Chapitre 2 : Représentation des Informations

Les Systèmes de Codage

Codes Numériques pondérés

Le code binaire pur est un code pondéré par des puissances de 2, utilisé en arithmétique binaire. Ses dérivées sont le code octal et le code hexadécimal.

- Exemples BCD (Décimal codé en Binaire), le code Aiken

Codes Numériques non pondérés

Dans ces types de code, aucun poids est affecté à la position d'un bit. On convient simplement d'un tableau de correspondances entre les objets à coder et une représentation.

- Exemple de Codes Numériques non pondérés Code de Gray, Code ASCII

Codes Numériques pondérés

Code BCD

- Dans ce système de codage, chaque chiffre est codé par son équivalent en binaire sur quatre bits. C'est un code pondéré avec les poids 1, 2, 4, 8, 10, 20, 40, 80, 100,200,400,800,1000..
- Exemple : $1998_{10} = 1111100110_2 = 0001100110011000_{BCD}$

Code binaire d'Aiken

- Ce code est pondéré par 2,4,2,1. Il peut être constitué par les règles suivantes :
- de 0 à 4 on code en binaire pur ;
- de 5 à 9 on ajoute 6 et on code en binaire pur. (c.à.d. 5 –> 5 + 6 = 11, 6 –> 6 + 6 =12, . . .)

Code binaire d'Aiken

Décimal	Aiken
	2 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	1 0 1 1
6	1 1 0 0
7	1 1 0 1
8	1 1 1 0
9	1 1 1 1

Codes Numériques non pondérés

Code de Gray (binaire réfléchi)

- Un seul bit change entre deux nombres consécutifs. On y trouve la notion d'adjacence entre deux termes. Le code présente des symétries miroir. Il est cyclique : il se referme sur lui-même.

Code de Gray

Décimal	Gray
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

Conversion Gray-Binaire/ Binaire-Gray

- Pour convertir un nombre en code binaire naturel (CBN) vers un nombre en code binaire réechi (CBR):
  - On ajoute le CBN trouvé à lui-même décalé d'un rang vers la gauche, sans tenir compte de l'éventuelle retenue
  - On abandonne dans le résultat le bit de poids faible OU
  - On addition (XOR) le bit de rang n-1 au bit de rang n, celui de rang n-2 au rang n-3, ... celui de rang 1 au bit de rang 0

- Exemple :

```
  1 0 0 1
1 0 0 1
———————
1 1 0 1 1 ⇐bit à ignoré
```

- Pour convertir un nombre du code de Gray (CBR) vers le un nombre en code binaire naturel (CBN) :
  - On abaisse le bit de poids fort (soit de rang n du CBR) et on fait un XOR de ce même bit abaissé avec le bit de de rang (n-1) du CBR.
  - Ensuite on fait un XOR du bit récemment obtenu avec celui de rang(n-2) du CBR. Ainsi de suite jusqu' à atteindre le bit de rang 0 du CBR

Codes Numériques non pondérés

Données non numériques :Codes ASCII , UNICODE

- Ils servent à coder des chiffres, des lettres, des signes de ponctuations et des caractères spéciaux.
- Le codage est réalisé par une table de correspondance, propre à chaque code utilisé.
- ASCII (American Standard Code for Information Interchange) à 7 ou 8 bits
- UNICODE (Extended Binary Coded Decimal Internal Code) à 16 bits

Addition en BCD

L'addition de deux nombres codés en DCB revêt une certaine particularité que nous examinons à travers des exemples.

Résultat inférieur à 9

- Pas de problème tant que le résultat est inférieur ou égal à 9

```
  0 1 0 0   0 1 0 1        4 5
+                        +
  0 1 0 0   0 0 1 1        4 3
-------------------      -----
  1 0 0 0   1 0 0 0        8 8
```

Quand le résultat est supérieur à 9

- Apporter une correction en additionnant 6, afin d'obtenir une réponse valide. Ceci est dû au fait que l'on représente un nombre modulo 10 avec un code modulo 16 : 16-10 = 6.

```
• Exemple 2 :
  0 1 1 0
+ 0 1 1 1
= 1 1 0 1
> 9, on ajoute 6
+ 0 1 1 0
= 0 0 0 1 0 0 1 1
    1   3

• Exemple 3 :
  1 0 0 0 0 1 1 1 1 0 0 1 0 1 0 1      8 7 9 5
+ 0 1 1 0 0 1 0 0 0 0 1 1          +      6 4 3
= 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 0      9 4 3 8
+ 0 1 1 0 0 1 1 0
= 1 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0
  9   4   3   8
```


Soustraction en BCD

Résultat inférieur à 9

- Dans l'addition, on ajoute 6 quand la somme de deux motifs de 4 bits dépasse 9. Pour la soustraction :
  - lorsque le résultat de la soustraction DCB est inférieure à 9, on ne change pas le résultat
  - lorsque le résultat de la soustraction DCB est supérieure à 9, on soustrait 6 résultat
  - lorsqu'il y a une retenue soustractive, on soustrait également 6 au résultat obtenu, même si la valeur est inférieure à 9

```
• Exemple 1 :
  2 0          0 0 1 0 0 0 0 0
- 8          - 1 0 0 0
------       -------------
  1 2        = 0 0 0 1 1 0 0 0
emprunt, on soustrait 6
                 - 0 1 1 0
                 -------------
                   0 0 0 1 0 0 1 0
                     1   2

• Exemple 2 :
  2 2          0 0 1 0 0 0 1 0
- 3          - 0 0 1 1
------       -------------
  1 2        = 0 0 0 1 1 1 1 1   emprunt et résultat
                                 supérieur à 9, on soustrait 6
               - 0 1 1 0
               -------------
                 0 0 0 1 1 0 0 1
                   1   9
```

Représentation des nombres

REPRESENTATION DES NOMBRES

- Représentation binaire des Nombres entiers positifs
- Représentation binaire des Nombres entiers signés
- Représentation binaire des Nombres réels

Représentation binaire des Nombres entiers positifs

Représentation binaire des Nombres entiers positifs

- Les nombres sont représentés en binaire sur n bits : n=nombre d'unités mémoires (n = 8, 16, 32, 64, ...) On peut représenter des nombres allant de 0 à $2^n - 1$.

Représentation binaire des Nombres entiers signés

- Traditionnellement on met un signe " - " pour représenter les nombres négatifs. Mais les systèmes logiques ne permettent de présenter qu'un des deux symboles " 0 " et " 1 ", il faut chercher une convention pour remplacer le " - "

Représentation binaire des Nombres entiers signés

- Représentation en module et signe
  - Un élément binaire est ajouté à gauche du module pour représenter le signe. Ainsi un nombre commençant par un " 0 " sera positif alors qu'un nombre commençant par un " 1 " sera négatif.
  - Exemple : Avec 4 éléments binaires les valeurs vont de -7 à+ 7.

Signe	Module	Valeur	Signe	Module	Valeur
1	111	-7	0	111	7
1	110	-6	0	110	6
1	101	-5	0	101	5
1	100	-4	0	100	4
1	011	-3	0	011	3
1	010	-2	0	010	2
1	001	-1	0	001	1
1	000	0	0	000	0

- Problème : on a ici deux représentations pour le zéro

Représentation binaire des Nombres entiers négatifs

- Représentation en complément restreint (CR) ou complément à 1(C1) ou complément logique
  - $-A = \overline{A}$:
  - Pour prendre l'inverse d'un nombre, il suffit de le complémenter (inversion de tous ses bits) comme dans le cas précédent, la nature du premier bit donnera le signe : 0 = +, 1= -.
  - Avec 4 bits : $+5 = 0101$; $-5 = 1010$
  - Problème : de nouveau,on a ici deux représentations pour le zéro
- Représentation en complément vrai (CV) ou complément à 2 (C2) ou ou complément arithmétique
  - C'est la représentation la plus utilisée. Le bit le plus à gauche est encore le bit de signe : 0 = + et 1 = -.
  - $-A = A+1$
  - Une seule représentation pour le 0
  - avec des mots de n bits, on obtient 2^n valeurs différentes, de 0 à 2^{n-1}-1 pour les valeurs positives, et de -1 à -2^{n-1} pour les valeurs négatives ;

Représentation binaire des Nombres entiers négatifs

- Représentation en complément vrait (CV) ou complément à 2 (C2) ou ou complément arithmétique (suite)
  - Exemple avec 4 bits :
    - Valeurs positives de 0 à 7
    - Valeurs négativeses de -1 à -8
  - Exemple avec 3 bits :

décimal	Module et signe	complément à 1	complément à 2
+3	011	011	011
+2	010	010	010
+1	001	001	001
+0	000	000	000
-0	100	111	....
-1	101	110	111
-2	110	101	110
-3	111	100	101
-4	.....	.....	100

Représentation binaire des Nombres Réels

Pour représenter les nombres fractionnaires il est nécessaire de définir la position de la virgule. Pour ce faire, il existe deux méthodes :

Représentation en virgule fixe

- On décide que la virgule soit toujours à une position donnée (un entier peut être représentatif d'un nombre fractionnaire si on connaît la place de la virgule). Tout nombre est mis sous la forme a,b avec "a " chiffre(s) avant la virgule et " b " chiffre(s) après la virgule.
- Exemple : x,y= 9,75 : x= 9 , y= 75 ; a= 5 ; b= 3
  La position de la virgule est fixée.
- 9,75 =01001,110

Représentation en virgule flottante

- Le nombre N est représenté sous la forme : $\underline{exposant}\underline{mantisse}$.Deux approches existent.

Représentation en virgule flottante

- Première approche
  - Soit N = $a_3a_2a_1a_0, a_{-1}a_{-2}a_{-3}$
  - N peut se noter : $\underbrace{a_6a_5a_4}_{ma}a_3\underline{a_2a_1a_0}\,2^{-3}$. Multiplication implicite par 2^{-3}
  - $\Longrightarrow$ $exposant = -3$
    $mantisse = a_6a_5a_4a_3a_2a_1a_0$
  - Les valeurs de la mantisse et l'exposant peuvent être notées en complément à 2.
  - Exemple : Soit la mémoire de taille suivante : $\underset{ex}{4\underline{bits}}\ \ 12\underline{bits}$. Coder la valeur 26,75 en virgule flottante.
  - $(26,75)_{10} = (11010,110)_2$
    $(11010,11)_2 = (11010110).2^{-3}$
  - $\Longrightarrow$ $exposant = -3$
    $mantisse = 11010110_2$
  - 1101 00001101011 0
    exp^{26}, 75_{10}=214.2^{-3} avec (-3 en complément à 2 =1101)

Représentation en virgule flottante

- La manière la plus évidente et la plus concise pour représenter un nombre en virgule flottante est donc d'employer un exposant et une mantisse signée.
- Exemple : $-123,45_{10} = -0,12345.10^{+3}$;$0,0000678_{10} = +0,678.10^{-4}$
- On peut noter également qu'à priori, une représentation en virgule flottante n'est pas nécessairement unique.
- Exemple : 0, 2340.10^{+2}=0, 002340.10^{+4}=0, 00002340.10^{+6}
- Il nous faut donc les représenter sous une forme **normalisée** afin que la représentation ne varie pas d'un matériel à l'autre.

Représentation en virgule flottante

- Deuxième approche : Normalisation
  - C'est la méthode inverse de la précédente : on considère que le bit le plus à gauche de la mantisse a pour poids 2^{-1}
  - Ainsi, un nombre normalisé, en virgule flottante, est un nombre dans lequel le chiffre suivant la marque décimale, à gauche de la mantisse (donc à droite de la marque décimale), n'est pas un zéro alors que le nombre à gauche de la marque décimale est un zéro.
  - Soit N=$a_3a_2a_1a_0, a_{-1}a_{-2}a_{-3}$, N peut se noter : $0.\underbrace{a_{-1}a_{-2}a_{-3}a_{-4}a_{-5}a_{-6}a_{-7}}_{mantisse} 2^{\overbrace{-4}^{exp}}$.
  - $(26,75)_{10}$ =$(11010,110)_2$=$(0,0011010110)_2.2^7$ n'est pas normalisé
  - Par contre $(26,75)_{10}$ =$(0,11010110)_2.2^5$ est normalisé. On peut omettre le 0 dans la représentation : $\Longrightarrow (,11010110)_2.2^5$
  - $(26,75)_{10}$ peut s'écrire : $\underbrace{0101}_{exposant}\ \underbrace{11010110000}_{mantisse}$
  - On peut trouver, en fonction des organismes de normalisation ou des constructeurs, plusieurs "normes" de représentation des nombres en virgule flottante (IEEE, IBM, ...), nous nous bornerons à présenter ici les normes IEEE.

Représentation en virgule flottante : Norme IEEE

- Le stabdard IEEE définit trois formats de représentation des nombres réels : la simple précision (sur 32 bits), la double précision (sur 64 bits) et la précision étendue (sur 80 bits). Ce dernier est surtout destiné à réduire les erreurs d'arrondis de calculs.

1 bit	8 bits	23 bits
signe manstisse	exposant	mantisse

1 bit	11 bits	52 bits
signe manstisse	exposant	mantisse

- Chaque format commence par un bit de signe de la mantisse (celui le plus à gauche), qui vaut 0 pour les nombres positifs et 1 pour les nombres négatifs. Puis l'exposant est codé en décalage (on dit aussi en excédant) par rapport à l'exposant de référence : 127 pour la simple précision, et 1023 pour la double précision. Enfin la mantisse, est codée en binaire sur 23 ou 52 bits.

Représentation en virgule flottante : Norme IEEE

- Fonctionnement
  - Exemple : pour normaliser le nombre décimal 10, 5_{10} en virgule flottante, format simple précision dans la base 2 (binaire), avec 64 comme exposant de référence.
  - Il conviendra donc dans un premier temps de transcrire notre nombre 10,50 en base 2, ce qui nous donne $1010,1_2$
  - Ensuite il faut "normaliser" (décaler la virgule vers la gauche de façon à trouver une forme normalisée), ce qui donne donc : **0**, 10101.2^4.
  - L'exposant est codé en décalage (on dit aussi en excédant) par rapport à l'exposant de référence, à savoir : 64_{10}.
  - On a donc : Exposant de référence + Décalage = Exposant décalé Soit 64_{10} + 4_{10} = 68_{10} soit encore 1000100_2
  - Le signe du nombre étant positif, le bit représentatif du signe sera donc positionné à zéro. Nous aurons ainsi en définitive :

0	01000100	10101000000000000000000
signe manstisse	exp. code excédent 64	mantisse

Représentation en virgule flottante : Norme IEEE

- Fonctionnement (suite)
  - Exemple : normaliser le nombre 432_{10} dans la base 2 exprimé sous la forme $0,0000000000011011_2.2^{20}$ avec un exposant de référence 64_{10}
  - $0,0000000000011011_2.2^{20}$=$0,1101100000000000_2.2^9$
  - On a donc : Exposant de référence + Décalage = Exposant décalé Soit 64_{10} + 9_{10} = 73_{10} soit encore 1001001_2

0	01001001	11011000000000000000000
signe manstisse	exp. code excédent 64	mantisse

Addition en Complément à 1 ou à 2

- En complément à 1 ou à 2, la soustraction d'un nombre se réduit à l'addition de son complément.
- Dans une addition en complément à 1, une retenue générée par le bit de signe doit être ajoutée au résultat obtenu. Par contre en complément à 2, on ignore cette retenue.
- Lorsqu' on utilise ce procédé, les nombres doivent avoir le même nombre de bits, si besoin est, on ajoute des zéros à l'un des nombres.
  - Exemple 1 : soustraction sur 5 bits : 7-2= 7+(-2)
  - $7_{10} = 00111_2$ et $2_{10} = 00010_2$
  - -2 en complément à 1 =11101
  - -2 en complément à 2 =11110
  - En complément à 1 : 7+(-2)

```
    00111
    11101
    -----
1<==00100
      + 1 -> retenue du bit de signe
    -------
    00101
```

En complément à 2 : 7+(-2)

```
  00111
  11110
  -----
1<-00101 on ne tient pas compte
           de la retenue
```


Addition en Complément à 1 ou à 2

- Exemple 2 : soustraction sur 5 bits : 3-6= 3+(-6)
- $3_{10} = 00011_2$ et $6_{10} = 00110_2$
- -6 en complément à 1 =11001
- -6 en complément à 2 =11010
- En complément à 1 : 3+(-6)

```
    00011
+
    11001
    -----
sans retenue 11100
à ajouter au résultat.
```

En complément à 2 : 3+(-6)

```
  00011
  11010
  -----
11101 sans retenue à
        ignorer
```

- On trouve un résultat en complément à 1 ou à 2.