Chapitre 4 : La chaîne d'information des systèmes

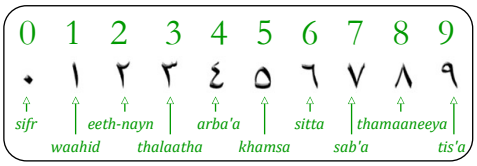
4.3 : Numération et codage des données

Cours – 4 h

Introduction

■ La ***numération*** est la ***formation*** *et la* ***représentation écrite*** des ***nombres***, à commencer par celle des ***entiers naturels***. En mathématiques, la notion de ***nombre*** est *primitive*, et intimement liée à celles d'*ensemble* et de *suite* (ensemble ordonné). Elle recèle deux abstractions fondamentales : le ***cardinal***[1], pour exprimer la ***quantité*** d'éléments d'un ensemble ; l'***ordinal***, pour exprimer la ***position*** (le numéro) d'un élément dans une suite.

■ Dans l'histoire de l'humanité, différents systèmes de numération se sont succédés. Le ***système décimal positionnel***[2] est apparu en Inde vers les IV^e^ et V^e^ siècles (9 chiffres dits *arabes*, puis introduction du 0). Il s'est progressivement imposé dans le monde entier pour sa concision, son absence de limites, son adaptation au comptage manuel et aux calculs écrits, sa facilité d'extension (notations décimale et exponentielle, etc.).



symboles et noms historiques des chiffres dits « arabes »

Depuis les années 1950, avec l'essor de l'informatique et des machines de calcul, est apparue la nécessité de représenter les nombres dans d'autres ***bases*** : en ***binaire*** (**base 2**) d'une part, pour coder l'information au ***niveau machine***, en ***octal*** (**base 8**, aujourd'hui en désuétude) et en ***hexadécimal*** (**base 16**) d'autre part, comme représentations plus concises des nombres en codage de ***haut niveau*** (notamment pour la programmation).

■ Dans les systèmes techniques, la numération, et plus généralement les ***représentations numériques de l'information*** (nombres, caractères, textes, images…) sont implémentées par différentes techniques de ***codage***. La connaissance de leurs caractéristiques (échelle, préfixe, format…) est indispensable à leur utilisation.

I. Éléments de mathématiques pour la numération entière

I.1 Principes de numération entière en base *b*

■ Une ***base*** ***b*** ***de numération*** est un ***ensemble de cardinal b*** ($b \geqslant 2$) dont les éléments sont pris ***dans l'ordre*** parmi les chiffres **0**, **1**, **2**… **9** puis, si $b \geqslant 11$, parmi les lettres **A**, **B**, **C**… qui *valent* respectivement **10**, **11**, **12**… en ***base 10***; ils sont les ***symboles valués*** (ou l'alphabet) de la base. En base ***b***, il n'y a ***pas de symbole de valeur b***.

Pour exprimer la ***valeur b*** en ***base b***, on juxtapose les deux symboles **1** et **0** pour former le ***nombre*** « **10** » (prononcer « ***un zéro*** ») successeur du nombre représenté par le symbole de valeur ***b–1*** (e.g. en base **8**, le chiffre **7**) . Avec deux symboles, on peut former b^2 nombres exprimant les valeurs de 0 à b^2-1 (e.g. en base 8, 64 nombres exprimant les valeurs 0 à 63 qui s'écrivent « 0 » à « 77 »). Au-delà, on juxtapose un 3^e^ symbole ; etc.

■ Un ***nombre entier naturel*** ***x*** s'écrit (on dit aussi « se décompose ») en base ***b*** par une juxtaposition (concaténation) de n symboles de l'alphabet de la base, ordonnés ***de droite à gauche*** :

- en base ***b autre que 10***, pour éviter les confusions, un entier ***x*** est ***indicé*** par ***b***, avec ***x*** ou ***b*** entre parenthèses, i.e. $(x)_b$ ou $x_{(b)}$; sans indice de base, tout entier est implicitement écrit en ***base 10***, la ***base par défaut*** ;
- on appelle ***digit***[3] chaque symbole d_k du nombre, dont l'indice k, de 0 à $n-1$, est le ***rang*** (la position) du digit[4] ;
- on attribue à chaque digit un ***poids*** (facteur multiplicatif) d'expression $(10^k)_b$ – le premier digit (d_0 à droite) et le dernier digit (d_{n-1} à gauche) étant respectivement dits ***de poids faible*** et ***de poids fort*** ; il en résulte que x admet un ***développement polynomial*** **en base** ***b*** qui est la somme discrète de n termes de forme $(d_k.10^k)_b$:

$$x = \underbrace{(d_{n-1}\, d_{n-2} \cdots d_1\, d_0)_b}_{\text{développement en base } b} = \underbrace{d_{n-1} \cdots d_0\,{}_{(b)}}_{\text{(indice de la base)}} = \left(\sum_{k=0}^{n-1} d_k.10^k\right)_b = (d_0.1 + d_1.10 + d_2.100 + \ldots + d_{n-1}.10^{n-1})_b$$

(digits de poids fort … faible ; rang ; autre notation ; expression des poids en base b : $10^0 = 1$, $10^1 = 10$, $10^2 = 100$ … ; somme discrète de puissances de $(10)_b$ coefficientées par les digits)

ex. : nombre à quatre digits développé en base 8 : $(2016)_8 = 2016_{(8)} = (6\times1 + 1\times10 + 0\times100 + 2\times1000)_8 = (6 + 10 + 2000)_8$

nombre à trois digits développé en base 16 : $(40E)_{16} = 40E_{(16)} = (E\times1 + 0\times10 + 4\times100)_{16} = (E + 00 + 400)_8$

1. Du latin *cardo*, qui signifie « gond », « pivot », « axe principal » (qui porte la *charge*).
2. *Décimal* signifie « en base 10 », et *positionnel* signifie « tout nombre est exprimé par la position de ses symboles (les chiffres) ».
3. Anglicisme récent, qui vient du latin *digitus* signifiant « doigt » ; il généralise la notion de *chiffre*.
4. Cet indice k est, comme b, toujours écrit en base *dix*, y compris dans les notations exponentielles comme 10^k employées par la suite.

Plus la base est petite, plus il faut de digits pour écrire (i.e. développer) un nombre. Par récurrence, on montre qu'**en base *b***, sur au plus n **digits**, on peut former b^n ***entiers*** de ***valeurs ordonnées*** de **0** à $b^n - 1$, i.e. $(10^n - 1)_b$.

Remarque : les nombres **0** et **1** ont respectivement le même développement dans toutes les bases.

■ La ***relation d'ordre total*** > qui caractérise l'***aspect ordinal*** des entiers naturels peut être évaluée dans toute base, pour deux nombres ***x*** et ***y***, par comparaison de la valeur de leurs digits respectifs de même rang dans l'ordre décroissant : le plus grand des deux est celui qui a, au plus haut rang, un digit supérieur à celui de l'autre.

Cette relation d'ordre définit, pour chaque entier naturel ***x***, son ***successeur*** $S(x)$ que l'on peut directement déterminer dans toute base ***b*** en remplaçant son digit de poids faible d_0 par :

- le *symbole suivant* dans l'alphabet de la base si $d_0 \neq b - 1$; e.g. $S(26)_8 = (27)_8$;
- le symbole 0 sinon, et auquel cas, on applique le même processus au digit suivant ; e.g. $S(27)_8 = (30)_8$.

De même, si $x \neq 0$, son ***prédécesseur*** $P(x)$ se détermine dans toute base comme usuellement en base dix.

I.2 Calculs arithmétiques en base *b*

■ Toutes les opérations arithmétiques définies en base dix le sont également dans toute autre base.

- L'***incrémentation*** et la ***décrémentation*** d'un entier ***x***, i.e. l'ajout et le retrait à ***x*** du nombre **1** donnent respectivement le ***successeur*** et le ***prédécesseur*** de ***x*** dans l'ordre de la numération dans la base ; formellement, **inc**$(x) = S(x)$ et, si $x \neq 0$, **dec**$(x) = P(x)$.
- L'***addition*** et la ***soustraction*** d'un entier ***y*** à ***x*** sont l'***itération*** y fois respectivement de l'***incrémentation*** et de la ***décrémentation*** de ***x*** ; formellement, $x+y = \mathrm{inc}_{i=1}^{y}(x)$ et si $x \geqslant y$, $x-y = \mathrm{dec}_{i=1}^{y}(x)$.
- La ***multiplication*** de ***x*** par ***y*** est l'***itération*** y fois de l'***addition*** de x ; formellement, $x \times y = \Sigma_{i=1}^{y} x$.
- L'***exponentiation*** de x par y est l'***itération*** y fois de la ***multiplication*** par x ; formellement, $x^y = \Pi_{i=1}^{y} x$.
- La ***division euclidienne*** de ***x*** par $y \neq 0$ est le compte du nombre ***quotient q*** d'***itérations*** de la ***soustraction*** de y à x jusqu'à ce que le ***reste r*** (résultat de la soustraction) soit inférieur à y, ce qu'exprime implicitement l'équation $x = y.q + r$; on note usuellement $q = x \div y$ ou $q = x$ **div** y et $r = x$ **mod** y.

■ Toutes ces opérations sont réalisables directement quelle que soit la base de décomposition des nombres, avec les ***mêmes algorithmes de calcul*** qu'en base 10 (cf. ci-contre, e.g. une addition posée en base 2 avec ***retenues***, qui correspond à 3 + 3 = 6). En revanche, les tables d'addition et de multiplication sont spécifiques à chaque base (cf. ci-contre la table d'addition pour des nombres à un ou deux digits en base 2).

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

+	0	1	10	11
0	0	1	10	11
1	1	10	11	100
10	10	11	100	101
11	11	100	101	110

I.3 Conversions usuelles

En se limitant aux bases 2, 10 et 16, il n'y a que **six conversions possibles** ; toutes sont fréquemment utilisées. La **base 10** joue un rôle central car les calculs y sont familiers ; fondamentalement, on a $(10)_b = (b)_{10}$.

■ Un entier naturel ***x*** de n digits $(d_k)_b$ vaut par ***développement polynomial en base 10*** la somme des produits de la valeur v_k de ses digits d_k fois la valeur de leur poids b^k.
Rem. : on a $d_k = v_k$ sauf en base 16 (A = 10, etc.)

$$x = (d_{n-1} \dots d_0)_b = \sum_{k=0}^{n-1} v_k . b^k = v_0.b^0 + v_1.b^1 + v_2.b^2 + \dots + v_{n-1}.b^{n-1}$$

(n digits en base b ; rangs ; valeurs des digits et valeurs des poids toujours exprimées en base 10 ; développement polynomial en somme discrète de puissances de b)

ex. : $(2016)_8 = 6\times8^0 + 1\times8^1 + 0\times8^2 + 2\times8^3 = 6 + 8 + 1024 = 1038$

$(40E)_{16} = 14\times16^0 + 0\times16^1 + 4\times16^2 = 14 + 0 + 1024 = 1038$

■ Soit un entier naturel ***x*** écrit en base 10. Sa conversion en n digits $(d_k)_b$ en base ***b*** nécessite un processus itératif de $n-1$ ***divisions*** de ***x*** par ***b*** dont les ***restes successifs*** et le ***dernier quotient*** (inférieur à ***b***) sont, dans cet ordre, les valeurs des $(d_k)_b$. Par exemple, en divisant $x = 1038$ par 8 trois fois de suite, on retrouve la décomposition $(2016)_8$ de l'exemple précédent. En effet, les divisions ci-contre se traduisent par $1038 = (2\times8 + 0)\times8 + 1)\times8 + 6 = 2\times8^3 + 0\times8^2 + 1\times8 + 6 = (2016)_8$.

De même, $1038 = (4\times16 + 0)\times16 + 14) = 4\times16^2 + 0\times16 + 14 = (40E)_{16}$ car $14 = (E)_{16}$.

```
1038 | 8
v0 → 6 | 129 | 8
v1 → 1 | 16 | 8
v2 → 0 | (2) | ← v3
```

$2 < 8 \Rightarrow$ fin du calcul

■ Entre les bases 2 et 16, les conversions sont faciles : $(1\ \text{digit})_{16} = (4\ \text{digits})_2$ car $\mathbf{16 = 2^4}$. On peut donc ***convertir les nombres par parties*** à l'aide du tableau ci-dessous, e.g. $(100\ 0111\ 1011)_2 = (47B)_{16}$.

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

Remarque : les nombres en base 2 se mémorisent facilement en s'aidant de la parité (un nombre pair se termine par 0) et des repères $(10^k)_2 = 2^k$ en gras ci-dessus. Il est donc utile de connaître également le tableau ci-dessous.

k	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
2^k	1	2	4	8	16	32	64	128	256	512	1 024	2 048	4 096	8 192	16 384	32 768	65 536

II. Codage des nombres entiers

Un ***codage*** – ou simplement ***code*** – est une ***fonction injective***[5] de ***représentation de l'information***, implémentée dans une ***machine*** (i.e. la fonction « traiter » d'un système) : à toute ***valeur*** abstraite d'un type donné (nombre, caractère, couleur, etc.), elle attribue un ***code*** *unique*. On appelle ***décodage*** la fonction réciproque qui, pour un code donné, retourne la valeur correspondante.

valeurs	*codes*
0 →	000
1 →	001
2 →	010
3 →	011
4 →	100
⋮	⋮

Pour un même type de valeurs, différents codages peuvent exister : les entiers se codent en *binaire pur*, en *BCD* ou en *Gray* ; les caractères en *ASCII*, en *Unicode*… (cf. infra). Chaque codage a son intérêt.

II.1 Caractéristiques d'un codage : échelle, format, préfixe, etc.

■ Au niveau physique, l'information est toujours mémorisée et exploitée sous une forme dite ***logique*** ou ***binaire*** par ***deux*** états électriques possibles d'un circuit élémentaire de mémoire. La caractéristique principale d'un codage est son ***échelle*** (on dit aussi son *format*) : le ***volume mémoire nécessaire au stockage d'une valeur***.

L'unité élémentaire d'échelle est le ***bit***, acronyme anglais de ***binary digit*** : c'est le volume pour mémoriser un digit en base 2. Les microprocesseurs ont été développés avec une capacité de traitement de ***multiples de 8 bits*** à la fois, donc les échelles usuelles sont l'***octet*** (8 bits, en anglais ***byte***, également employé comme unité), le ***mot***[6] (16 bits, ***word***), le ***double mot*** (32 bits, en abrégé ***dword***) et le ***quadruple mot*** (64 bits, ***qword***).

■ L'information est « manipulée » au niveau physique par le système d'exploitation de la machine. À plus haut niveau (programme de commande du système), toute donnée est mise en œuvre conformément aux règles du ***langage de programmation*** utilisé : elle doit être ***déclarée*** avec un ***type*** reconnu par le langage (e.g. booléen, entier, etc.), comme *constante* ou *variable*.

Dans les instructions du programme, les valeurs d'une variable doivent s'écrire selon une ***morphologie*** (préfixe, caractères autorisés et règles d'écriture, bornes, etc.) compatible avec le type choisi. Outre la base 10 par défaut, on peut, dans la plupart des langages, écrire les valeurs en bases 2 et 16 en ajoutant un ***préfixe*** idoine (cf. tableau ci-contre).

base	langage C, C++ Java, Python…	norme CEI 61131-3[7]
2	**`0b`** ou **`0B`**	**`2#`**
16	**`0x`** ou **`0X`**	**`16#`**

Implicitement, le type d'une donnée suppose aussi un ***format*** de codage qui ne consiste pas seulement en son échelle, mais aussi à la ***façon dont chaque code correspondent à leur valeur***. Pour les entiers, on utilise principalement ***trois formats*** : le ***binaire pur*** (avec une spécificité pour les valeurs signée ±), le ***binaire réfléchi*** et le format ***décimal codé binaire***. On indique le format employé pour un entier x par un indice : $(x)_{2\pm}$, $(x)_{BCD}$, etc.

II.2 Codage des entiers en binaire pur

Le format binaire pur (on dit aussi *naturel*) est celui usité ***par défaut*** pour le codage des entiers. La plupart des langages de programmation distinguent les types d'entier selon qu'ils sont signés (déclarés en général **`int`** pour *integer*) ou non signés (déclarés **`uint`** pour *unsigned integer*). Les échelles employées sont soit le ***mot*** (2^{16} = 65 536 valeurs – type **`short int`**), soit le ***double mot*** (2^{32} = 4 294 967 296 valeurs – type **`long`**[8] **`int`**).

5. Chaque code « image » peut avoir au plus une seule valeur « antécédent » ; mais un code peut ne correspondre à aucune valeur.
6. Par défaut, ce terme désigne en général tout groupement de bits en mémoire ; e.g. on parle de « mot de 8 bits ».
7. Norme de la *Commission Électrotechnique Internationale* (CEI) pour la programmation des automates industriels.
8. Attention, en Python, **`long`** est un type d'entier à échelle libre, limitée par la capacité mémoire de la machine.

■ Pour les types à **valeurs non signées**, le codage en binaire pur applique directement le principe de la numération en base 2, où le poids du bit $(d_k)_2$ de rang k vaut 2^k ; le ***bit de poids fort*** et le ***bit de poids faible*** sont respectivement désignés ***MSB*** et ***LSB*** (de l'anglais *most significant bit* et *least significant bit*).

Exem $1515 = 1024 + 256 + 128 + 64 + 32 + 8 + 2 + 1$
$= 2^{10} + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 + 2^1 + 2^0$

rang	15 MSB	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 LSB
bit	0	0	0	0	0	1	0	1	1	1	1	0	1	0	1	1

En langages C, C++, Java… 1515 peut s'écrire aussi `0b10111101011` ou `0x5EB` (`0` non significatifs omis).

■ Pour les types à **valeurs signées**, le bit de poids fort est diminué d'un rang (à $n-2$) pour réserver le bit de plus haut rang ($n-1$) au codage du signe, que l'on le nomme ***bit de signe***. Pour la compatibilité avec le format non signé, le bit de signe s s'écrit `0` (et vaut 0) si la valeur est ***positive ou nulle***, et s'écrit `1` (et vaut 1) sinon.

De plus, on opère un ***codage asymétrique des valeurs absolues*** :

- pour les **valeurs positives et nulle**, le codage est ***comme non signé*** ; à l'échelle n, le maximum est $\boxed{2^{n-1}-1}$;
- pour les **valeurs négatives**, le codage est ***complémenté à 2^n*** à l'échelle n ; le minimum est $\boxed{-2^{n-1}}$.

Ce codage revient à donner au bit de signe s un ***poids négatif*** égal à -2^{n-1} ; cf. la formule de la valeur d'un entier x de code $(s\,d_{n-2}\ldots d_0)_{2\pm}$:

$$x = (s\,d_{n-2}\ldots d_0)_{2\pm} = \left(\sum_{k=0}^{n-2} v_k.2^k\right) - s.2^{n-1}$$

$(1011)_{2\pm} = 1\times2^0 + 1\times2^1 + 0\times2^2 - 1\times2^3 = 1 + 2 - 8 = -5$

Ci-dessous, par exemple, le codage des valeurs signées pour $n = 4$.

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

Dans ce format, ***la fonction successeur reste opérationnelle*** en binaire, même à la transition de −1 à 0, moyennant la ***cyclicité*** du code (retour à `0000`). Les algorithmes usuels des opérations arithmétiques sont donc applicables ; e.g. on peut vérifier que $x + (-x) = 0$ et que $x - (-x) = 2x$ en posant ci-contre les opérations avec $x = 3$.

```
  1 1 1          1
    0011         0011
+   1101     -   1101
1 1 0000         0110
```

II.3 Codage BCD (*binary coded decimal*)

■ Le format ***décimal codé binaire*** est le plus souvent désigné par son sigle anglais ***BCD*** (*binary coded decimal*). Il consiste à ***coder séparément chaque digit décimal*** (chiffre) d'un entier sur un ***quartet*** (4 bits consécutifs), afin de faciliter les opérations sur ces digits, en particulier les procédures d'affichage (blocs d'afficheurs digitaux, cf. ex. ci-contre). Il en résulte aussi une grande simplicité de codage et décodage : il suffit de connaître l'écriture des chiffres décimaux en base 2 (cf. tableau p. 3). En contrepartie, le codage est moins compact en volume, puisque pour chaque quartet, les codes de `1010` (10) à `1111` (15) ne sont pas utilisés.

Par exemple, 1515 se code en BCD sur 4×4 = 16 bits (les trois `0` de gauche sont significatifs) alors qu'il en faut en théorie seulement 11 en binaire pur.

base 10	1	5	1	5
BCD	0001	0101	0001	0101

■ Le codage BCD est parfois implicite à certains types de données. Par exemple, dans les langages de programmation de la norme CEI 61131-3, le type *date* dont les valeurs sont saisies sous la forme `D#`*aaaa*`-`*mm*`-`*jj* (préfixe `D#`, 4 chiffres de l'année, tiret, 2 chiffres du mot, tiret, 2 chiffres du jour) sont codées dans la mémoire de l'automate au format BCD sur deux mots (8 chiffres sur 4 bits, donc au total 32 bits). Connaissant ce format, on peut aisément manipuler un seul des huit chiffres de la date pour exécuter par exemple un défilement cyclique de réglage, un clignotement, un test de valeur, etc.).

II.4 Codage binaire réfléchi (code *Gray*[9])

■ Le format ***binaire réfléchi*** (en anglais *reflex code*) emploie un codage binaire ***non pondéré*** et ***non décimal*** mais ***adjacent*** et ***cyclique*** : les codes de ***deux valeurs successives ne diffèrent que d'un seul bit***, y compris à la transition entre la première et la dernière valeur. Les valeurs codées peuvent être ***signées ou non***.

La simultanéité de 2 changements de bits ne pouvant être garantie, la propriété d'adjacence est recherchée pour les codeurs absolus de position, qu'ils soient linéaires ou rotatifs (la cyclicité requise aussi) car elle évite l'apparition de valeurs instables lors d'un changement de valeurs.

9. Du nom de Franck GRAY, ingénieur chez Bell Laboratories, qui en déposa le brevet en 1953 ; mais ce codage était déjà utilisé dans d'autres contextes auparavant (codage des lettres Baudot et résolution de problèmes divers comme celui des tours de Hanoï).

■ Le tableau ci-contre illustre le codage Gray d'entiers naturels sur n = 3 bits, notés g_0 à g_2. Les axes de symétrie repérés en pointillés rouges permettent de comprendre le principe de construction du tableau ***par récurrence*** à partir de la colonne g_{n-1} du bit de plus haut rang :

- on code la moitié supérieure de la colonne à **0**, la moitié inférieure à **1** ;
- on ignore la moitié inférieure du tableau et on applique le même codage pour la colonne suivante (moitié supérieure des lignes restantes à **0**, moitié inférieure à **1**) ;
- lorsque la colonne du bit de rang 0 est codée (seulement sur les 2 premières lignes), on complète par symétrie les parties du tableau non codées.

Et pour étendre un codage Gray sur n bits à l'échelle n + 1, on rajoute 2^n lignes symétriques sous celles déjà existantes, puis une nouvelle colonne g_n codée à **0** en moitié haute, **1** en moitié basse.

	g_2	g_1	g_0
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

III. Autres codages

L'information mise en œuvre dans les systèmes techniques ne se limite pas aux nombres entiers. Pour reconstituer numériquement des grandeurs physiques, et approcher les nombres réels, on a besoin de représenter des *nombres décimaux* ; leur codage sera abordé dans un autre cours. L'information se présente aussi sous d'autres formes comme les ***textes*** ou les ***images*** ; on introduit ici leurs techniques de codage.

III.1 Initiation au codage des caractères

■ Sans entrer dans des considérations de typographie et de mise en page, l'information textuelle se présente sous forme d'éléments qu'on appelle ***chaînes de caractères***. La notion de ***caractère*** doit être entendue au sens large : il peut s'agir de *lettres* (majuscules ou minuscules), de *chiffres*, de *signes de ponctuation*, de *symboles divers*, d'*espaces* et de caractères dits *« de contrôle »* (tabulation, retour à la ligne, fin de texte, etc.).

■ Le développement de l'informatique ayant commencé aux États-Unis, le premier standard de codage des caractères s'est limité aux lettres non accentuées de l'alphabet latin. Il s'agit du codage ***ASCII***[10] défini sur une échelle de **7 bits**, permettant de coder un jeu de 2^7, soit **128 caractères**. De part la structure des microprocesseurs, chaque caractère est en réalité mémorisé sur un octet. L'échelle du codage ASCII a donc été rapidement étendu à 1 bit de plus (jeu de 256 caractères). Néanmoins, le codage sur 7 bits reste le plus largement compatible entre machines ; il est donc encore utilisé dans de nombreux langages de programmation. Le tableau ci-dessous donne le code 7 bits de chaque caractère par son LSQ et son MSQ (*least & most significant quartet*) en bases 2 et 16 ; e.g. la lettre « A » (caractère n° 65) est codée **0100 0001**$_{(2)}$ ou **41**$_{(16)}$.

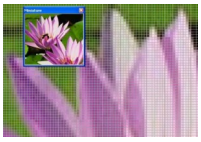
MSQ \ LSQ		0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000	0	NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	HT 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
0001	1	DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
0010	2	SP 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
0011	3	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
0100	4	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
0101	5	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
0110	6	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
0111	7	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127

10. *American standard code for information interchange*, conçu par Robert BEMER en 1961, et enregistré depuis 1975 comme variante de la norme internationale ISO/CEI 646-02 -06.

III.2 Initiation au codage des images numériques

■ Il existe de nombreux formats de codage d'image, qu'on peut classer d'abord en ***deux grandes familles*** :

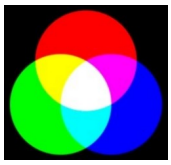
- les formats ***vectoriels***, constitués d'objets géométriques définis mathématiquement ; utilisés dans des logiciels de CAO ou de graphisme, ils garantissent la possibilité d'agrandir les images à l'écran sans perte de qualité, mais ils ne sont pas adaptés à la numérisation des photographies ;
- les formats ***matriciels*** (en anglais *bit map* – littéralement carte de bits), constitués d'une matrice de ***pixels*** (acronyme anglais de *picture element*) juxtaposés en lignes et colonnes régulières ; a contrario des formats vectoriels, ils sont bien adaptés aux photographies, mais perdent en qualité à l'agrandissement.



Les caractéristiques principales d'une image matricielle (ou d'un écran) sont ses ***dimensions*** (largeur et hauteur) et sa ***définition***, laquelle s'exprime par le produit du nombre de colonnes et de lignes de pixels (e.g. 1920 × 1080 = 2 073 600 pixels pour le standard vidéo « full HD » en proportion 16/9e). On peut caractériser la ***qualité*** d'une image en faisant le ***ratio*** de sa définition par ses dimensions, appelé ***résolution*** et qui s'exprime en nombre de *pixels par pouce* (ppp), en anglais *pixel per inch* (ppi). Ainsi, une résolution de 220 ppp (haute qualité) est obtenue par des pixels de 0,1 mm de côté ; avec des dimensions de 16 × 9 pouces (soit une diagonale d'environ 18 pouces), il faut une définition de 3520 × 1980 pixels pour atteindre cette résolution.

■ Parmi les formats matriciels, le plus élémentaire est le format ***bmp*** développé par Microsoft et IBM. En général non compressé, il concatène dans un fichier le code de chaque pixel ***de gauche à droite et de bas en haut*** (ligne par ligne). Ainsi, pour une image full HD monochrome où chaque pixel est codé par un bit, il faut donc 2 073 600 bits, soit 2 073 600/(1024×8) ≈ 253 ko (kilo-octets) – car usuellement, 1 ko = 1024 o ![11]

Le format *bmp* permet aussi de coder les images en couleur, en se basant sur le principe de la ***synthèse additive trichrome rouge vert bleu*** (RVB, en anglais RGB) : en superposant des ***sources de lumière*** de ces trois ***couleurs primaires*** et en faisant varier leur intensité respective, on peut engendrer autant de nuances de couleurs que de combinaisons permises par la gradation d'intensité des sources. Plus les sources superposées sont intenses, plus la lumière est claire.



Le format *bmp* **24 bits** permet de coder 2^{24}, soit **16,8 millions de couleurs** possibles par pixel. L'intensité respective des trois sources de couleurs primaires est codée dans l'ordre **bleu-vert-rouge** sur **8 bits** chacune, c'est-à-dire qu'elle peut varier sur 2^8, soit 256 niveaux d'intensité, codés en hexadécimal de **00** (intensité nulle, absence de lumière) à **FF** (intensité maximale). Le code d'un pixel est donc exprimé par un nombre à 3 × 2= 6 digits hexadécimaux de la forme ***bbvvrr*** : digits ***bb*** pour le bleu, digits ***vv*** pour le vert, et digits ***rr*** pour le rouge (cf. ci-dessous un nuancier de couleurs remarquables avec une gradation en quart d'échelle **00**, **40**, **80**, **C0**, **FF**).

bleu	***vert***	***rouge***	***cyan***	***magenta***	***jaune***	*nuances mixtes*		
FF C0 C0	C0 FF C0	C0 C0 FF	FF FF C0	FF C0 FF	C0 FF FF	FF C0 80	C0 80 FF	C0 FF 80
FF 80 80	80 FF 80	80 80 FF	FF FF 80	FF 80 FF	80 FF FF	FF C0 40	C0 40 FF	C0 FF 40
FF 40 40	40 FF 40	40 40 FF	FF FF 40	FF 40 FF	40 FF FF	C0 80 40	80 40 C0	80 C0 40
FF 00 00	00 FF 00	00 00 FF	FF FF 00	FF 00 FF	00 FF FF	80 40 00	40 00 80	40 80 00
C0 00 00	00 C0 00	00 00 C0	C0 C0 00	C0 00 C0	00 C0 C0	FF 80 C0	80 C0 FF	80 FF C0
80 00 00	00 80 00	00 00 80	80 80 00	80 00 80	00 80 80	FF 40 C0	40 C0 FF	40 FF C0
40 00 00	00 40 00	00 00 40	40 40 00	40 00 40	00 40 40	C0 40 80	40 80 C0	40 C0 80
00 00 00	40 40 40	80 80 80	C0 C0 C0	E0 E0 E0	FF FF FF	80 00 40	00 40 80	00 80 40
noir		***gris moyen***			*blanc*			

Remarques :

- les codes des nuances de gris ont des valeurs d'intensité égales pour les 3 sources (**00** → noir, **FF** → blanc) ;
- à partir d'une couleur primaire (e.g. le bleu : **FF 00 00**), on diminue l'intensité de sa source pour assombrir la nuance (**80 00 00**) ; on augmente l'intensité des deux autres sources pour l'éclaircir (**FF 80 80**).

Le codage de la couleur multiplie par l'échelle le volume occupé en mémoire. Pour l'exemple précédent (image full HD), il faut donc 2 073 600×24 =49 766 400 bits, soit 49 766 400/(1024×1024×8) ≈ 5,9 Mo ![11] D'où la nécessité d'utiliser d'autres format mettant en œuvre des algorithmes de compression (jpeg, png…).

11. Usage de Microsoft, non conforme au Système International d'unités, qui définit 1 **kibi** = 2^{10} = 1024 (*kilo-binaire*, abrégé **ki**) ; on doit donc plutôt employer 1 kio = 1024 o, au lieu de 1 ko ; de même, on définit 1 Mebi = 2^{20} = 1024×1024, donc 1 Mio = 1 048 760 o.