict

Entiers de taille limitée

- La plupart des objets manipulés en informatique sont de taille fixe :
  - Parce qu'on a besoin de savoir la place qu'ils prennent en mémoire.
  - Parce que les circuits qui les traitent sont de taille fixe (registres, accumulateurs, bus ...). On parle de systèmes 8, 16,32,64 bits ...
- En particulier, les nombres entiers seront codés en utilisant toujours la même taille de codage.

Tailles standards

Tailles

- Les tailles des entiers utilisés dans les ordinateurs sont 8,16,32,64 : des puissances de 2.
- *A priori, il n'y a pas de raison impérative pour avoir choisi des tailles qui soient des puissances de 2.*
- Si l'on sait faire une machine qui fonctionne sur N bits, il est facile d'améliorer cette machine en la faisant fonctionner sur $2 \times N$ bits : on dédouble tous les circuits.

Les différents formats

Description

- Un octet : 8 bits, peut coder $2^8 = 256$ valeurs différentes.
  Par exemple les nombres entiers compris entre 0 et 255 inclus.
- Sur deux octets (16 bits) : on peut coder $2^{16} = 65536$ valeurs différentes (de 0 à 65535, par exemple).

Octal, hexadécimal

- Afin d'obtenir une représentation plus compacte des nombres exprimés en binaire, on utilise les bases 8 (octale) et 16 (hexadécimale).
- Les passages de binaire vers octal ou de binaire vers hexadécimal sont plus simples qu'un changement de base entre deux bases quelconques.
- Un chiffre en octal représente une valeur parmi 8, et sera donc codée en binaire sur trois bits exactement.
- De même, un chiffre en hexadécimal peut prendre 16 valeurs différentes, qui se codent sur quatre chiffres en binaire.

Un brin de Python

Ecriture en Python

- En Python, un nombre en octal est précédé du préfixe 0o : 0o217 est un nombre écrit en octal qui vaut

$$2 \times 8^2 + 1 \times 8^1 + 7 \times 8^0 = 143 \text{ en base } 10.$$

- Un nombre hexadécimal commence par le préfixe 0x :

$0xA2$ en hexadécimal vaut $10 \times 16^1 + 2 \times 16^0 = 162$ en base 10.

Octal vers binaire

- On remplace chaque chiffre octal par l'écriture en binaire de ce chiffre sur trois bits.
- Exemple : écrire $\overline{7325}_8$ en binaire

En octal	7	3	2	5
	↓	↓	↓	↓
En binaire	111	011	010	101

- Résultat : $\overline{111011010101}_2$

Binaire vers octal

- Réciproquement, on regroupe trois par trois les chiffres du nombre écrit en binaire, en commençant par la droite, et on remplace chaque groupe de trois chiffres par le chiffre octal correspondant.
- Si le nombre binaire n'a pas un nombre de chiffres divisible par 3 on le complète à gauche par un ou deux zéros.
- Exemple : écrire $\overline{1110010100110}_2$ en octal.

En binaire	001	110	010	100	110
	↓	↓	↓	↓	↓
En octal	1	6	2	4	6

- Résultat : $\overline{16246}_8$

Hexadécimal vers binaire

- On remplace chaque chiffre hexadécimal par l'écriture binaire de la valeur de ce chiffre sur quatre bits :
- Exemple : écrire $\overline{\texttt{FFE17}}_{16}$ en binaire :

En hexadécimal	F	F	E	1	7
	↓	↓	↓	↓	↓
En binaire	1111	1111	1110	0001	0111

- Résultat : $\overline{1111111111000010111}_2$

Binaire vers hexadécimal

- On regroupe les chiffres binaires par groupe de quatre, en commençant par la g. On complète éventuellement le dernier groupement avec un, deux ou trois zéros.
- Exemple : traduire le nombe binaire $\overline{1101101110101000 10_2}$ en hexadécimal :

En binaire	0011	0110	1110	1010	0010
	↓	↓	↓	↓	↓
En hexadécimal	3	6	E	A	2

- Résultat : $\overline{36EA2_{16}}$

Pense-bête

Pour gagner du temps, il est utile de connaître les écritures binaires, octales et hexadécimales des 16 premiers nombres.

Valeur	Binaire	Octal	Hexa	Valeur	Binaire	Octal	Hexa
0	0	0	0	8	1000	10	8
1	1	1	1	9	1001	11	9
2	10	2	2	10	1010	12	A
3	11	3	3	11	1011	13	B
4	100	4	4	12	1100	14	C
5	101	5	5	13	1101	15	D
6	110	6	6	14	1110	16	E
7	111	7	7	15	1111	17	F

Représentation des entiers signés

Motivation

- Jusqu'à présent, on n'a considéré que les entiers naturels (entiers positifs ou nuls).
- Comment peut-on représenter les entiers relatifs (positifs ou négatifs) ?
- Première idée : réserver un bit pour indiquer le signe de l'entier. Par exemple 0 pour un nombre positif, 1 pour un nombre négatif.
- Problèmes :
  - On doit *sacrifier un bit pour cette information*
  - Il faut redéfinir les opérations de base pour tenir compte de ce signe (et de la façon choisie pour le représenter).
  - On perd une valeur : on aura deux représentations pour 0...
- On parle de *représentation par signe et valeur absolue*.

Signe et valeur absolue

Représentation

On considère des nombres écrits sur un nombre de bits fixés. Le bit le plus à gauche représentera le signe. Par exemple, sur 4 bits, le signe positif représenté par un '0', et négatif par un '1' :

Binaire	Valeur	Binaire	Valeur
0000	+0	1000	-0
0001	+1	1001	-1
0010	+2	1010	-2
0011	+3	1011	-3
0100	+4	1100	-4
0101	+5	1101	-5
0110	+6	1110	-6
0111	+7	1111	-7

Signe et valeur absolue

Caractéristiques générales

Pour une représentation signe + valeur absolue sur N bits (exemple avec $N = 4$) :

- Plus petite valeur représentée : $-(2^{n-1} - 1)$ (-7)
- Plus grande valeur représentée : $2^{n-1} - 1$ (7)
- Nombre de valeurs représentées : $2^n - 1$ (15)

Opérations : algorithmes

On considère des nombres écrits sur n bits (signe compris).

Addition/soustraction

- Comparer les valeurs absolues (i.e. enlever les signes, et comparer les deux entiers)
- Mêmes signes : faire la somme (sur $n-1$ bits), puis remettre le signe.
- Signes opposés : effectuer la soustraction, puis remettre le signe du plus grand nombre en valeur absolue.

Opérations : algorithmes

Multiplication, division

- Enlever les signes.
- Effectuer l'opération.
- Signes identiques : mettre le signe '+' devant le résultat.
- Signes opposés : mettre le signe '-'

Complément à deux

Motivation

- La représentation des entiers en signe+valeur absolue entraîne une modification de l'algorithme d'addition :
  - Comparer les valeurs absolues.
  - Effectuer l'opération.
  - Remettre le signe.

  Cette modification de l'algorithme a un coût...
- Puisque 0 a deux représentations, on perd la possibilité de stocker une valeur supplémentaire.

Complément à deux

Définition

- On choisit N le nombre de bits sur lesquels seront stockés les entiers.
- Les configurations binaires dont le bit de poids fort est '0' seront lues comme des entiers positifs codés en binaire.
- Les configurations binaires commençant par un '1' seront considérées comme des nombres négatifs.
- Pour passer d'un nombre positif à son opposé, on inverse tous les bits de sa représentation binaire, et on ajoute 1 (voir cours précédent, le complément à 10).

Complément à deux

Prenons $N = 4$

Binaire	0000	0001	0010	0011	0100	0101	0110	0111
Décimal	0	1	2	3	4	5	6	7
Inversion	1111	1110	1101	1100	1011	1010	1001	1000
$+1$	10000	1111	1110	1101	1100	1011	1010	1001
Décimal	0	-1	-2	-3	-4	-5	-6	-7

- Une configuration binaire n'est pas utilisée : 1000
- Calculons son complément à 2 : $1000 \longrightarrow 0111 \longrightarrow 1000$
- Ce nombre commence par 1 : c'est donc un négatif.
- Le prochain négatif non codé est -8, on lui attribue ce codage.
- Il est son propre complément à 2 (comme 0).

Complément à deux

Définition alternative

- On choisit un codage sur N bits.
- Les nombres compris entre 0 et $2^{N-1}-1$ sont codés en binaire *naturel* sur N bits.
- Les nombres négatifs p compris entre -1 et -2^{N-1} sont codés par la représentation binaire de $2^N - |p|$
- Exemple : $N = 4$
  codage de -5 =codage binaire naturel de $2^4 - 5$

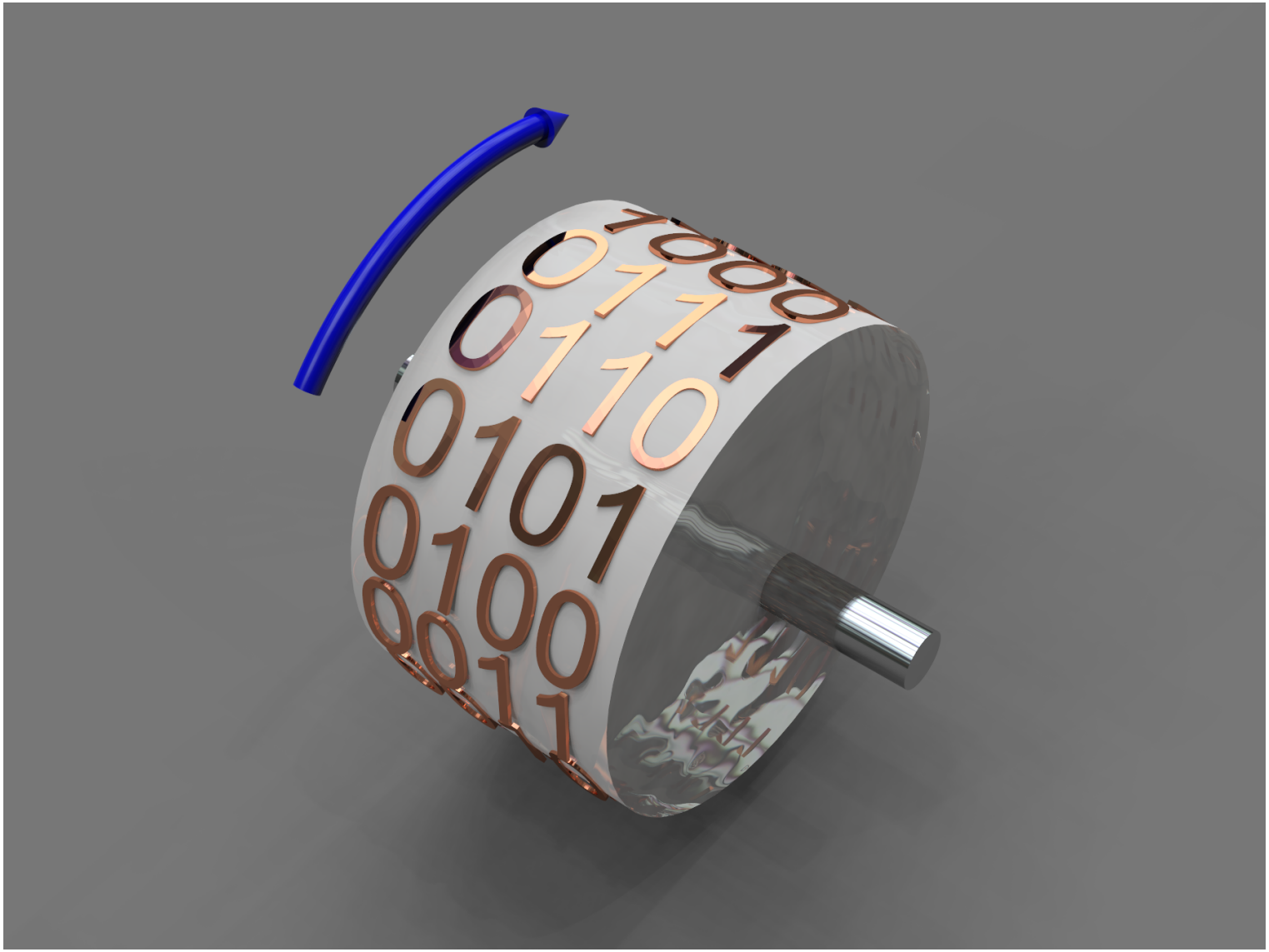
$$2^4 - 5 = 16 - 5 = 11 = \overline{1011}_2$$

Complément à deux

Valeur	codage
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

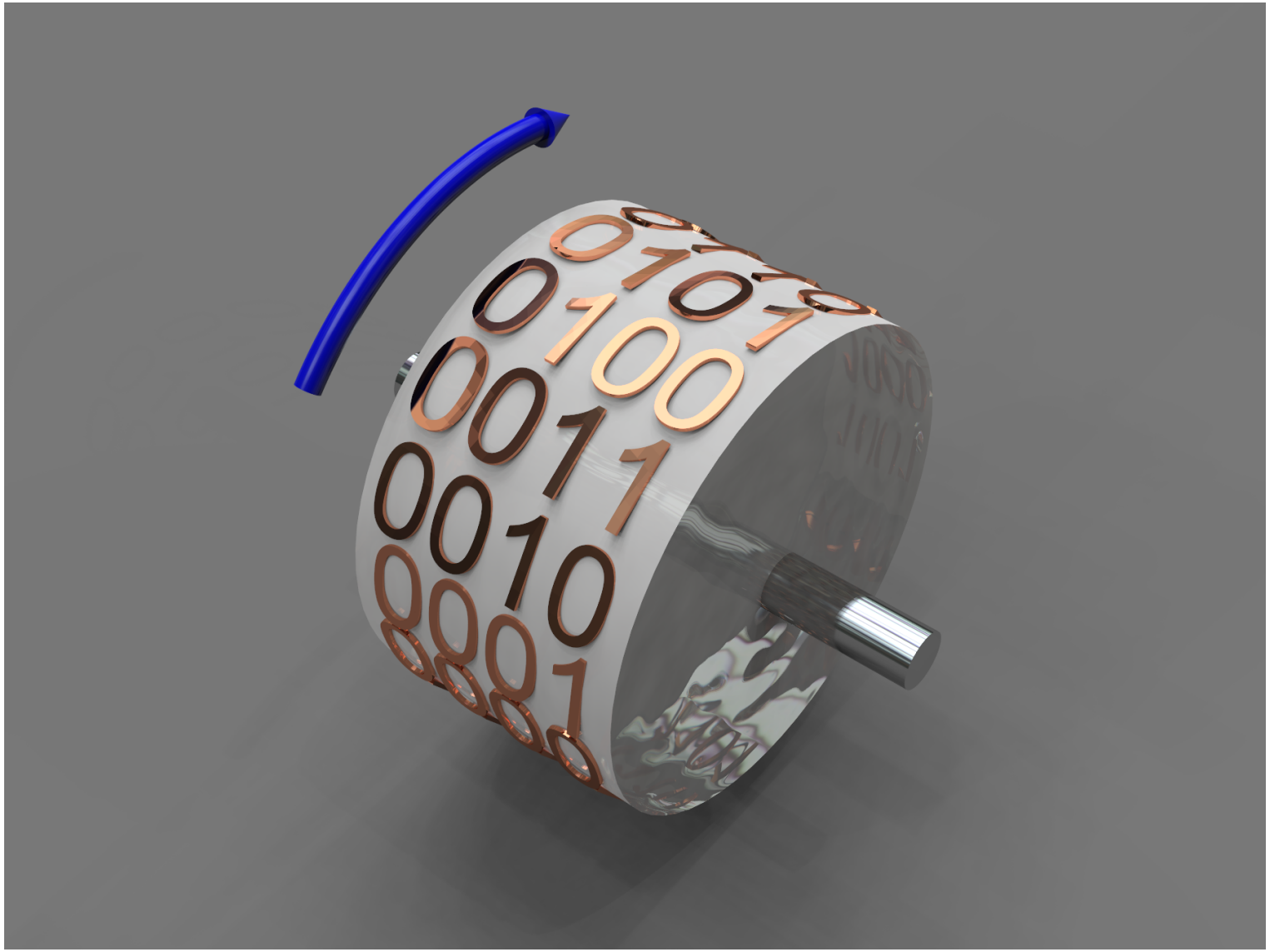
Complément à deux

On peut imaginer que les chiffres sont placés sur une roulette.



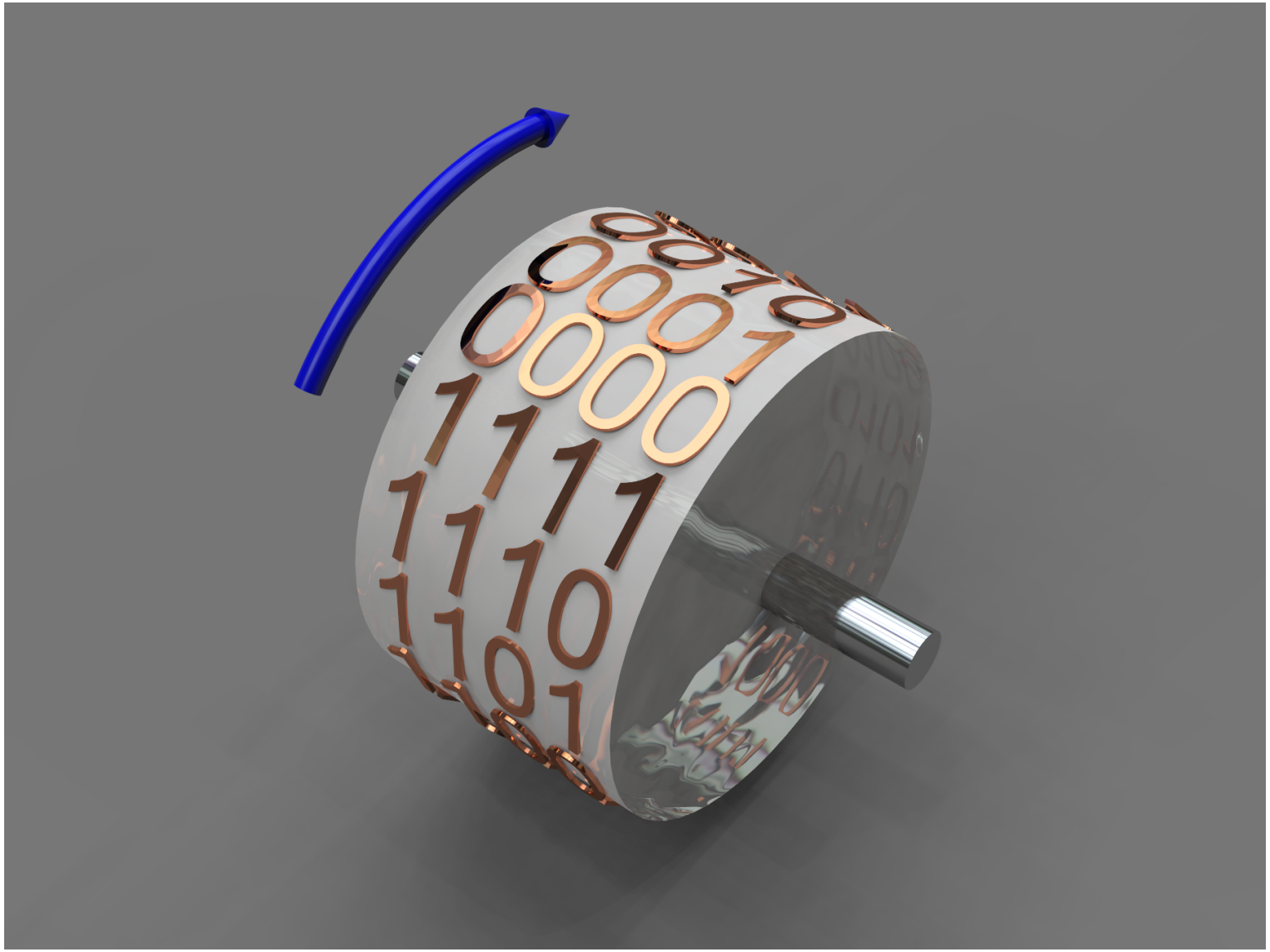
Complément à deux

La flèche donne le sens de l'addition.



Complément à deux

Où l'on repasse des négatifs vers les positifs.



Complément à deux

Propriétés

- L'addition de nombres de signes différents n'implique pas d'étape de traitement supplémentaire.
- Il existe des algorithmes de multiplication efficaces fonctionnant avec des nombres en complément à deux.
- Rien de tel (à ma connaissance) pour la division...
- Plusieurs exemples d'utilisation des nombres en complément à deux seront vus en TD.

Autres codages des entiers

Décimal codé binaire

Définition

- C'est un codage hybride, entre la base 10 et le binaire.
- Pour coder un nombre de N chiffres écrit en base 10, on code individuellement chacun de ses chiffres en binaire.
- Puisqu'il faut pouvoir coder dix chiffres en binaire, on a besoin de quatre bits pour chaque chiffre.
- Chaque chiffre est codé sur 4 bits en *binaire naturel*.
- Par exemple : 128 sera codé 0001 0010 1000
- Il a été défini et utilisé aux débuts de l'informatique.
- L'acronyme anglais est BCD (Binary coded decimal), on rencontre aussi BCD en français.

Code biquinaire

Le biquinaire permet de coder les chiffres décimaux sur 7 bits.

Chiffre	Biquinaire	
0	01	00001
1	01	00010
2	01	00100
3	01	01000
4	01	10000
5	10	00001
6	10	00010
7	10	00100
8	10	01000
9	10	10000

Biquinaire

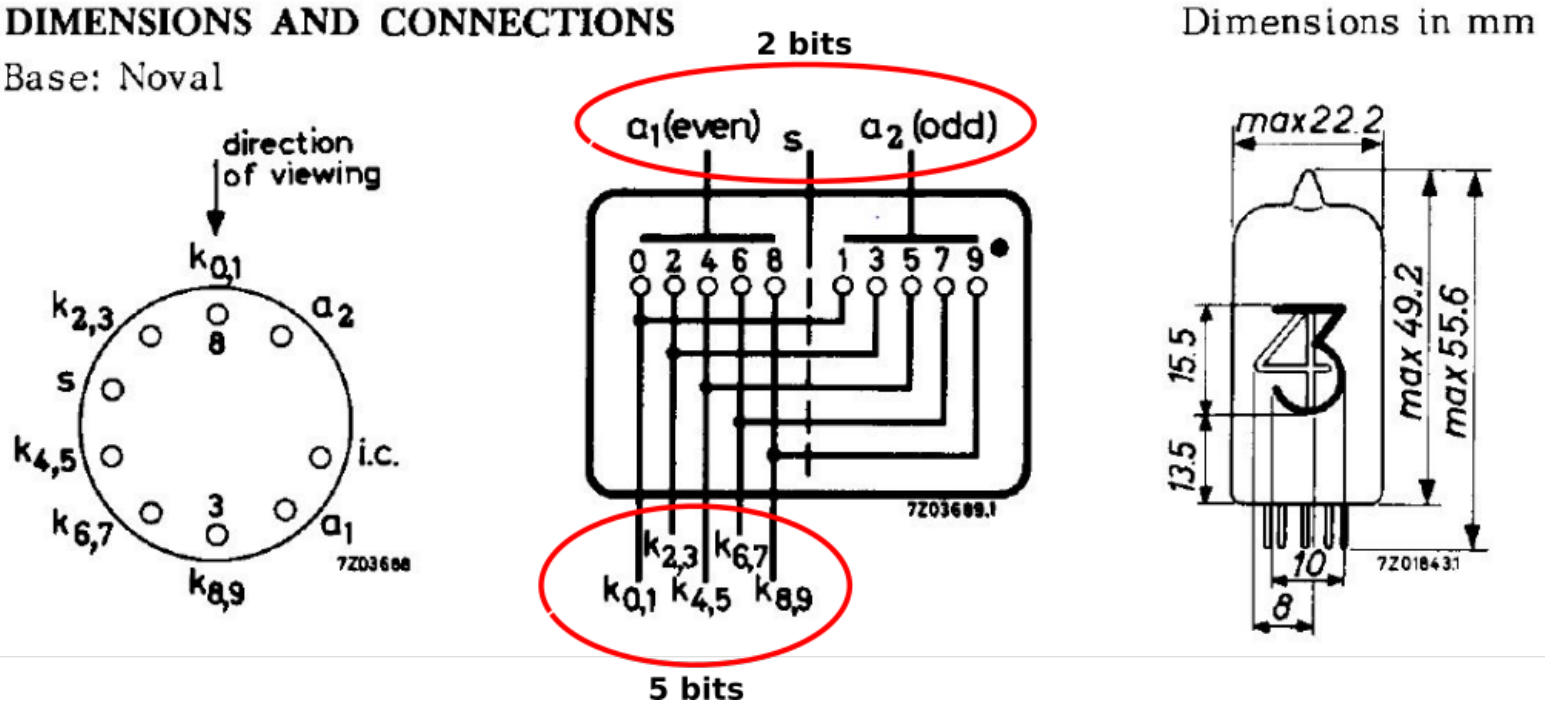
Propriétés

- Il y a toujours exactement deux 1 dans l'écriture d'un chiffre...
- ...dont un parmi les deux bits de poids fort.
- $\Rightarrow$ detection d'erreurs.
- Utilisé pour représenter les nombres dans certains des premiers ordinateurs.

Biquinaire : application

- Les premières interfaces de sortie étaient des imprimantes, des lampes et des tubes electroniques (bien avant les écrans).
- Les *NIXIES* sont des tubes permettant d'afficher un chiffre décimal.
- A chaque chiffre correspondait un filament qui devenait incandescent lorsqu'il était parcouru par le courant.
- On commandait ces tubes à l'aide de sept fils.

crédit : Martin Subrt



- Un seul des bits a_1 et a_2 vaut 1.
- Un seul des bits $K_{0,1}$, $k_{2,3}$, $k_{4,5}$, $k_{6,7}$, $k_{8,9}$ vaut 1.

credit : Phillips

Biquinaire : comme un goût de déjà-vu