Centre de Réadaptation de Mulhouse

Accès aux Bases et Compétences

DEVELOPPEMENT INFORMATIQUE

Numération

Représentation de l'information

Dossier d'informations

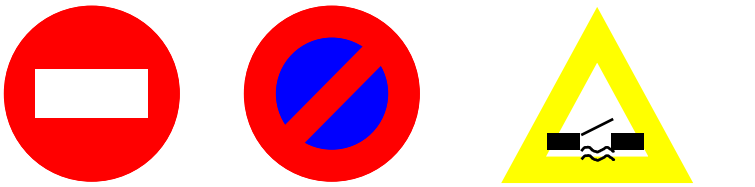
1. Introduction aux codifications

1.1. Définition :

Un code est un système de représentation d'informations permettant une interprétation sans ambiguïté de ladite information

1.2. Exemple :

Les panneaux de signalisation routière



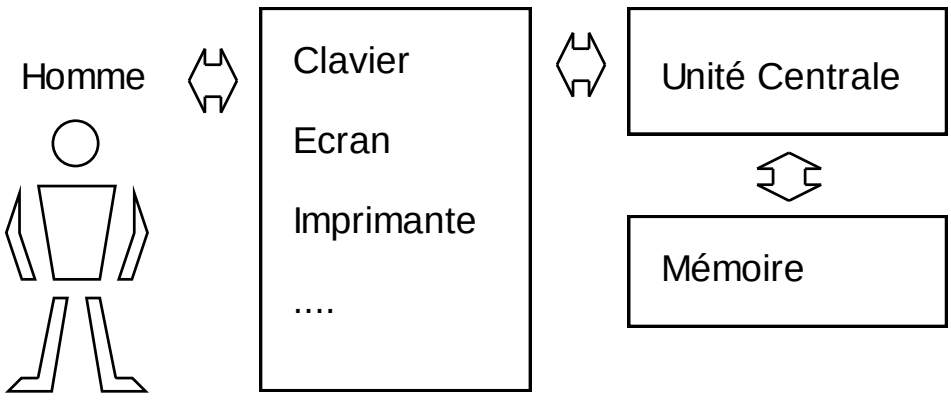
...

Ce code est dit iconographique car il fait appel à des symboles graphiques pour représenter l'information.
L'inconvénient majeur de ce type de représentation est qu'il nécessite un nouveau symbole pour chaque nouvelle information à représenter.

Le " langage " employé a un certain vocabulaire.

1.3. Codifications informatiques

Si l'on rappelle le schéma fonctionnel de l'ordinateur



On doit établir un certain nombre de *communications* entre

l'**utilisateur** et le **cœur du système informatique**

L'ensemble de ces communications (échanges) est basé sur un échange de caractères alphanumériques qui sont codés dans un

un *code* **alphabet**

Concernant les traitements informatiques de l'information, nous distinguerons les notions de codification et de codage (bien que ces 2 termes soient souvent indifféremment utilisés)

Codification: Consiste à affecter une référence (un code) à un ensemble d'objets traités par un ordinateur.

Exemples: Un produit est représenté par un numéro
Un stagiaire est représenté par un code
Un individu est codifié par son numéro de sécurité sociale

Codage: Représentation des informations dans un code donné

Ce dossier est essentiellement consacré au codage (représentation) des informations au sein de l'ordinateur; cependant avant d'aborder cette notion, nous consacrons un court paragraphe à la codification

1.4. Codification de l'information

Dans tous les cas elle devra être étudiée au mieux pour assurer une *identification unique et sans ambiguïté* de l'information; dans la mesure du possible elle permet des *gains de place mémoire et de temps* et offre des possibilités de *contrôle de validité.* On distingue plusieurs types de codes:

Le *code séquentiel:* Le plus simple, il associe à chaque information un code défini de façon consécutive.

Exemple : les numéros de client numérotés de 1 à N, N étant le dernier client enregistré, le prochain enregistré aura le numéro N + 1.

Le *code séquentiel par tranches:* un élément supplémentaire d'information est inclus dans le code. Par exemple dans une bibliothèque les références 1 à 9999 concernent l'Electricité, les références de 10000 à 19999, l'Automatisme, les références 20000 à 29999, la Thermodynamique, ...

Le *code significatif:* constitué d'une association de chiffres ou/et de lettres permettant la désignation d'une information complexe. Le meilleur exemple est le numéro de sécurité sociale français constitué de plusieurs tranches ou champs ayant chacun une signification précise. On n'a plus la notion de références consécutives.

Le *code contrôlable* : identique en principe aux précédents il ne fait que rajouter une clé de contrôle déterminée en fonction des carctères constituant le code et utilisée pour vérifier la présence ou plutôt l'absence d'erreur.

2. Codage des informations

2.1. Les systèmes de numération

Lorsqu'il s'agit de dénombrer un ensemble d'éléments plusieurs solutions peuvent être utilisées. On peut par exemple tracer N tirets verticaux alignés côte à côte pour numéroter les N objets à compter; si l'on a 10 objets à compter, c'est acceptable mais si vous désirez compter le nombre de français âgés de 10 à 20 ans, cela devient inextricable et si l'on veut faire un calcul simple sur ces "nombres", ou plutôt ces tas de tirets cela devient impossible.
Historiquement, des notations plus condensées sont rapidement apparues. On peut citer la *notation romaine* où les seuls symboles de notations sont: I, V, X, L, C, D, M ce qui donne des nombres relativement compacts mais si l'on veut ajouter MCMLXXXVII à XCIX le problème n'est guère facile à résoudre.
Le système le plus connu et couramment utilisé est le système décimal qui nous vient des Arabes qui le tenaient probablement de l'Inde : c'est un **système de numérotation** simple et très commode pour les calculs arithmétiques.

Rappelons la structure d'un nombre décimal:

$$125= 1*100 + 2*10 + 5*1 = 1*10^2 + 2*10^1 + 5*10^0$$

Les éléments du nombre **1, 2 et 5** sont les **coefficients**
10 est la **base du système de numération**
2, 1 et 0, correspondant à la position de chaque élément, représente le **poids** de chaque chiffre

D'une façon générale, dans un système de numération n un nombre d'éléments N est représenté par la formule

$$N=a_p n^p+a_{p-1}n^{p-1}+\ldots+a_1 n+a^0$$

où : a_0 = nombre d'objets, inférieur à n n'ayant pu être classés dans un paquet de n éléments,
a_1 = nombre de paquets (<n) de n objets, qui n'ont pu être classés dans un paquet de n^2 objets, etc.

n est appelé la **base du système** de **numération**.
On remarque qu'il faut n-1 symboles (de 1 à n-1) pour représenter les coefficients a_p . On leur adjoint un $n^{ième}$ symbole noté 0 (zéro) dont le rôle est fondamental : s'il n'y a pas de groupe n éléments, on remplacera le terme a_i par 0.
Dans un but de simplification, on convient de transcrire le nombre N sous la forme:

$$\mathbf{N = a_p\ a_{p-1}\ldots\ a_2\ a_i\ a_o}$$

La position i occupée par le symbole est appelé le **poids** attaché à ce symbole.

Dans le **système décimal,** la base est prise égale à 10 (sûrement due à la similitude des 10 doigts de la main ?), les différents coefficients représentent pour a_0 le chiffre des unités (poids 1), pour a_1 celui des dizaines (10 unités, poids 10), a_2 celui des centaines (poids 100), etc.

Remarquons qu'en base 10 il y a 10 symboles unitaires différents: 1, 2, 3,..., 8, 9 et 0 conformément à la règle énoncée ci-dessus qui stipule que pour une base n il faut n symboles différents (y compris le 0).

Muni de tous ces éléments il devient facile d'établir les notations dans un sytème de numération à base quelonque.
Dans tous les cas lorsque la base n'est pas implicite ou évidente, on note la valeur de cette base en indice après le nombre.

Exemples: 10100100_2 en système binaire (base deux)
12842_{10} en système décimal (base 10)
12842_{12} en système duodécimal (base 12).

Remarques:1. Deux nombres peuvent s'écrire sous la même forme dans deux bases différents, MAIS ils n'auront pas la même valeur!

2. On veillera à ne pas confondre les termes numération et numérotation, ce dernier désignant l'opération consistant à numéroter, attribuer un nombre à un objet et non à définir le nombre lui-même.

2.2..Le système binaire

C'est le système à base 2 qui se prête à une utilisation par les machines et à fortiori par les calculateurs électroniques où tout fonctionne sur le principe de deux états d'équilibre stables. Les symboles utilisés pour représenter ces 2 états sont le 0 et le 1 ce qui permet d'établir la correspondance décimal-binaire suivante:

1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Ici les poids sont respectivement de droite vers la gauche: 0,1,2,3,4,5,6, 7, ...

On désigne les poids de droite par les **poids faibles** (**LSB** en anglais pour Lost Significant Bit) alors que ceux de gauche sont dits **poids forts.** (**MSB** pour Most Significant Bit)

Exemple: traduire en décimal le nombre binaire N = 100101

Prenons le nombre et indiquons les poids:

1	0	0	1	0	1	←nombre
5	4	3	2	1	0	← poids

Ce qui donne:

N = 1. 2^5 + 1.2^2 + 1.2^0, les autres coefficients étant nuls. Soit N = 32 + 4 + 1 = 37_{10}

d'où le résultat: $100101_2 = 37_{10}$

Les règles de calcul arithmétique sont les suivantes:

Addition		Multiplication
0 + 0	=0	0 x 0 0
0 + 1	=1	0 x 1 0
1 + 0	=1	1 x 0 0
1 + 1	=10	1 x 1 1

Exemples d'opérations simples, utilisant les mêmes principes qu'en décimal:

```
   101101
 x    101
--------
   101101
 101101
--------
 11100001
```

```
101101 | 11
-------+----
  101  | 1111
  100  |
   011 |
    00 |
```

La **conversion binaire-décimal** utilise simplement la formule de définition vue plus haut

$$N=a_p2^p+a_{p-1}2^{p-1}+\ldots+a_22^2+a_12+a_0$$

On peut même envisager la notation en binaire de nombres fractionnaires si l'on utilise les *puissances négatives de la base* sachant que:

$$a^{-n}=\frac{1}{a^n}$$

ce qui permet en base 2 d'obtenir:

2^{-1}	=	0,5	
2^{-2}	=	0,25	etc.

On peut envisager la notation binaire suivante d'un nombre décimal non entier : $1101,11_2$ ce qui donne en notant les poids au dessus des bits:

3	2	1	0	-1	-2						
1	1	0	1,	1	1	=	1	x	2^3	=	8
						+	1	x	2^2	=	4
						+	1	x	2^0	=	1
						+	1	x	2^{-1}	=	0,5
						+	1	x	2^{-2}	=	0,25
										=	$13,75_{10}$

Il est intéressant de connaître la table des puissances de 2:

2^{-n}	n	2^n
1	0	1
0,5	1	2
0,25	2	4
0,125	3	8
0,0625	4	16
0,03125	5	32
0,015625	6	64
0,0078125	7	128
0,00390625	8	256
0,001953125	9	512
0,0009765625	10	1024
0,00048828125	11	2048
0,000244140625	12	4096
0,0001220703125	13	8192
0,00006103515625	14	16384
0,000030517578125	15	32768
0,0000152587890625	16	65536

Conversion décimal - binaire : on sépare la partie entière de la partie fractionnaire et on applique la méthode générale à chacune de ces deux parties

Exemple:

Soit à convertir en binaire le nombre décimal 125,625

1) Partie entière: 125. On divise le nombre par le nombre 2 autant de fois qu'il est possible, les restes successifs étant les poids binaires obtenus dans l'ordre des puissances croissantes

125	:	2	=	62 +	1	0
62	:	2	=	31 +	0	1
31	:	2	=	15 +	1	2
15	:	2	=	7 +	1	3
7	:	2	=	3 +	1	4
3	:	2	=	1 +	1	5
1	:	2	=	0 +	1	6
						↑Poids
					↑Restes	

la partie entière en binaire est : 1 1 1 1 1 0 1

2) Partie fractionnaire : 0,625. On la multiplie par 2, la partie entière obtenue représentant le poids binaire et la partie fractionnaire étant à nouveau multipliée par 2, etc.

0,625	x	2	=	1,250	Poids binaire 1 x 2^{-1}
0,250	x	2	=	0,500	Poids binaire 0 x 2^{-2}
0,500	x	2	=	1,000	Poids binaire 1 x 2^{-3}

le résultat est donc 0, 1 0 1

d'où le résultat final: $125{,}625_{10} = 1111101{,}101_2$

2.3. Base octale et base hexadécimale

Si la représentation binaire est très pratique et commode pour le fonctionnement des calculateurs il n'en est pas de même pour l'utilisateur qui préfère les écritures condensées plutôt que ces suites interminables de 1 et de 0. C'est pourquoi on a pris pour convention de grouper les bits par groupes de 3 ou 4 ce qui donne respectivement le système octal (base 8) et le système hexadécimal (base 16)

En base 8 point n'est besoin de créer de nouveaux symboles par rapport à la numérotation décimale ; par contre, en hexadécimal, base 16, il a fallu créer 6 nouveaux symboles qui ne sont autres que les lettres majuscules A, B, C, D, E et F pour représenter les valeurs décimales 10 à 15. Finalement on a les différentes écritures suivantes en base 2, 8, 10 et 16.

Décimal	Binaire	Octal	Hexadécimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Les représentations octales et hexadécimales d'un nombre se déduisent de sa représentation binaire par décomposition en groupes de 3 ou 4 bits et en remplaçant ces groupes par le chiffre correspondant. Ainsi le nombre binaire

10110110111 s'écrira également 2667_8 ou $5B7_{16}$ selon les décompositions suivantes:

10	110	110	111	
2	6	6	7	pour l'octal
101	1011	0111		
5	B	7		pour l'hexadécimal

l'arithmétique octale et hexadécimale suivant les mêmes règles que la numération binaire. Les principes de conversion sont identiques.

2.4. Le bit

Un système de numération comporte n états d'équilibre; chaque état correspondant à une valeur de l'information est appelé **digit**. Le système binaire comporte 2 états stables. Une information binaire sera donc une alternative : VRAI-FAUX , OUI-NON, … Ces 2 états sont notés conventionnellement par 1 et 0.
Un **bit** (binary digit) est la plus petite unité d'information manipulable par une machine. Un bit peut prendre la valeur 0 ou 1.

Avec 2 bits, on peut représenter 4 états différents (2^2): 00-01-10-11
Avec 3 bits, on peut représenter 8 états différents (2^3): 000-001-010-011-100-101-110-111
Avec 8 bits, on peut représenter 256 états différents (2^8): de 00000000 à 11111111. Un ensemble de 8 bits est un **octet**.

2.5. Taille des informations dans un calculateur

Au moment de la conception d'un calculateur il est un problème de taille à résoudre : la *taille des informations* manipulées par l'ordinateur, grandeur exprimée en nombre de bits. On a distingué longtemps suivant leur concept de base : le mot ou le caractère.

Le **caractère** : concept répondant au besoin de traiter les différents caractères alphanumériques. Un caractère (lettre, chiffre, signe de ponctuation …) est représenté sur un **octet** (8 bits), ce qui donne 256 caractères différents.

Le **mot** : est un nombre de bits supérieurs à 8 pouvant varier jusqu'à 64. C'est l'unité d'information traitée par la machine, mot qui est traité d'un bloc que ce soit lors de l'accès mémoire ou lors du traitement interne. On parle ainsi de *machines 16, 32 ou 64 bits* pour les cas les plus fréquents : 16 ou 32 bits sont du domaine des mini-ordinateurs alors que les 64 bits sont déjà de gros calculateurs (en général).

3. Représentation d'un nombre

On appelle représentation d'un nombre la façon selon laquelle il est décrit sous forme binaire. La représentation des nombres sur un ordinateur est indispensable pour que celui-ci puisse les stocker, les manipuler. Toutefois le problème est qu'un nombre mathématique peut être infini (aussi grand que l'on veut), mais la représentation d'un nombre doit être faite sur un nombre de bits prédéfini. Le nombre de bits prédéfini dépend du type du nombre représenté.

3.1. Représentation d'un entier naturel

Un entier naturel est un entier positif ou nul. On distingue généralement (en particulier dans le domaine des PC) les **entiers courts** et les **entiers longs**:

Les entiers courts sont codifiés sur 16 bits
Les entiers longs sont codifiés sur 32 bits

3.2. Représentation des nombres négatifs

La solution la plus simple consiste à associer un bit pour le signe (ce sera le bit de poids fort) les autres bits représentant la valeur absolue du nombre. La convention généralement adoptée est la suivante :

0 ⇨ Signe +
1 ⇨ Signe -

Ainsi 011011 représentera $+27_{10}$ et 111011 représentera -27_{10}

Cette représentation bien que très pratique nécessite un traitement spécial du signe et donc des circuits différents pour l'addition et la soustration. Cet inconvénient est contourné si l'on utilise une notation des nombres négatifs sous forme complémentée. On distingue le **complément vrai** ou **complément à 2** en binaire (à 10 en décimal) et le **complément**

restreint ou **complément à 1** en binaire (à 9 en décimal). Le complément restreint est obtenu en soustrayant chaque digit de (n-1) pour une base n; le complément vrai est obtenu en ajoutant 1 au complément restreint. Ces principes font qu'en binaire le complément restreint s'obtient par substitution des 0 par des 1 et vice versa; le complément vrai (à 2) s'obtient de la même manière mais en ajoutant ensuite 1. Le tableau ci-après représente les notations décimales binaires avec bit de signe, binaire en complément à 1 et binaire en complément à 2.

Décimal	Binaire + Bit de signe	Binaire Complément à 1	Binaire Complément à 2
7	0111	0111	0111
6	0110	0110	0110
...			
2	0010	0010	0010
1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	0000
-1	1001	1110	1111
-2	1010	1101	1110
...			
-6	1110	1001	1010
-7	1111	1000	1001
-8	/	/	1000

Il est à noter que le 0 (zéro) n'a qu'une seule représentation en complément à 2 alors que le +0 et le -0 sont différents ailleurs.

Le principal avantage de ce type de notation (complément vrai) réside dans la soustraction où il suffit de réaliser une simple addition sans tenir compte de la retenue. Pour bien comprendre prenons un exemple en décimal sur 2 digits donc la valeur maximale sera 100_{10} soit à calculer:

$$\begin{array}{rr} & 63 \\ - & 28 \\ \hline & 35_{10} \end{array}$$

remplaçons 28 par son complément vrai (à 100) qui est égal à 72 et faisons l'addition sans tenir compte de la retenue

$$\begin{array}{rr} & 63 \\ + & 72 \\ \hline & 135 \end{array}$$

ce qui donne bien 35_{10}

On imagine sans peine l'intérêt évident de remplacer une soustraction par une addition. Pratiquement ce sera la notation en complément vrai qui sera la plus fréquemment utilisée.

3.3. Représentation d'un nombre réel

3.3.1. Le format virgule fixe

C'est la façon naturelle d'écrire un nombre binaire dans une cellule mémoire : on le considère comme un entier et le placement de la virgule est laissé aux bons soins du programmeur. Le stockage de la virgule n'est pas apparent en mémoire, son emplacement est précisé dans le programme, l'ordinateur ne travaille alors que sur des entiers. Les résultats obtenus peuvent être imprécis car on peut être amené à tronquer une partie du nombre (1/3 = 0.333333…).

3.3.2. Le format virgule flottante

Si le format en virgule fixe est simple de fonctionnement, il entraîne néanmoins un suivi très méticuleux de la part du programmeur. Aussi pour garder le maximum de précision et de chiffres significatifs, il a été développé la notation dite à *virgule flottante*. Cette notation représente tout nombre sous la forme :

$$N = S\,M \times \alpha^{E}$$

où S est le signe du nombre (+ ou -)
M est la *mantisse* du nombre
E est l'exposant du nombre
α est, en général, choisi égal à la base du système de numération ; en virgule flottante binaire $\alpha=2$

Cette notation est directement inspirée de la notation scientifique (ou ingénieur) utilisant les puissances de 10 ; en effet, on écrit souvent :

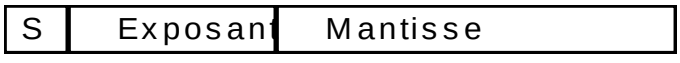
125 000 000 = 125×10^{6}
ou = $1{,}25 \times 10^{8}$
ou = $12{,}5 \times 10^{7}$, etc...

De toutes ces notations on retient la notation dite normalisée qui permet de garder le maximum de chiffres significatifs. Pour normaliser, on décale la mantisse vers la gauche jusqu'à ce que le premier digit soit significatif et on diminue l'exposant d'autant évidemment.

$$0{,}012 \times 10^{-2} = 1{,}2 \times 10^{-4}$$

Remarque: le nombre de digits de la mantisse influence la précision des calculs (nombre de chiffres significatifs) alors que les digits de l'exposant déterminent les nombres extrêmes que la machine peut représenter.

En général, le principe de notation est le suivant:



Le **signe +** est codé par le bit à 0, le signe **-** par le bit 1.

L'exposant exprime une puissance de deux signée. Il est souvent biaisé: ainsi s'il occupe e bits il est représenté par excès de 2^{e-1}-1. Par exemple, si un exposant est codifié sur 8 bits, un exposant égal à 2 sera représenté par la valeur 127+2=129

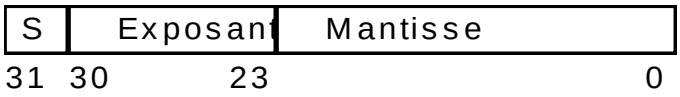
La mantisse : elle est considérée avec la virgule implicitement à gauche. On raménera toujours le nombre sous la forme 1,x. (0,0101 = 1,01 $*2^{-2}$). Il devient alors inutile de représenter le 1 figurant devant la virgule; la mantisse correspond aux bits après la virgule; on dit qu'elle a un bit caché.

Le standard IEEE définit 3 formats de représentation en virgule flottante:

Réel simple précision
Réel double précision
Réel en précision étendue

3.3.2.1. Réel simple précision

Appelé souvent "float" dans les langages de programmation, il est codé sur 32 bits



La mantisse est codée sur 23 bits et l'exposant sur 8 bits. Enfin le 32ème bit est le bit de signe de la mantisse.

La mantisse est à bit caché, ce qui signifie que les bits M_0 à M_{22} représentent les puisssances de 2^{-1} à 2^{-23}.
L'exposant est biaisé sur 8 bits. Exposant biaisé = exposant réel + biais

Exposant réel = exposant à représenter
Biais = 2^{n-1} Ici = 2^{8-1}-1 = 2^7-1= 127

Exemple 1: $-0{,}08203125_{10} = -0{,}00010101_2 = -1{,}0101 \times 2^{-4}$

Signe = 1
Exposant= -4 → Exposant réel = -4+127 = 123 = 01111011
Mantisse=01010000000000000000000

Résultat = 1 01111011 01010000000000000000000$_2$ = BDA80000$_{16}$

Exemple 2: $15{,}5_{10} = 1111{,}1_2 = 1{,}1111 \times 2^3$

Signe = 0
Exposant= 3 → Exposant réel = 3+127 = 130 = 10000010
Mantisse=11110000000000000000000

Résultat = 0 10000010 11110000000000000000000$_2$ = 41780000$_{16}$

Exemple 3: $-4,25_{10} = -100,01_2 = -1,0001 \times 2^2$

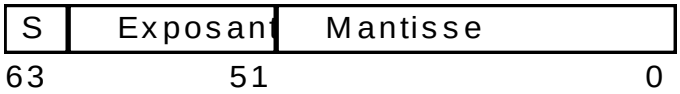
Signe = 1
Exposant= 2 → Exposant réel = 2+127 = 129 = 10000001
Mantisse=00010000000000000000000

Résultat = 1 10000001 00010000000000000000000 = $C0880000_{16}$

3.3.2.2. Réel double précision (15 chiffres significatifs)

Celui-ci est codé sur 64 bits. La valeur absolue de son étendue est

$$1,7.10^{-308} <= x <= 1,7.10^{+308}$$



Son nom est "double".

3.3.2.3. Réel en précision étendue (format 80x87)

Ce format est directement lié au format du coprocesseur arithmétique d'INTEL. Son nom en langage est "long double."

Il est codé sur 80 bits. Son étendue est la suivante :

$$3,4.10^{-4932} <= x <= 1,1.10^{+4932}$$

4. Codage des informations non numériques

Nous avons vu jusqu'à présent le codage des nombres au sein de la machine. Or, il existe beaucoup d'autres informations gérées par la machine qui ne sont pas des nombres : ce sont les **chaînes de caractères,** un caractère pouvant être un chiffre, une lettre, un signe de ponctuation ou un symbole connu et défini et les **instructions machine,** lesquelles permettent de commander ses actions.

4.1. Le codage des caractères

Dans ce cas il s'agit de coder des caractères numériques, entre autres, mais aussi des *caractères non numériques* tels que des lettres, des signes, des fonctions (tabulations, retour à la ligne …)

De nombreux codes ont été développés pour la représentation des caractères en machine (sur 6, 7, 8 bits), pour le dialogue avec les périphériques ou pour les transmissions. Parmi tous ces codes le principal, pratiquement universellement employé est le **code A.S.C.I.I.** (prononcer ASKI) qui est l'acronyme de "American Standard Code for Information Interchange",

standard américain pour l'échange des informations. Ce code associe à tout caractère un groupe de 7 bits ce qui donne 2^7 combinaisons possibles soit 128 au total, dont en fait seules 96 correspondent à des symboles graphiques ; les 32 autres à des fonctions bien particulières (émission d'un bip sonore, retour en début de ligne, fin de texte, gestion des transmissions ...). Le tableau ci-après donne les valeurs binaires associées aux différents codes. Les deux premières colonnes (O et 1) contiennent les **codes de contrôle** ou **caractères non imprimables** qui n'ont pas de symbole associé sur le clavier d'une console. Les colonnes 4, 5 et 6, 7 donnent respectivement les majuscules et les minuscules. On notera que le passage de minuscule à majuscule se fait en remettant à zéro le bit 6. On a ainsi les codes respectifs 104 pour "D" et 144 pour "d", codes en octal.

Code ASCII 7 bits

				b7	0	0	0	0	1	1	1	1
				b6	0	0	1	1	0	0	1	1
				b5	0	1	0	1	0	1	0	1
b4	b3	b2	b1		0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	§	1	A	Q	a	q
0	0	1	0	2	STX	DC2	«	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	‘	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Le code binaire à 7 bits d'un symbole quelconque de la table est obtenu par lecture directe

Exemple: Lettre g ---- > Code 1100111 soit 67 en Hexadécimal, ou 147 en octal, ou 103 en décimal.

Il faut noter toutefois que, suivant les pays, certains caractères peu utilisés, sont adaptés aux besoins de la langue : ainsi, en France, les consoles dites AZERTY, sont à même de générer les caractères accentués, hélas non prévus par la norme américaine (étant donné qu'il n'y a pas d'accents en anglais). Pour ce faire, il existe deux techniques différentes:

Code ASCII national

Il y a simplement substitution du symbole original par le nouveau symbole dans le code à 7 bits. On dénombre alors plusieurs jeux de caractères pour le même code et ce pour les pays suivants : U.K. (United Kingdom ou Royaume Uni), U.S. (U.S.A.), France, Italie, Danemark, etc. Le tableau suivant indique, à titre d'exemple, les symboles spécifiques obtenus suivant les claviers nationaux, et ce, pour les matériels HEWLETT PACKARD.

Codes par langue

	KEYBOARD	DECIMAL VALUE											
LANGUE	OPTION #	35	39	64	91	92	93	94	96	123	124	125	126
USASCII	(standard)	#	‘	@	[	\	]	^	‘	{	|	}	~
Swedish	101	#	‘	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Norvegian	102	#	‘	@	Æ	Ø	Å	^	`	æ	ø	å	¨
French	103	£	‘	à	°	ç	§	^	`	é	ù	è	¨
German	104	£	‘	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
United Kingdom	105	£	‘	@	[	\	]	^	`	{	|	}	~
European Spanish	106	#	‘	@	¡	Ñ	¿	)	`	{	ñ	}	¨
French Canadian	107	#	‘	@	[	ç	]	^	`	é	Ç	É	¨
English Canadian	108	#	‘	@	[	ç	]	^	`	é	Ç	É	¨
Italian	109	£	‘	§	°	ç	é	^	ù	à	ò	è	ì
Dutch	110	#	‘	@	ç	\	§	^	`	*ƒ*	|	¨	~
Finnish	111	#	‘	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
Danish	112	§	‘	@	Æ	Ø	Å	^	`	æ	ø	å	¨
German Swiss	113	£	‘	à	°	ç	§	^	`	ä	ö	ü	¨
French Swiss	114	£	‘	à	°	ç	§	^	`	ä	ö	ü	¨
Latin American Spanish		#	‘	@	¡	Ñ	¿	^	`	‘	ñ	ç	¨
Belgian	116	£	‘	à	°	ç	§	^	`	é	ù	è	¨

La table ASCII 8 bits de l'IBM PC français

La table ASCII 8 bits française est couramment utilisée dans tous les compatibles IBM en environnement DOS© (sous Windows©, c'est la table ANSI qui est utilisée et celle-ci est légèrement différente par rapport à la table ASCII).

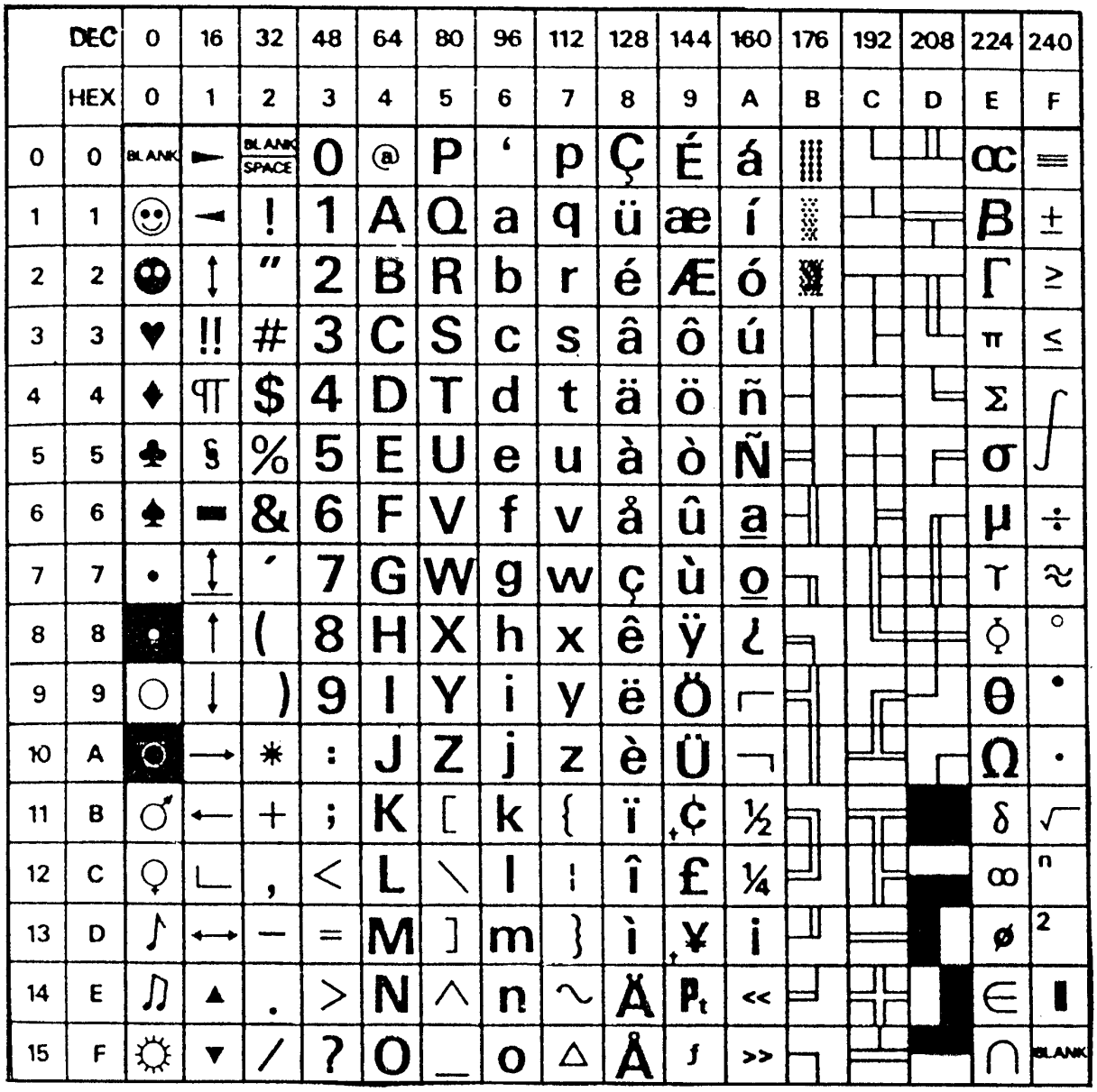


Table ASCII 8 bits française (représentation graphique)

Le code EBCDIC

Le code EBCDIC (Extended Binary Coded Decimal Interchange Code) est un code à 8 bits (ou 8 moments), très utilisé surtout sur les machines IBM. Sur les 256 possibilités de codage, toutes ne sont pas utilisées mais on y retrouve aussi des codes de contrôle comme pour le code ASCII.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NUL	DLE	DS		blanc	1	-									0
1	SOH	DC1	SOS				/		a	j			A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	DC3							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	EOB	UC					f	o	w		F	O	W	6
7	DEL	IDL	PRE	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z		I	R	Z	9
10	SMM	CC	SM		⊂	!		/								
11	VT				.	$	.	#								
12	FF	IFS		DC4	<	*	%	@								
13	CR	IGS	ENQ	NAK	(	)	-	`								
14	SO	IRS	ACK		+	;	<	=								
15	SI	IUS	BEL	SUB	|	¬	?	~								

Code EBCDIC

4.2. Le codage des instructions

Les instructions, stockées en mémoire, comportent un certain nombre d'informations, notamment le *code opération* indiquant l'opération à effectuer, *des opérandes* ou des adresses ou numéro de registres ou de cellules mémoire, des *conditions d'adressage* spécifiant les calculs éventuels à faire subir aux adresses de cellules mémoire.

Suivant le nombre et l'importance de ces informations, les instructions seront codées sur un ou plusieurs mots machine. A chacun des éléments d'information composant l'instruction on associe généralement une zone formée d'un nombre de bits suffisant pour coder les différents états possibles de cette information. C'est ainsi qu'une zone de 6 bits allouée au code instruction permettra de coder $2^6 = 64$ opérations différentes, qu'une zone de 4 bits affectée à un numéro de registre permettra de désigner un registre parmi 16 et qu'une zone de 16 bits permet d'adresser une mémoire de 64 k-mots (soit 65 536 adresses différentes).

L'ensemble de toutes les instructions exécutables par la machine constitue le **jeu d'instructions.**

Dans le problème crucial du codage des instructions il existe pratiquement toujours un logiciel spécifique dit *assembleur*, qui permet la génération des instructions binaires à partir de langages plus ou moins évolués ; dans le cas de langages évolués tels C, PASCAL, FORTRAN un outil de traduction, le *compilateur*, traduit les instructions « évoluées » en langage d'assemblage, ou instructions de base, lesquelles seront converties en binaire par l'assembleur. Sur les systèmes très bas de gamme (systèmes à microprocesseurs par exemple) l'utilisateur doit parfois réaliser cet assemblage manuellement par transcription des opérations

à partir d'un tableau constructeur donnant les configurations binaires associées aux instructions.

5. Contrôles de l'information

Le codage binaire est très pratique pour une utilisation dans des appareils électroniques tels qu'un ordinateur, dans lesquels l'information peut être codée grâce à la présence ou non d'un signal électrique. Toutefois, le signal électrique peut subir des perturbations, notamment lors du transport des données (dans un réseau par exemple) car les données circulent sur un long trajet. C'est pourquoi il existe des mécanismes permettant de garantir un certain niveau d'intégrité des données; il s'agit de codes particuliers permettant de détecter les erreurs - **codes détecteurs** ou **auto-vérificateurs -** voire de les corriger - **codes auto-correcteurs -.** Pour ce faire, la seule solution consiste à ajouter un certain nombre de bits supplémentaires (non utilisés pour coder l'information) d'où le nom de *code redondant*.

Voici deux exemples classiques de ces codes:

5.1. Contrôle 2 sur 3

Grossièrement, au lieu de transmettre une information A codée sur n bits, on la transmet trois fois. A la réception de ces trois informations trois cas peuvent se présenter:

1) les trois informations reçues sont identiques : on en déduit qu'il n'y a pas d'erreur (à moins qu'il n'y ait eu trois erreurs identiques, cas très improbable !) ;

2) on obtient deux informations identiques, la troisième étant différente: on en déduit une **erreur** sur la troisième, et une bonne transmission sur les deux autres (hormis le cas très improbable inverse!). Le code est alors auto-correcteur;

3) on obtient trois informations différentes. On peut en déduire qu'il y a eu au moins deux erreurs et l'on ne peut statuer sur le bon résultat : le code n'est qu'auto-vérificateur.

5.2. Contrôle de parité

Le code de parité est très utilisé dans les milieux "sûrs" où la probabilité d'erreur est faible. Son principe consiste à associer aux bits d'informations un (n + 1) ième bit, dit *bit de parité*, positionné de telle sorte que le nombre total de bits égaux à 1 soit pair (code à parité paire) ou impair (code à parité impaire). Ce code n'est que partiellement auto-vérificateur car il ne permet pas de détecter une double erreur (deux bits erronés dans la même information).

Exemple : soit l'information sur 8 bits suivante: 11010011 à transmettre avec une parité paire. Il suffira dans ce cas d'ajouter un 9ième bit à 1 ce qui donne bien au total 6 bits à 1.

5.3. Enoncés

1. Convertir les nombres décimaux suivants en binaire, en octal et en hexadécimal :

	0	8	10	12	16	18
Binaire						
Octal						
Hexadécimal						

2. Convertir les nombres non signés suivants en hexadécimal

	10_8	100_8	256_8
Hexadécimal			

3. Convertir les nombres non signés suivants en octal

	10_{16}	256_{16}	$CAFE_{16}$	2764_{16}	$FFFE_{16}$
Octal					

4. Convertir les nombres suivants de décimal en binaire

Décimal	5	12	22	126
Binaire				

5. Convertir les nombres suivants de binaire en décimal

Binaire	11	101	1000	10101
Décimal				

6. Convertir les nombres suivants de décimal en octal

Décimal	2	132	1024	10101	35000
Octal					

7. Convertir les nombres suivants d'octal en décimal

Octal	7	10	75	333	25033
Décimal					

8. Convertir les nombres suivants de décimal en hexadécimal

Décimal	8	10	43	47000
Hexadécimal				

9. Convertir les nombres suivants d'hexadécimal en décimal

hexadécimal	B	10	5B	ABB	101010
décimal					

10. Convertir les nombres suivants de binaire en octal

binaire	11000	1010110	11101110
octal			

11. Convertir les nombres suivants d'octal en binaire

octal	75	562	1234
binaire			

12. Convertir les nombres suivants de binaire en hexadécimal

binaire	11000	1010110	111101010
hexadécimal			

13. Convertir les nombres suivants d'hexadécimal en binaire

hexadécimal	B12	CED	F1A9
binaire			

14. Soit le contenu (en hexadécimal) d'une suite de cellules mémoires ; sachant que le code utilisé est le code ASCII 8 bits français, indiquer le message en clair associé à cette suite d'octets :
4C,27,49,6E,66,6F,72,6D,61,74,69,71,75,65,20,63,65,20,6E,27,65,73,74,20,70,61,73,20,64,69,66,66,69,63,69,6C,65,20,21,07 .

15. Pour les nombres énumérés ci-dessous, représentez le contenu binaire des octets qu'ils occupent:
-dans le cas où ils sont stockés comme chaîne de caractères
-dans le cas où ils sont stockés en binaire
Le code ASCII est utilisé.

31 , 945