1. Introduction

La représentation interne consiste à donner un codage binaire à l'information

Information externe → codage → information interne (binaire)

2. Typologie de l'information

Les ordinateurs traitent deux types d'informations :

1. Les instructions : représentent les opérations effectuées par l'ordinateur
2. Les données : ce sont les opérandes sur lesquels portent les opérations

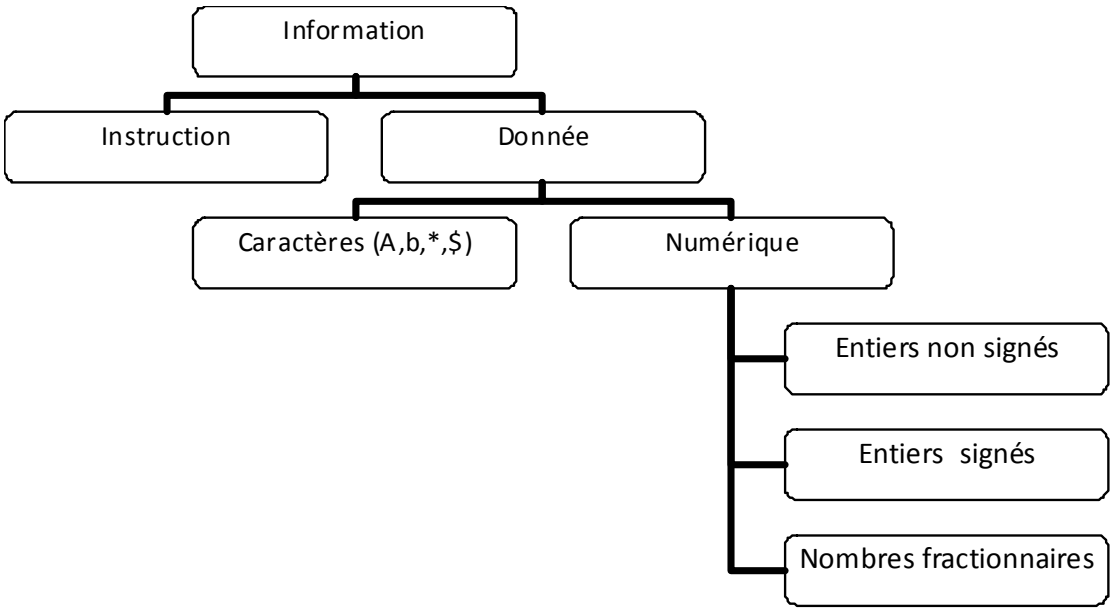


Figure1 : *Typologie de l'information*

3. Le codage binaire :

En binaire, on distingue trois principaux systèmes de codage :

3.1 Le codage binaire pur (naturel)

Ce codage est celui dans lequel les nombres sont représentés en binaire (b=2). Ce code présente l'inconvénient de changer plus qu'un seul bit lorsqu'on passe d'un nombre à son suivant (voir tableau ci-dessous).

3.2 Code GRAY (code binaire réfléchi)

Son intérêt réside dans des applications d'incrémentation où un seul bit change d'état à chaque incrémentation.

- **Conversion binaire → code Gray**

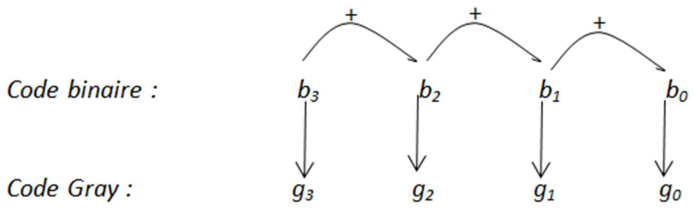
  La conversion du binaire naturel au binaire réfléchi (Gray code) est comme suit :

  - Le bit du poids le plus fort, reste le même qu'en binaire (inchangé)

- En partant de la gauche vers la droite, chaque bit est additionné à son voisin de droite. La somme est reportée à la ligne inférieure qui correspond au code Gray. Les retenues sont négligées.

- Le code Gray comporte toujours le même nombre de bits que la représentation binaire naturel.

Exemple : soit un nombre codé en binaire naturel sur 4 bits $(b_3b_2b_1b_0)_2$, le code Gray $(g_3g_2g_1g_0)_2$ est obtenu comme suit :



Exemple : convertir les valeurs 35 et 36 en binaire pur et en Gray code

Binaire 35 = $(1\,0\,0\;\;0\,1\,1)_2$ 36 = $(1\,0\,0\;\;1\,0\,0)_2$

Gray code 35 = $(1\,1\,0\;\;0\,1\,0)_{gc}$ 36 = $(1\,1\,0\;\;1\,1\,0)_{gc}$

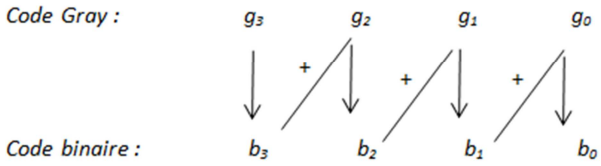
Exemple : écrire les nombres de 0 à 15 en code binaire naturel et en code Gray

décimal	binaire	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Le code gray est dit réfléchi, car les **n-1** bits sont générés par réflexion (un miroir).

- **Conversion code Gray → binaire**

- Le bit du poids le plus fort, reste le même qu'en gray code (inchangé)
- En partant de la gauche vers la droite, chaque bit du code binaire est additionné à son voisin diagonal (Gray). La somme est reportée à la ligne inférieure qui correspond au code binaire. Les retenues sont négligées.



3.3 Le code DCB (Décimal codé binaire)

Dans le code **DCB**, en anglais **BCD** (binary coded decimal), chaque chiffre décimal est remplacé par son équivalent binaire sur **4** bits

Décimal	0	1	2	3	4	5	6	7	8	9
Binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Exemple : Convertir le nombre décimal 275 en BCD :

$2 \Rightarrow (0010)_2$

$7 \Rightarrow (0111)_2$

$5 \Rightarrow (0101)_2$

275= **0010 0111 0101** en BCD

L'avantage principal du code DCB réside dans sa facilité de conversion au système décimal. Ce code est couramment utilisé dans les instruments numériques (calculatrices électroniques, horloges numériques …)

4. Représentation des entiers

4.1 Entiers non signés (entiers naturels)

Le nombre est représenté en binaire naturel sur **n** bits.

L'intervalle des nombres représentables sur **n** bits : **[0, 2^n-1]**

Exemple : représenter la valeur 25 sur 8bits, réponse $(00011001)_2$

4.2 Entiers signés

Il existe 3 méthodes pour représenter les entiers signés:

- Signe et valeur absolue (SVA)
- Complément à 1 (C1)
- Complément à 2 (C2)

4.2.1 Signe et valeur absolue

Le bit du poids **fort** indique le signe (1 : signe + et 0 : signe -), les **n – 1** bits restants indiquent la valeur absolue du nombre. L'intervalle des nombres représentables sur n bits en SVA est : **$[-(2^{n-1}-1), +(2^{n-1}-1)]$**

Exemple **:** représenter la valeur -15 et +20 sur 8bits en SVA

- ***Conversion SVA →décimal***

Convertir les **n – 1** bits en décimal et ajouter (+) si le bit du poids fort=0 ou (-) sinon

Exemple : donnez la valeur décimale des nombres suivants représentés en SVA sur 8 bits ?

$(00001001)_2 = +9$ $(10000101)_2 = -5$

- ***Avantages et inconvénients de SVA***

SVA est assez simple, mais présente l'inconvénient de la double représentation du zéro -0 et +0 (voir le tableau ci-dessous), et aussi la difficulté des opérations arithmétiques qui sont compliquées, à cause du bit de signe qui doit être traité à part. Cette méthode n'est pas utilisée par les constructeurs, car il faut deux circuits : un pour l'addition et l'autre pour la soustraction. Mais l'idéal est d'utiliser un seul circuit qui fait les deux opérations

Exemple : calculer 5 + (-2) sur 4 bits en SVA ? résultat faux = -7

```
  0101
+ 1010
------
  1111
```

4.2.2 Complément à 1 (C1)

- Les nombres positifs sont obtenus par conversion vers le binaire naturel

  Exemple : représenter +12 en C1 sur 8 bits, réponse : +12= $(00001100)_2$
- Les nombres négatifs sont complémentés c.-à-d à partir de la représentation binaire de l'opposé positif, les 0 sont inversés en 1, et les 1 en 0.

Exemple **:** représenter -23 en C1 sur 8 bits :

$-23 = (11101000)_2$

- L'intervalle des nombres représentables sur **n** bits en C1 est : **$[-(2^{n-1}-1), +(2^{n-1}-1)]$**

- ***Conversion C1 →décimal***

- Les nombres positifs (bit du poids fort = 0) sont obtenus par conversion binaire →décimal.
- Les nombres négatifs (bit du poids fort = 1) sont obtenus en inversant les **n-1** bits puis introduire le signe -

Exemple : donnez la valeur décimale des nombres suivants représentés en C1 sur 8 bits ?

$(00001111)_2 = +15$ $(11110011)_2 = -12$

- ***L'addition en complément à 1***

  - Si aucune retenue n'est générée par le bit de signe, le résultat est correct, il est représenté en C 1
  - sinon, la retenue sera additionnée au résultat de l'opération, celui-ci est représenté en C1

Exemple : calculer +5 + 2 -5-2 +5 + (-2) -5 +2 sur 4 bits en C1 ?

+5+ (+2)	-5+(-2)	+5+(-2)	-5+(+2)
0101 + 0010 --- 0111	^{1}1010 + 1101 --- 0111 + 1 --- 1000	^{1}0101 + 1101 --- 0010 + 1 --- 0011	1010 + 0010 --- 1100

Remarque :

L'inconvénient de la méthode C1 est la **double** représentation du zéro +0 et - 0 (voir le tableau ci-dessous)

4.2.3 Complément à 2

- En C2 les nombres positifs sont obtenus par conversion vers le binaire naturel

  Exemple : représenter +12 en C2 sur 8 bits

  $$+12 = (00001100)_2$$

- Les nombres négatifs sont obtenus en calculant d'abord le complément à 1, puis ajouter 1.

  $$C_2(N) = C_1(N) + 1$$

- L'intervalle des nombres représentables sur **n** bits en C2 est : **$[-2^{n-1}, +(2^{n-1}-1)]$**

Remarque :

Une astuce rapide pour calculer le C2 d'un nombre négatif, consiste à conserver tous les bits (de l'opposé positif) à partir de la droite jusqu'au premier **1** compris et d'inverser le reste des bits.

Exemple **:** représenter les valeurs suivantes en C2 sur 8bits : -12 -22

$+12 = (00001100)$ $+31 = (00011111)_2$

$-12 = (11110100)_2$ $-22 = (11100001)_2$

- ***Passage C2 → décimal***

- Les nombres positifs (bit du poids fort = 0) sont obtenus par conversion binaire → décimal.
- Les nombres négatifs (bit du poids fort = 1) sont obtenus par conversion binaire → décimal (on affecte un signe - au bit du poids fort)

Exemple : donnez la valeur décimale des nombres suivants représentés en C2 sur 8 bits ?

$(00001110)_2 = +14$ $(10001000)_2 = -120$ $(11101110)_2 = -18$

- ***L'addition en complément à 2***

o S'il y a une retenue qui est générée par le bit de signe, elle est ignorée et le résultat est en C2

Exemple : calculer 5 + 2 -5-2 5 -2 -5 +2 sur 4 bits en C2 ?

+5+(+2)	-5+(-2)	+5+(-2)	-5+(+2)
0101	1011	0101	1011
+ 0010	+ 1110	+ 1110	+ 0010
----------	----------	----------	----------
0111	1001	0011	1101

- ***Avantages de complément à 2***

o la représentation la plus utilisée pour les nombres négatifs

o Une seule représentation du zéro

o Un nombre négatif supplémentaire **- 2^{n-1}**

Remarque :

En C1 ou C2, les opérations arithmétiques sont avantageuses. En effet, la soustraction d'un nombre se réduit à l'addition de son complément. Ce qui permet à la machine d'utiliser un seul circuit (additionneur-soustracteur)

Exercice : Quelles sont les valeurs représentables sur 4 bits en SVA, C1 et C2 ?

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

C1	décimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	- 0
1110	-1
1101	-2
1100	-3
1011	-4
1010	-5
1001	-6
1000	-7

C2	décimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

- **Le Débordement (Overflow)**

Dans un ordinateur la taille des registres est fixe, ce qui impose une limite aux nombres représentés. Il y a un débordement ou dépassement de capacité si :

- la somme de deux nombres négatifs donne un nombre positif ou la somme de deux nombres positifs donne un nombre négatif
- le résultat de l'opération dépasse l'intervalle des valeurs représentables
- Pour éviter le débordement, il faut choisir un nombre de bits plus grand

Exemple : calculer en C2 sur 4 bits les opérations suivantes 5+7 -5-7 ?

On remarque que les résultats sont incorrects

5+7	-5-7
0101	1011
+ 0111	+ 1000
----------	----------
1100 =- 4	0011 =+4

5. Représentation des Nombres fractionnaires

Un nombre réel est constitué de deux parties : la partie entière et la partie fractionnelle. Le problème qui se pose est comment indiquer à la machine la position de la virgule ? Il existe deux méthodes pour représenter les nombre réels :

Virgule fixe : la position de la virgule est fixe, **Virgule flottante** : la position de la virgule change (dynamique)

5.1 Représentation en virgule Fixe

Cette représentation pose un problème au niveau de la machine. La première solution était de ne pas représenter matériellement la virgule et de traiter le nombre fractionnaire comme un nombre entier. La virgule est virtuelle, elle est gérée par le programmeur, c'est lui qui définit sa position, chose qui n'est pas facile, d'où son inconvénient. Un autre inconvénient est la limitation des valeurs et il n'y a pas une grande précision.

Dans cette représentation le bit du poids fort représente le signe (+/-), la partie entière est représentée sur **n** bits et la partie fractionnelle sur **p** bits.

Exemple : On considère un nombre représenté sur 6 bits

1 bit : pour le signe

3 bits pour la partie entière

2 bits pour la partie fractionnaire Valeur Min= $(1\ 111\ 11)_2$ = -7.75 Valeur MAX $(0\ 111\ 11)_2$ = +7.75

[-7.75, -7.50, -7.25,……+0.00……,+7.25, +7.50, +7.75]

5.2 Représentation en virgule flottante

Au début, la représentation en virgule flottante n'était par normalisée où chaque ordinateur utilisait son propre format. Plusieurs normes ont été définies, parmi elles la norme *IEEE 754 (Institute of Electrical and Electronics Engineers)*

- **Représentation en IEEE 754 simple précision**

Le nombre est exprimé comme suit :

- 1 bit de signe
- 8 bits pour l'exposant décalé = exposant réel +127. L'exposant est décalé de 2^{8-1} - 1. Ce décalage est utile car l'exposant peut être positif ou négatif
- 23 bits pour la mantisse

Pour écrire un nombre sous la forme IEEE754 il faut :

1. Convertir le nombre en binaire
2. Ecrire le nombre sous la forme normalisée **$N= (+/-)(1,m)_2.2^{ER}$**
3. Calculer l'exposant décalé= Exposant réel+ 127

Remarque:

Le 1 précédant la virgule n'est pas codé en machine (appelé bit caché)

Exemple : représenter en IEEE 754 le nombre N= -3.625

1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Pour convertir en décimal un nombre écrit sous la forme IEEE754 :

1. calculer l'exposant réel =Eb-127
2. Calculer *valeur* = *signe* ×(1, *mantisse)*$_2$ × 2^{Er}, avec *signe* = ±1

Exemple : Quelle est la valeur en décimal du nombre suivant représenté en IEEE 754

1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Réponse : - **4.75**

Dans la norme IEEE754 il y a des valeurs spéciales

Le zéro : MANTISSE =0 EXPOSANT = 0

1/0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Infini : MANTISSE = 0 EXPOSANT = 1 PARTOUT

Une valeur infinie $\pm\infty$ est utilisée pour indiquer par exemple un débordement ou une division par zéro.

1/0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NaN (Not a Number) MANTISSE ≠ 0 EXPOSANT = 1 PARTOUT

NAN est le résultat d'une opération arithmétique erronée. Par exemple le calcul de la racine carrée des nombres négatifs, 0/0, inf/inf, 0*inf

Remarque :

La norme IEEE754 définit trois formats de représentations des nombres flottants :

. Simple précision sur 32 bits (1bit 8bits 23 bits Biais= 127)

. Double précision sur 64 bits (1 bit 11bits 52 bits Biais= 1023)

. Précision étendue sur 80 bits (1 bit 15 bits 64 bits Biais= 16383)

Exemple : représenter la valeur -12.75 en IEEE754 Double Précision

Réponse : 1 10000000010 1001100000000000000000000000000000000000000000000000000000000000

6. Représentation des Caractères

L'ensemble des caractères alphanumériques regroupe les lettres 'a'……'z' ,'A…….'Z', les chiffres 0…..9,et les caractères spéciaux +,-,*, ? # . Le codage des caractères se fait par une table de correspondance propre à chaque code utilise. Parmi les standards les plus utilisés, on peut citer les codes suivant :

- **Le code EBCDIC** (Extended Binary Coded Decimal Interchange Code) : Codage sur 8 bits, utilisé principalement par IBM
- **Codage ASCII standard** (American Standard Code for Information Interchange): chaque caractère sur 7 bits pour représenter 128 caractères
- **Codage ASCII étendu**: chaque caractère sur 8 bits pour représenter 256 caractères
- **Unicode :** c'est le codage de la plupart des alphabets : Arabe, Chinois, hébreu, cyrillique …….. Il regroupe plusieurs standards : UTF-8, UTF-16, UTF-32…...

  UTF-8 (UCS : Universal Character Set Transformation Format) est le codage le plus utilisé pour représenter le texte Unicode dans les pages Web et les bases de données. Dans ce codage, un caractère est représenté sur 1, 2 3 ou 4 octets. Dans cette norme, il y a de moins en moins de problèmes de compatibilité.

Exemple : voir la table ASCII étendu

Le code $(41)_{16}$= $(01000001)_2$ correspond au caractère A

Le code $(61)_{16}$ =(01100001) correspond au caractère a

Le code $(3F)_{16}$ =$(00111111)_2$) correspond au caractère ?