Chap. 0 : Rappels - Représentations des données

Laurent Poinsot

UMR 7030 - Université Paris 13 - Institut Galilée

Cours “Architecture et Système”

Dans le cours d'"Architecture des Ordinateurs et Système" seront présentés les principes de fonctionnement des ordinateurs. Le but étant de comprendre l'organisation "interne" de ces machines que vous utilisez tous les jours (ou presque !). Pour ce qui concerne l'aspect "Architecture" du cours (la première partie), nous étudierons de façon détaillée l'architecture du PC, son processeur et le langage machine, les fonctions de base de son système d'exploitation, et ses mécanismes de communication avec l'extérieur (entrées/sorties). Nous évoquerons également le fonctionnement de différents périphériques de l'ordinateur (écran, clavier, disques durs, CD-ROMs...).

Introduction

Les informations traitées par un ordinateur peuvent être de différents types (texte, nombres, etc.) mais elles sont toujours représentées et manipulées par l'ordinateur sous forme binaire. Toute information sera traitée comme une suite de 0 et de 1. L'unité d'information est le chiffre binaire (0 ou 1), que l'on appelle bit (pour "binary digit", chiffre binaire en anglais). Le codage d'une information consiste à établir une correspondance entre la représentation externe (habituelle) de l'information (le caractère A ou le nombre 36 par exemple), et sa représentation interne dans la machine, qui est une suite de bits. On utilise la représentation binaire car elle est simple, facile à réaliser techniquement à l'aide de bistables (système à deux états réalisés à l'aide de transistors). Enfin, les opérations arithmétiques de base (addition, multiplication, etc.) sont faciles à exprimer en base 2 (par exemple, la table de multiplication est 0x0 = 0, 1x0 = 0 et 1x1 = 1).

Plan du chapitre

Dans ce premier chapitre, constitué d'une série de rappels, nous verrons comment sont codées en interne les données manipulées par un ordinateur. Ce chapitre est articulé de la façon suivante :

1. Les bases de numération ;
2. Les changements de base ;
3. Codage des nombres entiers ;
4. Représentation des caractères ;
5. Représentation des nombres réels (norme IEEE).

Avant d'aborder la représentation des différents types de données (caractères, nombres naturels, nombres réels), il convient de se familiariser avec la représentation d'un nombre dans une base quelconque (par la suite, nous utiliserons souvent les bases 2, 8, 10 et 16). Habituellement, on utilise la base 10 pour représenter les nombres, c'est-à-dire que l'on écrit à l'aide de 10 symboles distincts, les chiffres (décimaux). En base b, on utilise b chiffres.

Exemples

1. Décimal : $b = 10$ et les chiffres sont $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$;
2. Binaire : $b = 2$ et les chiffres sont 0 et 1 ;
3. Octal : $b = 8$, base employée en informatique, aujourd'hui abandonnée au profit de la base 16. Elle a été utilisée par les Yukis (tribu indienne) ;
4. Duodécimal : $b = 12$, et a été employée de manière embryonnaire par les Égyptiens pour le compte en heures et mois ;
5. Hexadécimal : $b = 16$ et les chiffres sont $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$ (on utilise les 6 premières lettres de l'alphabet comme des chiffres) ;
6. Vigésimal : $b = 20$, base qui a été utilisée par les Mayas et les Aztèques, ainsi que dans la France du Moyen Âge (vestiges : "quatre-vingt", "quatre-vingt-dix") ;

Exemples

Historiquement d'autres bases furent jadis employées :

1. Sexagésimal : la base 60 employée par les Sumériens, les Akkadiens, puis les Babyloniens pendant l'Antiquité. Ce système de numération sert encore actuellement dans la mesure du temps et des angles ;
2. La base 150 ou base indienne, utilisée notamment en astronomie par les Indiens.

En base b, on utilise b chiffres. Notons a_i la suite des chiffres utilisés pour écrire un nombre

$$x = a_n a_{n-1} \cdots a_1 a_0 \tag{1}$$

où a_0 est le chiffre des unités.

1. En décimal, $b = 10$, $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$;
2. En binaire, $b = 2$, $a_i \in \{0, 1\}$;
3. En hexadécimal, $b = 16$,
   $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$.

En base 10, on écrit par exemple “1993” pour représenter le nombre

$$1993 = 1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 3 \times 10^0 \qquad (2)$$

(Pour rappel, $n^0 = 1$ quel que soit $n > 0$.)
Dans le cas général, en base b, le nombre représenté par une suite de chiffres $a_n a_{n-1} \cdots a_1 a_0$ est donné (en base dix) par :

$$a_n a_{n-1} \cdots a_1 a_0 = \sum_{i=0}^{n} a_i b^i \ . \qquad (3)$$

a_0 est appelé le chiffre de poids faible, et a_n le chiffre de poids fort.

Exemple en base 2 :

$$(1101)_2 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 = 8+4+0+1 = 13\,. \quad (4)$$

La notation $()_b$ indique que le nombre est écrit en base b (on peut parfois s'en passer !).

Il est également possible de représenter des nombres fractionnaires. Les nombres fractionnaires sont ceux qui comportent des chiffres après la virgule. Dans le système décimal, on écrit par exemple :

$$12,346 = 1 \times 10^1 + 2 \times 10^0 + 3 \times 10^{-1} + 4 \times 10^{-2} + 6 \times 10^{-3} \ . \quad (5)$$

En général, en base b, on écrit :

$$\begin{aligned} a_n a_{n-1} \cdots a_1 a_0 , a_{-1} a_{-2} \cdots a_{-p} &= a_n b^n + a_{n-1} b^{n-1} + \cdots + a_0 b^0 \\ &+ a_{-1} b^{-1} + \cdots + a_{-p} b^{-p} \ . \end{aligned} \quad (6)$$

Passage d'une base quelconque à la base 10

C'est le changement le plus facile à effectuer. En effet, il suffit d'écrire le nombre comme dans les équations (3) si c'est un entier, ou (6) si c'est un nombre fractionnaire. Par exemple, nous avons en hexadécimal :

$$(AB)_{16} = 10 \times 16^1 + 11 \times 16^0 = 160 + 11 = (171)_{10} \qquad (7)$$

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

Passage de la base dix à une base b quelconque

Le passage de la base dix à une base quelconque utilise l'opération de division entière. Soient a, b deux entiers positifs, et $b \neq 0$. Alors il existe un unique couple d'entiers positifs (q, r) tels que

$$a = bq + r \text{ et } 0 \leq r < b \ . \tag{8}$$

q est le quotient dans la division entière de a par b.
r est le reste dans la division entière de a par b.
Remarques : Si $a = b$, alors $q = 1$ et $r = 0$.
Si $a < b$, alors $q = 0$ et $r = a$.

Passage de la base dix à une base b quelconque

Exemples :

$10 = 3 \times 3 + 1$ $(a = 10, b = 3, q = 3, r = 1)$.
$10 = 2 \times 5 + 0$ $(a = 10, b = 2, q = 5, r = 0)$.
$10 = 7 \times 1 + 3$ $(a = 10, b = 7, q = 1, r = 3)$.
$10 = 12 \times 0 + 10$ $(a = 10, b = 12, q = 0, r = 10)$.

Remarque : Trouver le reste dans une division entière par 2 est facile : le reste de x par 2 est égal à 1 si x est impair, c'est égal à 0 si x est pair.

Passage de la base dix à une base b quelconque

Commençons par les nombres entiers : On procède donc par divisions (entières) successives. On divise le nombre par la base b, puis on divise le quotient obtenu par la base, et ainsi de suite jusqu'a obtention d'un quotient nul (c'est la condition d'arrêt de l'algorithme). La suite des restes obtenus correspond aux chiffres dans la base b, $a_0,\ a_1,\ \cdots,\ a_n$.

Étudions cela sur un exemple.

Exemple

On souhaite convertir $(44)_{10}$ vers la base 2. On effectue la série de divisions suivantes :

$$\begin{array}{rcrcrcl} 44 & = & 22 \times 2 + 0 & \Rightarrow & a_0 & = & 0 \\ 22 & = & 11 \times 2 + 0 & \Rightarrow & a_1 & = & 0 \\ 11 & = & 2 \times 5 + 1 & \Rightarrow & a_2 & = & 1 \\ 5 & = & 2 \times 2 + 1 & \Rightarrow & a_3 & = & 1 \\ 2 & = & 1 \times 2 + 0 & \Rightarrow & a_4 & = & 0 \\ 1 & = & 0 \times 2 + 1 & \Rightarrow & a_5 & = & 1\,. \end{array} \tag{9}$$

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

Le cas des 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 : $(54)_{10} = (110110)_2$, obtenu par divisions.
Partie fractionnaire :

$$\begin{array}{rcrcrcl} 0,25 \times 2 & = & 0,50 & \Rightarrow & a_{-1} & = & 0 \\ 0,50 \times 2 & = & 1,00 & \Rightarrow & a_{-2} & = & 1 \\ 0,00 \times 2 & = & 0,00 & \Rightarrow & a_{-3} & = & 0\,. \end{array} \qquad (10)$$

Il en résulte donc que $(54,25)_{10} = (110110,010)_2$

Cas des bases 2, 8 et 16

Ces bases correspondent à des puissances de 2 ($2 = 2^1$, $8 = 2^3$ et $16 = 2^4$), 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 données 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 :

$$(10011011)_2 = (1001\ 1011)_2 = (9B)_{16} \qquad (11)$$

car $(1001)_2 = (9)_{10} = (9)_{16}$ et $(1011)_2 = (11)_{10} = (B)_{16}$. De même, chaque chiffre octal représente 3 bits. Pour passer du binaire à l'octal, on regroupe donc les bits par paquets de trois bits.
On manipule souvent des nombres formés de 8 bits, nommés octets, qui sont donc notés sur 2 chiffres hexadécimaux.

Codage des nombres entiers

La représentation (ou codage) des nombres est nécessaire afin de les stocker et manipuler par un ordinateur. Le principal problème est la limitation de la taille du codage : la valeur d'un nombre peut être arbitrairement grande, mais comme nous travaillons dans un environnement "fini" (les différentes mémoires des ordinateurs sont finies en taille), le codage dans l'ordinateur doit s'effectuer sur un nombre de bits fixé. Les valeurs minimales et maximales des nombres manipulés par les machines sont donc limitées.

Cas des entiers naturels

Les entiers naturels (positifs ou nuls) sont codés sur un nombre d'octets fixé (rappel : 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). 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 $255 = 2^8 - 1$.

La représentation en machine est effectuée de la façon suivante : On représente le nombre en base 2 et on range les bits dans les cases mémoires binaires contiguës correspondant à leur poids binaire, de la droite vers la gauche. Si nécessaire, on complète à gauche par des zéros (bits de poids fort), ce qui évidemment ne change pas la valeur de l'entier !

Exemple

Par exemple, l'entier $n = (144)_{10}$ est représenté par $(0000000010010000)_2$ en base deux sur 2 octets (16 bits). En effet, $(144)_{10} = (10010000)_2$, puis on complète à gauche par des zéros. Noter que ce nombre peut être écrit sur 8 bits mais pas sur moins de 8 bits.

En mémoire nous avons donc :

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

(12)

Cas des entiers relatifs

Les nombres entiers relatifs peuvent être positifs ou négatifs. Ils possèdent donc un signe. En machine nous devons également représenter le signe de ces nombres. Pour ce faire, on utilise le codage en complément à deux, lequel permet d'effectuer ensuite les opérations arithmétiques (addition, soustraction, multiplication, etc.) entre nombres relatifs de la même façon qu'entre nombres naturels. Il faut dissocier deux cas : celui des entiers positifs ou nuls, et celui des entiers négatifs.

Codage en complément à deux : entiers positifs ou nuls

On représente le nombre en base 2 et on range les bits dans les cases mémoires comme pour les entiers naturels. Cependant, la case contenant le bit 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 $2^{n-1}-1$ (puisque le bit le plus à gauche est fixé à zéro).

Codage en complément à deux : entiers négatifs

Rappelons tout d'abord la définition de la valeur absolue. Soit x un entier quelconque (positif ou négatif). On note $|x|$ sa valeur absolue. Elle est définie par

$$|x| := \begin{cases} x & \text{si} \quad x \geq 0\,, \\ -x & \text{si} \quad x \leq 0\,. \end{cases} \tag{13}$$

Par exemple, $|-26| = 26$, $|13| = 13$ et $|0| = 0$.

Codage en complément à deux : entiers négatifs

Pour obtenir le codage sur n bits d'un nombre x négatif, on code en binaire sa valeur absolue sur n bits, puis on complémente (ou inverse) tous les bits et on ajoute 1 (attention aux retenues lors de l'addition binaire !).

Complémenter signifie que l'on transforme les 0 en 1 et *vice et versa*.

Exemple

Soit à coder la valeur -2 sur 8 bits. On exprime la valeur absolue de -2, à savoir 2, en binaire sur 8 bits, soit 00000010. On complémente la séquence 00000010, on obtient donc 11111101. On ajoute 1 (addition binaire) et on obtient comme résultat : 11111110.

Remarque : Lorsque l'on effectue l'addition en binaire $11111101 + 1$, on commence (comme pour les additions en base dix) par calculer $1 + 1 = 10$ (car $(1 + 1)_2 = (2)_{10} = (10)_2$). On pose donc 0 et on retient 1. Puis on calcule $1 + 0 = 1$.

Exercice :

Calculer la somme de $(1101)_2 + (11)_2$ en posant l'addition, puis vérifier votre résultat en passant par la base dix.

Solution :

On calcule tout d'abord $1 + 1$: on pose 0 et on retient 1, puis on calcule $(0 + 1) + 1 = 10$, donc on pose 0, on retient 1, puis $1 + 1 = 10$, ..., on obtient $(10000)_2 = 2^4 = 16$. Par ailleurs $(1101)_2 = 13$, $(11)_2 = 3$, et $13 + 3 = 16$.

Remarque

1. le bit de poids fort d'un nombre négatif est toujours 1 ;
2. sur n bits, le plus grand entier positif est $2^{n-1} - 1 = 011\cdots1$;
3. sur n bits, le plus petit entier négatif est -2^{n-1}.

Introduction

Si on utilise souvent les nombres en informatique, vous conviendrez également qu'il nous arrive de manipuler des mots, des lettres, des caractères. Il faut donc que l'on soit en mesure de les coder sous forme binaire, afin qu'ils soient utilisés par les machines.

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.

Introduction (suite)

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). Notons que le code ASCII original, défini pour les besoins de l'informatique en langue anglaise, ne permet pas 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.

Code ASCII

À 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 zero. La table indique aussi les valeurs en base 10 (décimal) et 16 (hexadécimal) du nombre correspondant.

Code ASCII (suite)

Plusieurs points importants à propos du code ASCII :

1. Les codes compris entre 0 et 31 ne représentent pas des caractères, ils ne sont pas affichables (on dit aussi “imprimables”). Ces codes, souvent nommés caractères de contrôles, sont utilisés pour indiquer des actions comme passer à la ligne (CR, LF), émettre un bip sonore (BEL), etc.
2. 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.
3. On passe des majuscules au minuscules en modifiant le 6ième bit, ce qui revient à ajouter 32 au code ASCII décimal.
4. 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.

Quelques remarques

En base dix, le fait de multiplier un nombre à virgule par 10^n permet de décaler la virgule de n places vers la droite. Par exemple,

1. $23,256 \times 10^1 = 232,56$;
2. $23,256 \times 10^2 = 2325,6$;
3. $23,256 \times 10^3 = 23\,256$;
4. $23,256 \times 10^4 = 23\,325\,60$.

En base deux, c'est exactement la même chose. Si on se donne un nombre, disons $(11,0101)_2$, et qu'on le multiplie par 2^3, on décale la virgule de 3 places vers la droite, et on obtient $(11\,010,1)_2$. C'est ce principe que l'on va utiliser pour la représentation des nombres réels, également appelés nombres à virgules flottantes.

Signe, mantisse et exposant

Un nombre à virgule flottante possèdent un signe $s \in \{+1, -1\}$, une mantisse m et un exposant e. Par exemple pour le nombre $(-101,011)_2 = (-1,01011 \times 2^2)_2$, le signe est -1, la mantisse est le nombre situé après la virgule, soit 01011, et l'exposant est 2. Un tel triplet représente un réel $s \times m \times b^e$ où b est la base de représentation (généralement 2 sur ordinateur, mais aussi 16 sur certaines anciennes machines, 10 sur de nombreuses calculatrices, ou éventuellement toute autre valeur). En faisant varier e, on fait « flotter » la virgule décimale. Généralement, m est d'une taille fixée.

Norme IEEE

Le signe est représenté sur le bit de poids fort (le bit le plus à gauche) s, $+1$ est représenté par 0 et -1 par 1.

1. L'exposant e est codé sur 8 bits (un octet). On code en binaire la valeur $n + 127$, c'est-à-dire $e = n + 127$.
2. La mantisse m est codée sur les 23 bits de poids faibles.

Remarques

1. Les exposants 00000000 et 11111111 sont interdits :
   - l'exposant 00000000 signifie que le nombre est dénormalisé ;
   - l'exposant 11111111 indique que l'on n'a pas affaire à un nombre (on note cette configuration NaN, Not a Number, et on l'utilise pour signaler des erreurs de calculs, comme par exemple une division par 0).
2. Le plus petit exposant est donc -126, et le plus grand +127.