Chapitre 1 Systèmes de numération et codage

Plan du chapitre:

1. Systèmes de numération
   - 1.1 Introduction
   - 1.2 Conversion de bases
     a. Conversions base quelconque (X) → décimal (base10)
     b. conversion décimal→ base quelconque (X)
     c. Conversions base quelconque (X) → une autre base quelconque (Y)
     d. Conversions base quelconque (X =2^n) → une autre base quelconque (Y=2^m)
2. Représentation des nombres signés en binaire
3. Opération arithmétiques
   - 3.1 en binaire
   - 3.2 en Complément à 2
4. Représentation des réels dans la machine
5. Codage
   - 5.1 Introduction
   - 5.2 Les codes décimaux.
   - 5.3 Le code Gray
   - 5.4 Codes correcteurs d'erreurs
   - 5.5 *Codage des caractères : code ASCII*

1. Systèmes de numération

1.1 Introduction

La base habituelle que nous utilisons est la base 10 (système décimal). Les machines numériques utilisent d'autres bases. Une base b, utilise b symboles différents :

- **Base 10 (base décimal):** C'est la base la plus couramment utilisée, elle est constituée de 10 symboles : $\{0,1,2,3,4,5,6,7,8,9\}$.
- **Base 2 (base binaire):** C'est la base qui est utilisée par l'ordinateur, elle est constituée de 2 symboles $\{0,1\}$.
- **Base 8 (base Octale):** Permet de représenter 3 bits par un seul symbole, cette représentation n'est plus utilisée en informatique. Elle est constituée de 8 symboles $\{0,1,2,3,4,5,6,7\}$.
- **Base 16 (base Hexadécimal):** C'est la plus utilisée actuellement, elle permet de représenter 4 bits par un seul symbole. Elle est constituée de 16 symboles :
  $\{0,1,2,3,4,5,6,7,8,9, A\ (pour 10), B\ (pour\ 11), C\ (pour\ 12), D\ (pour\ 13), E\ (pour\ 14), F\ (pour\ 15)\}$

1.2 Changement de base

Consiste à ré-exprimer un nombre défini dans une base donnée, dans une autre base de numération.

a. Base quelconque X vers la base décimale:

On utilise le développement polynomial : $\sum_{i=-m}^{n-1} a_i b^i$

Soit une *base b* associée à *b* symboles $\{S_0, S_1, S_2, \ldots, S_{b-1}\}$

Soit : N= $(a_{n-1}a_{n-2} \ldots \ a_1a_0 \ \ a_{-1}a_{-2}..a_{-m+1}a_{-m})_b$ un nombre positif dans la base b

Avec :

a_i : Est le chiffre de rang i appartient un ensemble des b symboles

a_{n-1} : Est le chiffre le plus significatif

a_{-m} : Est le chiffre le moins significatif

$a_{n-1}a_{n-2} \ldots \ a_1a_0$: partie entière

$a_{-1}a_{-2}..a_{-m+1}a_{-m}$: partie fractionnaire

Alors N en décimal est égal :

$$N = (a_{n-1}b^{n-1} + a_{n-2}b^{n-2} \ldots + a_1b^1 + a_0b^0 + a_{-1}b^{-1} + \ a_{-2}b^{-2}..+a_{-m}b^{-m})_{10}$$

Exemples: $(11001{,}11)_2 = (?)_{10}, (31{,}4)_8 = (?)_{10}, (19, C)_{16} = (?)_{10}.$

- $(11001{,}11)_2 = \ 2_0.1 + \ 2_1.0 + 2_2.0 + 2_3.1 + 2_4.1 + 1.2_{-1} + 1.2_{-2}$
  $= 2 + 8 + 16 + 1/2 + 1/4 = \ (25{,}75)_{10}$
- $(31{,}4)_8 = \ 8^0 * 1 + 8^1 * 3 + 8^{-1} * 4 = \ (25{,}5)_{10}$
- $(19, C)_{16} = \ 16^0 * 9 + 16^1 * 1 + 16^{-1} * 12 = \ (25{,}75)_{10}$

Décimal	Hexadécimal	Binaire
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

base 10

$$d_i \in \{0,1,...,9\}$$

$$N = \sum_{i=0}^{k} d_i.10^i$$

base 2

$$b_i \in \{0,1\}$$

$$N = \sum_{i=0}^{k} b_i.2^i$$

base 16

$$h_i \in \{0,...,9,A,B,C,D,E,F\}$$

$$N = \sum_{i=0}^{k} h_i.16^i$$

b. Base décimale vers une base quelconque :

✓ **Pour la partie entière** : La méthode utilisée est les divisions successives :

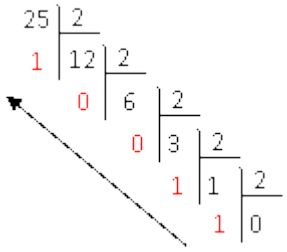
- On devise le nombre par la base "b", puis le quotient par "b" jusqu'a l'obtention d'un quotient Nul.
- La concaténation des restes de devisions constituera le résultat dans la base visée (le sens de concaténation: du bas vers le haut).

✓ **Pour la partie fractionnaire**

La partie décimale est obtenue en effectuant des multiplications successives de la partie fractionnaire par la base b jusqu'à ce que la partie décimale soit nulle ou bien la précision désirée est atteinte.

Exemple: convertir le nombre suivant 25.625 en binaire.

- partie entiere: $(25)_{10}=(?)_2$.



$(25)_{10} = (11001)_2$

- Partie fractionnaire : $(0.625)_{10} = (?)_2$

$0.625 * 2 = 1.25$ (on garde le 1)
$0.25 * 2 = 0.5$ (on garde le 0)
$0.5 * 2 = 1.0$ (on garde le 1 et on s'arrête)

Ainsi : $(0.625)_{10} = (0.101)_2$ en regroupant les deux parties on obtient :

$$(\mathbf{25.625})_{\mathbf{10}} = (\mathbf{11001.101})_{\mathbf{2}}$$

c. Conversions base quelconque (X) → une autre base quelconque (Y)

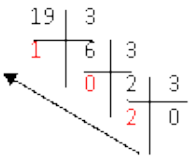
En général on ne peut pas passer directement d'une base X (différente de 10) vers une autre base Y (différente de 10), il faut passer par la base 10

Base X → base 10 → base y

Exemple : $(34)_5 = (?)_3$

$(34)_5 = 5^0 * 4 + 5^1 * 3 = 4 + 15 = (19)_{10}$

$(19)_{10} = (?)_3$



$(19)_{10} = (201)_3$ Donc : $(\mathbf{34})_{\mathbf{5}} = (\mathbf{201})_{\mathbf{3}}$

d. Conversions base quelconque (X =2^n) → une autre base quelconque (Y=2^m)

Dans ce cas la conversion direct est possible et elle est plus simple sans passer par la base 10.

- **Base 8 vers Base 2:** remplacer chaque chiffre par son équivalent binaire sur 3 bits.
  Exemple : $(31)_8 = (?)_2$.
  $(1)_8=(001)_2$
  $(3)_8=(011)_2$
  $\}\mapsto (31)_8 = (011001)_2$
- **Base 16 vers Base 2:** remplacer chaque chiffre par son équivalent binaire sur 4 bits
  Exemple: $(19,5)_{16}= (?)_2$.
  $(9)_{16}=(1001)_2$
  $(1)_{16}=(0001)_2$
  $(5)_{16}=(0101)_2$
  $\}\mapsto (19)_{16}= (00011001,0101)_2$

- **Base 2 vers Base 8:** regroupement des bits en des groupements de 3 bits, puis remplacer chaque groupe par son équivalent en octale.
  Exemple: $(11001, 1101)_2 = (?)_8$.
  $(\underline{011}\ \underline{001}, \underline{110}\ \underline{100})_2 = (31,64)_8$
- **Base 2 vers Base 16:** regroupement des bits en des groupements de 4 bits, puis remplacer chaque groupe par son équivalent hexadécimal.
  Exemple: $(11001,110111)_2 = (?)_{16}$.
  $(\underline{01}\ \underline{1001}, \underline{1101}\ \underline{1100})_2 = (19,DC)_{16}$.

2. Représentation des nombres signés en binaire

Il existe principalement 3 méthodes pour représenter le signe:

- Signe et valeur absolue.
- Le complément a 1 (restreint).
- Le complément a 2 (à vrai).

2.1 Signe et valeur absolue:

Dans cette représentation, le bit de plus fort poids est utilisé pour représenter le signe comme suit :

- Bit du signe = 1→ nombre négatif.
- Bit de signe = 0 → nombre positif.

Les bits restants représenteront en binaire la valeur absolue du nombre.

Exemple : $(0101) = +5 \ ; \ 1101 = -5$

Etendu : avec n bits on peut représenter les valeurs de *:* $[-\mathbf{2^{n-1}-1} \ \text{à} \ +\mathbf{2^{n-1}-1}]$

Pour n=8 , l'étendu est $[-127 \ \text{à} \ +127]$

Pour n=3 , l'étendu est $[-3 \ \text{à} \ +3]$

signe	Valeur absolue		valeur
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-0
1	0	1	-1
1	1	0	-2
1	1	1	-3

Inconvénients :

- Deux représentations pour le zéros, exemple sur 3 bits: (+0) qui s'écrit 000, et le (-0) qui s'écrit 100. Ce qui engendre des difficultés pour réaliser les opérations arithmétiques.

2.2 complément à 1:

- Il connu également sous les noms, Complément Logique ou Complément Restreint.
- Le bit de plus fort poids est utilisé pour représenter le signe comme suit :

1. Bit du signe = 1→ nombre négatif.
2. Bit de signe = 0 → nombre positif.

- Le complément à 1 d'un nombre N s'obtient en inversant les bits

  Exemples : si $N = 1011 \rightarrow C1(N) = 0100$

  $(101011)_{C1} = -(010100) = -20_{10}$

- Etendu : Pour une longueur de n bits l'étendu est : $-\mathbf{2^{n-1}-1}$ à $+\mathbf{2^{n-1}-1}$

Exemple pour n=3, l'étendu est -3 à $+3$

C1	valeur
0 0 0	+0
0 0 1	+1
0 1 0	+2
0 1 1	+3
1 0 0	-3
1 0 1	-2
1 1 0	-1
1 1 1	-0

- L'inconvénient : Deux représentation pour le zéros $\rightarrow$ difficultés pour réaliser les opérations arithmétiques.

2.3 complément à 2:

- Aussi connu sous le nom, Complément à vrai
- Représentation du signe la plus utilisée
- Soit $A = (a_{n-1}a_{n-2} \dots \ a_1a_0 \ a_{-1}a_{-2}..a_{-m+1}a_{-m})_{C2}$ sur n bits, alors le MSB (Most Significatif Bit) $\boldsymbol{a_{n-1}}$ indique le signe de A : si $\boldsymbol{a_{n-1}=0}$ $\rightarrow$A est positif et si $\boldsymbol{a_{n-1}=1}$ $\rightarrow$A est négatif
- $A = (a_{n-1}a_{n-2} \dots \ a_1a_0 \ )_{C2} = (-a_{n-1} * 2^{n-1} + \sum_{i=0}^{i=n-2} a_i * 2^i)_{10}$
- Le complément à 2 d'un nombre X, est le complément à 1 de X plus 1, et égal à –X : $\boxed{C2(X) = C1(X) + 1 = -X}$;
- Codage en C2
  - Si le nombre est positif**,** il est codé en C2 de la même manière que le binaire pur

    Exemple : $(+25)_{10} = (?)\ en\ C2\ sur\ 8\ bits$ (Il faut spécifier la longueur n)

    $(25)_{10} = (11001)_2 = \mathbf{(00011001)_{C2}}$
  - Pour coder un nombre négatif en C2, on code la valeur absolue en binaire pur, on inverse les bits (C1), puis on ajoute un 1

    Exemple : $(-25)_{10} = (?)\ en\ C2\ sur\ 8\ bits$

    $(25)_{10} = (00011001)_2$

    $C1(00011001)_2 = 11100110$

    $(-25)_{10} = (11100110) + 1 = \mathbf{(11100111)_{C2}}$
  - Inversement : quel est l'équivalent décimal de $(00110111)_{C2}\ et\ (10110101)_{C2}$

    $(00110111)_{C2}$ est un nombre positif donc directement :

    $(00110111)_{C2} = 2^5 + 2^4 + 2^2 + 2^1 + 2^0 = 32 + 16 + 4 + 2 + 1 = +\mathbf{(55)_{10}}$

    Par contre $(10110101)_{C2}$ est un nombre négatif donc :

    $(10110101)_{C2} = -(C1(10110101) + 1) = -(01001010 + 1) = -(01001011)$

    $= -(2^6 + 2^3 + 2^1 + 2^0) = -(64 + 8 + 2 + 1) = -\mathbf{75_{10}}$
- Etendu du codage: sur un codage de n bits, il est possible de représenter les nombres : de $[(-2^{n-1})$ à $(2^{n-1} - 1)\ ]$; exemple :
  - pour <u>n=3</u> : l'étendu est de $[-4$ à $+3]$;
  - pour <u>n=8</u> : l'étendu est de $[-128$ à $+127]$

C2	valeur
$-2^2 2^1 2^0$	
0 0 0	0
0 0 1	+1
0 1 0	+2
0 1 1	+3
1 0 0	-4
1 0 1	-3
1 1 0	-2
1 1 1	-1

- Avantage du codage:
  - une seule représentation pour zéros, par exemple sur 4 bits: le 0 s'écrit 0000.
  - Un seul circuit pour réaliser l'addition et la soustraction puisque : $A - B = A + C2(B)$

3. Représentation des réels dans la machine

Problème comment indiquer à la machine la position de la virgule ?? Deux formats de représentation des nombres réels : la virgule fixe, et la virgule flottante.

3.1 Représentation en virgule fixe

Cette représentation est utilisée dans les premières machines, elle est constituée d'une partie entière et une partie décimale séparées par une virgule. Le nom de la "virgule fixe" vient du fait que la virgule est fixée.

Problème :

- Nombre de valeurs limité
- Pas une grande précision

3.2 Représentation en virgule flottante

Cette représentation est utilisée dans les machines actuelles, elle est dynamique , on écrit le nombre sous forme : $\boldsymbol{S, M * 2^E}$, avec : ***S***: Signe, ***M* :** Mantisse, et ***E*** :exposant

La norme IEEE 754

Le standard IEEE 754 $\rightarrow$ deux format : le format simple précision sur 32bits,
et le format double précision sur 64 bits. Seul le premier format est considéré dans ce cours.

Format IEEE 754 simple précision.

32 bits $\begin{cases} 1\ bit\ pour\ le\ signe \\ 8\ bits\ pour\ l'exposant \\ 23\ bits\ pour\ la\ mantisse \end{cases}$

Exposant biaisé, $E = e^{+127}$ (0111 1111)

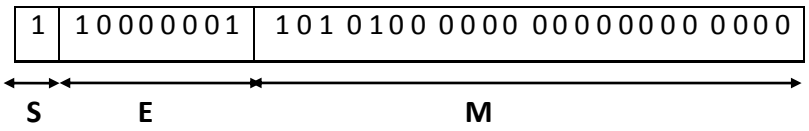
Exemple: convertir le nombre suivant $(\boldsymbol{-6.625})$ en binaire format IEEE simple précision.

Démarche

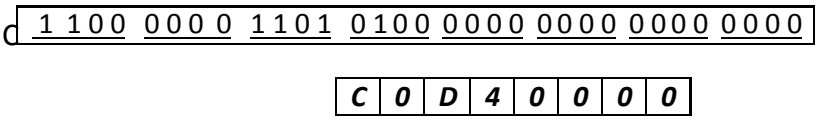
- Le signe: prend 1 si le nombre est négatif, 0 si le nombre est positif ⇻ **bit du signe =1.**
- Convertir la valeur absolue du nombre décimal en binaire virgule fixe ⇻ **110.101**
- Décaler la virgule de manière à ne laisser qu'un seul 1 à gauche de la virgule.
  ⇻ $\boldsymbol{1.10101 * 2^2}$ forme normalisée
- Le coté droit de la virgule construira la mentisse, à compléter par des zéros pour former 23 bits ⇻ **Mantisse= 101 0100 0000 0000 0000 0000**
- Pour calculer Exposant biaisé on applique la formule suivante:
  Exposant biaisé = Exposant normalisé + Biais. (avec biais =127).

$\mapsto \boldsymbol{E_b = 2 + 127 = 129_{10} = (10000001)_2}$

Ainsi: $(-6.625)_{10}$ en binaire format IEEE 754 simple précision est:



Ce qui est équivalent en hexadecimal à :



C	0	D	4	0	0	0	0

En résumé : $\boldsymbol{(-6.625)_{10} = (C0D40000)_{IEEE754}}$

4. L'arithmétique binaire

Les opérations arithmétique en binaire se font de la même manière qu'en décimal.

4.1 En binaire pur

a. Addition

0+ 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 plus une retenue

Exemple :

3,25	11,01
+ 7,75	+ 101,11
= 9,00	= 1001,00
En décimal	en binaire

b. Soustraction

0 – 0 = 0
0 – 1 = 1 retenue de 1
1 – 0 = 1
1 – 1 = 0

Exemple :

6,25	110,01
- 4,50	- 100,11
= 1,75	= 001,10
En décimal	en binaire

c. Multiplication

0 * 0 = 0
0 * 1 = 0
1 * 0 = 0
1 * 1 = 1

Exemple :

1,25	1,01
* 2,5	* 10,1
625	101
+250 .	+ 000.
=3,125	+ 101..
En décimal	= 11,001
	en binaire

d. Division

0 / 1 = 0
1 / 1 = 1

Exemple :

33	12	100001	1100
-24	2,75	- 1100	10, 11
= 090		= 0100 10	
-84		- 1100	
= 060		= 0110 0	
-60	En décimal	- 1100	En binaire
= 00		= 0000	

4.2 Opérations arithmétiques en Complément à 2

Interet : Même citcuit pour l'addition et la soustraction $[(A-B) = A + (-B)]$

L'addition est l'opération de base en arithétique binaire ; toutes les autres opérations reviennet à faire une suite d'additions, aussi nous ne considérons dans ce paragraphe que l'opération d'addition en C2.

Exemple1 **:** En décimal En Complément à 2

$+9$	0 1 0 0 1
-4	+ 1 1 1 0 0
$= +5$	= **1** 0 0 1 0 1

Débordement à ignorer

Résultat positif juste sur 5

$(+5)_{10} = (00101)_{C2}$

Exemple2 **:** En décimal En Complément à 2

-9	1 0 1 1 1
-4	+ 1 1 1 0 0
-13	= **1** 1 0 0 1 1

Débordement à ignorer

Résultat négatif juste sur 5

$(-13)_{10} = (10011)_{C2}$

Cas de débordement de signe **:**

Exemple 3**:** En décimal En Complément à 2

$+9$	0 1 0 0 1
$+8$	+ 0 1 0 0 0
$+17$	= **0** 1 0 0 0 1

Débordement de signe

Résultat négatif sur 5 = (-15) ce qui est faux

Le résultat iuste est sur 6 bits : (+17)

En effet l'étendu des valeurs qu'on peut représenter sur 5 bits est $[-16 + 15]$ donc 5 bits ne suffisent pas pour représenter +17

Exemple 4**:** En décimal En Complément à 2

-9	1 0 1 1 1
-8	+ 1 1 0 0 0
-17	= **1** 0 1 1 1 1

Débordement de signe

Résultat positif sur 5 = (+15) ce qui est faux

Le résultat iuste est sur 6 bits : (-17)

5 bits ne suffisent pas pour représenter -17, il y a eu donc un débordement de signe sur le 6ème bit

En Résumé :

- On a un débordement de signe quand l'addition de 2 nombres positifs donne une somme négative, ou bien l'addition de 2 nombres négatifs donne une somme positive.
- On n'a jamais un débordement de signe quand on additionne 2 nombres de signes contraires.

5. Codage

5.1 Introduction
5.2 Les codes décimaux.
5.3 Le code Gray
5.4 Codes correcteurs d'erreurs
5.5 *Codage des caractères : code ASCII*