Troisième Partie

Plan

Conversion de Bases

- Définition des différentes bases
- Conversion vers la base 10
- Conversions de la base 10 vers d'autres bases

Codage des informations

La numération arabe utilise une représentation positionnelle contrairement à la numération romaine : le rang de chaque chiffre indique son poids.

Numération romaine : MCMXCIX où M vaut toujours 1000, C vaut toujours 100, etc.

Numération positionnelle : 1999 où le « 9 » le plus à droite vaut « 9 », celui immédiatement à sa gauche vaut « 90 », etc. La valeur d'un chiffre dépend de sa position.

XXX_b indique que le nombre XXX est écrit en base *b*.

Exemples :

- $10^3\ 10^2\ 10^1\ 10^0$
- $5\ 9\ 3\ 1_{10} = 5*1000+9*100+3*10+1*1$
- $2^3\ 2^2\ 2^1\ 2^0$
- $1\ 0\ 0\ 1_2 = 1*8+0*4+0*2+1*1 = 9_{10}$
- $5^2\ 5^1\ 5^0$
- $1\ 4\ 3_5 = 1*25+4*5+3 = 48_{10}$

Codage des informations

- Bases usuelles : base 10, 8.
- Représentation des nombres dans une base *b* :

  1. Si *b* <= 10, on utilise simplement les chiffres de 0 à *b*-1 (exemple : base 2).

  2. Si *b* > 10, on utilise les chiffres de 0 à 9 et des lettres.

Ainsi, en base 16 (numération hexadécimale) on utilise les chiffres de 0 à 9 et les lettres de A (=10) à F (=15).

$E2A1_{16} = 14*16^3+2*16^2+10*16+1 = 58017_{10}$

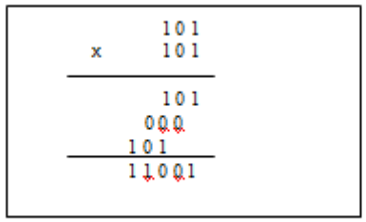
Exemple : l'adresse mémoire.

Opérations arithmétiques

Les principes sont les mêmes que ceux utilisés en bc

```
  10101100
+   1100101
= 100010001
```

Exemples de calculs en base 2 (172 + 101 = 273



– Tables d'addition et de multiplication en base 2.

Dans le système de calcul binaire la soustraction ressemble en quelque sorte à l'addition, sauf que lorsque l'on soustrait un bit égal à un d'un bit égal à zéro, on obtient une retenue pour le bit de poids plus élevé.

Principe de base :

$0 - 0 = 0$ $\quad$ $0 - 1 = 1$ (avec retenue)

$1 - 0 = 1$ $\quad$ $1 - 1 = 0$

Exemple de soustraction :

```
   r   r r r
 1 1 0 1 1 1 0
 -   1 0 1 1 1
 ---------------
 1 0 1 0 1 1 1
```

Les conversions

Conversion Binaire décimal:

Passage d'une base quelconque à la base 10

Il suffit d'écrire le nombre comme ci-dessus et d'effectuer les opérations en décimal.

Exemple en hexadécimal :

$(AB)_{16} = 10 * 16^1 + 11 * 16^0 = 160 + 11 = (171)_{10}$

(En base 16, A représente 10, B → 11, et F→ 15).

Les conversions

Passage de la base 10 vers une base quelconque

Nombres entiers : On procède par divisions successives. On divise le nombre par la base, puis le quotient obtenu par la base, et ainsi de suite jusqu'a l'obtention d'un quotient nul.

La suite des restes obtenus correspond aux chiffres dans la base visée, $a_0a_1 \ldots a_n$.

Exemple : soit à convertir $(44)_{10}$ vers la base 2.

$44 = 22 \times 2 + 0 \Rightarrow a_0 = 0$

$22 = 11 \times 2 + 0 \Rightarrow a_1 = 0$

$11 = 2 \times 5 + 1 \Rightarrow a_2 = 1$

$5 = 2 \times 2 + 1 \Rightarrow a_3 = 1$

$2 = 1 \times 2 + 0 \Rightarrow a_4 = 0$

$1 = 0 \times 2 + 1 \Rightarrow a_5 = 1$

Donc $(44)_{10} = (101100)_2$.

Les conversions

Nombres fractionnaires : On multiplie la partie fractionnaire par la base en répétant l'opération sur la partie fractionnaire du produit jusqu'a ce qu'elle soit nulle (ou que la précision voulue soit atteinte).

Pour la partie entière, on procède par divisions comme pour un entier.

- Exemple : conversion de $(54, 25)_{10}$ en base 2

  Partie entière : $(54)_{10} = (110110)_2$ par divisions.

  Partie fractionnaire :

- $0,25 \times 2 = 0,50 \Rightarrow a_{-1} = 0$
- $0,50 \times 2 = 1,00 \Rightarrow a_{-2} = 1$
- $0,00 \times 2 = 0,00 \Rightarrow a_{-3} = 0$

Les conversions

Cas des bases 2, 8 et 16

Ces bases correspondent à des puissances de 2 (2^1, 2^3 et 2^4), d'où des passages de l'une à l'autre très simples. Les bases 8 et 16 sont pour cela très utilisées en informatique, elles permettent de représenter rapidement et de manière compacte des configurations binaires.

La base 8 est appelée notation octale, et la base 16 notation hexadécimale.

Chaque chiffre en base 16 (2^4) représente un paquet de 4 bits consécutifs. Par exemple :

$(10011011)_2 = (1001\ 1011)_2 = (9B)_{16}$

De même, chaque chiffre octal représente 3 bits.

On manipule souvent des nombres formés de 8 bits, nommés octets, qui sont donc notés sur 2 chiffres hexadécimaux.

Les conversions

Exemples :

Décimal → binaire : on procède par division entière successive par 2. Exemple : 29_{10} :

29 : 2 = 14 reste 1

14 : 2 = 7 reste 0

7 : 2 = 3 reste 1

3 : 2 = 1 reste 1

1 : 2 = 0 reste 1 donc $29_{10} = 11101_2$:

Les conversions

Exemple : $14_5 = 9_{10} = 12_7$:

– Conversion de la base 5 vers la base 7: division entière par $7 = 12_7$ en base 7.

$14_5 : 12_5 = 1$ reste 2

$1_5 : 12_5 = 0$ reste 1

Le codage de 14_5 est donc 12_7.

– Conversion de la base 7 vers la base 5 : division entière par $5 = 5_7$ en base 7.

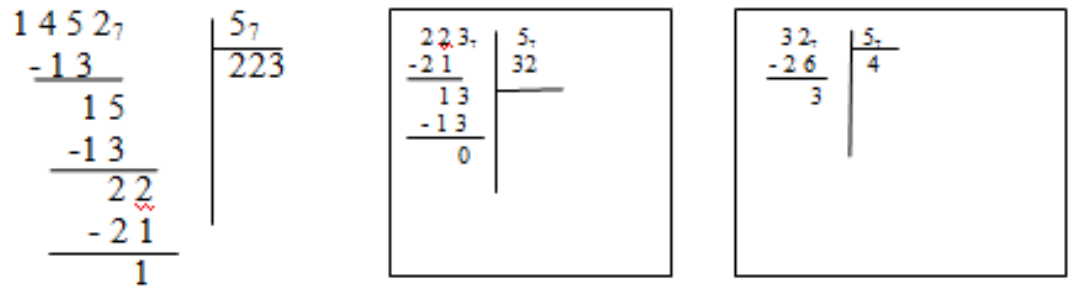
x	0_7	1_7	2_7	3_7	4_7	5_7	6_7	10_7
5_7	0_7	5_7	13_7	21_7	26_7	34_7	42_7	50_7

- $12_7 : 5_7 = 1$ reste 4
- $1_7 : 5_7 = 0$ reste 1
- Le codage de 12_7 est donc 14_5.

Les conversions

REPRÉSENTATION DES ENTIERS

Exemple plus compliqué : conversion de 1452_7 en base 5



D'où $1452_7 = 4301_5$. Les sceptiques peuvent vérifier que $1452_7 = 1*7^3+4*7^2+5*7+2 = 576$ et $4301_5 = 4*5^3+3*5^2+0*5+1 = 576$...

En pratique, si *b*1 != 10, on évite d'effectuer la division en base b_1 en convertissant *X* en base 10 puis en procédant par division sur le nombre obtenu pour effectuer la conversion de la base 10 vers la base b_2.

Certaines conversions sont très faciles à réaliser comme la conversion binaire vers octal (base 8) ou binaire vers hexadécimal (base 16).

Les conversions

Exemple : 1010011101_2

– base 8 découpages par blocs de 3 chiffres :

001 010 011 101

1 2 3 5 $= 1235_8$

– base 16 découpages par blocs de 4 chiffres :

0010 1001 1101

2 9 D $= 29D_{16}$

De manière générale, les conversions sont faciles lorsque b_2 est une puissance de b_1. Les conversions en base 8 ou 16 sont très fréquentes pour l'affichage des nombres binaires qui, par leurs longueurs, sont rapidement illisibles.

Exemple : octal pour les codes de caractères, hexadécimal pour les adresses mémoires.

Autres Opérations

Opérations arithmétiques

Les opérations arithmétiques s'effectuent en base quelconque b avec les mêmes méthodes qu'en base 10. Une retenue ou un report apparait lorsque l'on atteint ou dépasse la valeur b de la base.

Représentation des entiers

Codage machine

La représentation (ou codification) des nombres est nécessaire afin de les stocker et manipuler par un ordinateur. Le principal problème est la limitation de la taille du codage : un nombre mathématique peut prendre des valeurs arbitrairement grandes, tandis que le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fixé.

Codage

Entiers naturels

Les entiers naturels (positifs ou nuls) sont codés sur un nombre d'octets fixé (un octet est un groupe de 8 bits). On rencontre habituellement des codages sur 1, 2 ou 4 octets, plus rarement sur 64 bits (8 octets, par exemple sur les processeurs DEC Alpha).

Un codage sur n bits permet de représenter tous les nombres naturels compris entre 0 et 2^n - 1. Par exemple sur 1 octet, on pourra coder les nombres de 0 à 255 = 2^8 - 1.

On représente le nombre en base 2 et on range les bits dans les cellules binaires correspondant à leur poids binaire, de la droite vers la gauche. Si nécessaire, on complète à gauche par des zéros (bits de poids fort).

Codage

Entiers relatifs

Il faut ici coder le signe du nombre. On utilise le codage en complément à deux, qui permet d'effectuer ensuite les opérations arithmétiques entre nombres relatifs de la même façon qu'entre nombres naturels.

1. Entiers positifs ou nuls : On représente le nombre en base 2 et on range les bits comme pour les entiers naturels. Cependant, la cellule de poids fort est toujours à 0 : on utilise donc n - 1 bits.

   Le plus grand entier positif représentable sur n bits en relatif est donc $2^{n-1}-1$.

Codage des entiers négatif

Pour obtenir le codage d'un nombre x négatif, on code en binaire sa valeur absolue sur n - 1 bits, puis on complémente (ou inverse) tous les bits et on ajoute 1.

Exemple : soit à coder la valeur -2 sur 8 bits. On exprime 2 en binaire, soit

00000010. Le complément à 1 est 11111101. On ajoute 1 et on obtient le résultat : 1111 1110.

Remarque :

- ✓ le bit de poids fort d'un nombre négatif est toujours 1 ;
- ✓ sur n bits, le plus grand entier positif est $2^{n-1} - 1 = 011 \ldots 1$;
- ✓ sur n bits, le plus petit entier négatif est -2^{n-1}.

Représentation des caractères: Code ASCII

Les caractères sont des données non numériques : il n'y a pas de sens à additionner ou multiplier deux caractères. Par contre, il est souvent utile de comparer deux caractères, par exemple pour les trier dans l'ordre alphabétique.

Les caractères, appelés symboles alphanumériques, incluent les lettres majuscules et minuscules, les symboles de ponctuation (& ~ , . ; # " - etc...), et les chiffres.

Un texte, ou *chaîne de caractères*, sera représenté comme une suite de caractères.

Le codage des caractères est fait par une table de correspondance indiquant la configuration binaire représentant chaque caractère. Les deux codes les plus connus sont l'EBCDIC (en voie de disparition) et le code ASCII (American Standard Code for Information Interchange).

Le code ASCII représente chaque caractère sur 7 bits (on parle parfois de code ASCII étendu, utilisant 8 bits pour coder des caractères supplémentaires).

Code ASCII

Notons que le code ASCII original, défini pour les besoins de l'informatique en langue anglaise ne permet la représentation des caractères accentués (é, è, à, ù, ...), et encore moins des caractères chinois ou arabes. Pour ces langues, d'autres codages existent, utilisant 16 bits par caractères. A chaque caractère est associée une configuration de 8 chiffres binaires (1 *octet*), le chiffre de poids fort (le plus à gauche) étant toujours égal à zéro. La table indique aussi les valeurs en base 10 (décimal) et 16 (hexadécimal) du nombre correspondant.

Plusieurs points importants à propos du code ASCII :

– Les codes compris entre 0 et 31 ne représentent pas des caractères, ils ne sont pas affichables. Ces codes, souvent nommés *caractères de contrôles* sont utilisés pour indiquer des actions comme passer à la ligne (CR, LF), émettre un bip sonore, etc.

– Les lettres se suivent dans l'ordre alphabétique (codes 65 à 90 pour les majuscules, 97 à 122 pour les minuscules), ce qui simplifie les comparaisons.

– On passe des majuscules aux minuscules en modifiant le 5ième bit, ce qui revient à ajouter 32 au code ASCII décimal.

– Les chiffres sont rangés dans l'ordre croissant (codes 48 à 57), et les 4 bits de poids faibles définissent la valeur en binaire du chiffre.

Récapitulatif

– Codage en binaire, ici un chiffre est appelé un bit (binary digit : chiffre binaire).

– Les nombres sont codés sur *n* octets, généralement 2 (*short* en Langage C) ou 4 (*int* ou *long* en langage C).

– *m* bits --> 2^m nombres différents, si $n = 2$ ---> $2^{16} = 65\ 536$ nombres, si $n = 4$ ---> $2^{32} = 4294\ 967\ 296$ nombres.

– le problème c'est que la capacité de la mémoire est limitée. Si le résultat de l'opération est supérieur au nombre maximum représentable ----> *overflow* (dépassement de capacité).

Exemple sur 4 bits : 9 + 7 = 16 qui ne peut être stocké sur 4 bits...

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

Récapitulatif

Complément binaire et codage des entiers négatifs

– En binaire, le principe du complément est le même, le complément à 1 revient simplement à inverser les bits d'un nombre et le complément à 2 ajoute 1 au complément à 1.