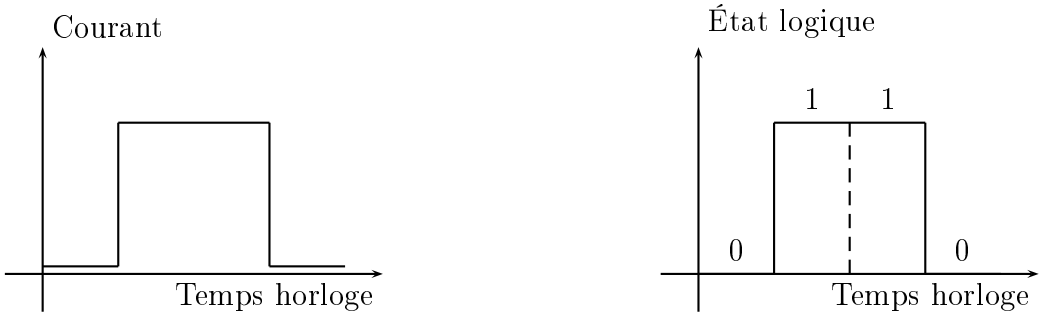
Chapitre 3

Représentation des données

3.1 Représentation machine

Au niveau machine, la présence de courant ou non dans les composants est modélisée naturellement par le langage binaire : courant faible ou absent donne 0 et courant fort ou présent donne 1.



Définition 3.1.

- *Un **bit** (contraction de « binary digit ») est un chiffre binaire i.e. 0 ou 1.*
- *Un **octet** est une suite de 8 bits.*
- *Un **mot** est une suite de 16 bits, donc 2 octets.*

Au niveau architecture matérielle, le « mot » est la taille du bus mémoire, soit la taille de la donnée unitaire capable de transiter entre les zones de stockage mémoire et les registres du processeur central. Cette taille n'est pas normalisée.

3.2 Représentations des entiers naturels

Il n'est évidemment pas possible de coder des nombres de tailles infinies. Les entiers naturels sont donc principalement codés sur 1, 2, 4 ou 8 octets, ce qui donne respectivement 8, 16, 32 et 64 bits. Sur un octet, la plus petite valeur possible est 0 et la plus grande $11111111_2 = 255$, soit au total $2^8 = 256$ possibilités. Sur huit octets (et donc 64 bits), on a 2^{64} possibilités, ce qui représente environ 2×10^{19} (deux milliards de milliards) de nombres.

3.3 Représentations des entiers relatifs

3.3.1 Heuristique

Il n'est pas possible de coder directement un entier négatif en binaire puisque que la machine ne traite que des 0 et des 1. Impossible donc d'écrire -1011_2. Il faut donc une convention pour écrire des entiers négatifs uniquement à partir de 0 et de 1.

Pour comprendre d'où vient et comment fonctionne la convention permettant de coder des entiers relatifs, considérons des entiers codés sur 8 bits. On a vu précédemment que l'on avait 256 nombres possibles ainsi. Si l'on veut coder des entiers relatifs, la moitié des 256 écritures devra correspondre aux négatifs et l'autre moitié aux positifs.

Par ailleurs, observons que

$$11111111_2 + 1_2 = 0_2. \tag{3.1}$$

En effet, comme nous sommes limités à 8 bits, la retenue de la dernière addition ne peut se reporter nul part, elle dépasse la taille de l'encodage et tous les bits précédents sont laissés à 0. On déduit d'une part de l'équation précédente que sur 8 bits

$$-1_2 = 11111111_2.$$

D'autre part que les entiers relatifs ainsi codés forment un groupe cyclique. En effet, en ajoutant 1_2 à 11111111_2 qui est le dernier élément du groupe, on retrouve 0_2, qui en est le premier.

Regardons à présent comment trouver l'opposé d'un nombre codé sur 8 bits grâce à la **méthode du complément à deux** sur deux exemples.

3.3.2 Méthode du complément à deux : exemple 1

Considérons $x = 01101011_2$. La première étape consiste à remplacer tous les 0 par des 1 et réciproquement, on obtient alors le « miroir » de x :

$$\bar{x} = 10010100_2.$$

Remarquons alors que $x + \bar{x} = 11111111_2$. En vertu de 3.1, on a alors

$$x + \bar{x} + 1_2 = 11111111_2 + 1_2 = 0_2.$$

On en déduit que $-x = \bar{x} + 1_2 = 10010101_2$.

3.3.3 Méthode du complément à deux : exemple 2

Considérons maintenant $x = 11001000_2$. À nouveau, on remplace tous les 0 par des 1 et réciproquement, on obtient alors :

$$\bar{x} = 00110111_2.$$

À nouveau on obtient que $x + \bar{x} = 11111111_2$ et en vertu de 3.1 que

$$x + \bar{x} + 1_2 = 11111111_2 + 1_2 = 0_2.$$

On en déduit que $-x = \bar{x} + 1_2 = 00111000_2$.

3.3.4 Méthode du complément à deux : cas général

Pour obtenir l'opposé d'un nombre binaire x codé sur n bits, on commence par écrire son nombre « miroir » $\bar{x}$ en inversant tous les 0 et les 1. Ainsi, comme on peut le voir dans les exemples ci-dessus, on aura nécessairement

$$x + \bar{x} = 1 \dots 1_2.$$

En ajoutant 1_2 à $\bar{x}$, la retenue dépasse la taille de la représentation de x et on retrouve 0_2. On a donc

$$-x = \bar{x} + 1_2.$$

Autrement dit, pour obtenir l'opposé d'un nombre binaire x codé sur n bits, il suffit d'inverser ses 0 et 1 et lui ajouter 1 !

3.3.5 Représentation des entiers relatifs

Au vu de la méthode du complément à deux, il ressort que l'opposé d'un nombre commençant par 0 est un nombre commençant par 1. Ainsi, notre moitié de nombres positifs apparaît naturellement, il s'agit des nombres commençant par 0. Et il en va de même avec les négatifs, il s'agit des nombres commençant par 1.

On a donc la règle suivante : le signe d'un entier codé sur n bits est donné par son bit de « poids fort » ; si le premier bit est 0, c'est positif ; si c'est un 1, c'est négatif.

Sur un octet, on a donc

Représentation		
machine	binaire	décimale
0	0_2	0_{10}
1	1_2	1_{10}
...	...	...
01111111	01111111_2	127_{10}
1000000	-1000000_2	-128_{10}
10000001	-01111111_2	-127_{10}
...	...	...
11111111	-1_2	-1_{10}

On remarquera que -128_{10} fait figure de cas particulier dans le tableau ci-dessus. En effet, son écriture commençant par un 1, elle doit donc représenter un nombre négatif. Comme 0_2 est son propre opposé, on a alors une dissymétrie entre les nombres négatifs et les positifs : il y a un négatif qui n'a pas d'opposé « valable » dans ce système, c'est -128_{10}. « Valable » car on pourra remarquer qu'à l'instar de 0, celui-ci est son propre opposé, mais ça ne fait guère de sens du point de vue binaire ou décimal, seulement machine.

Sur un octet, on peut donc représenter les entier entre -128 et 127. Dans le cas général, pour n bits, on représente les entiers de l'intervalle

$$[\![-2^n\,;2^n-1]\!]\,.$$

3.4 Représentation des réels

Il est évidemment impossible de représenter en binaire des nombres réels quelconques pour des raisons de mémoire. Seuls les rationnels dont la forme irréductible est $\frac{n}{2^q}$, peuvent avoir une représentation exacte ; les autres ont nécessairement une représentation approchée. Par exemple, le nombre décimal $\frac{1}{10}$ a $0,0\,0011\overline{0011}$ comme représentation en base 2, la partie soulignée étant la période, répétée indéfiniment.

3.4.1 Virgule fixe

L'une des premières approches possibles pour représenter un nombre à virgule sur un nombre déterminé de bits est de fixer arbitrairement un nombre de bits réservés à la partie entière et un autre pour la partie fractionnaire. Cependant, cela a de gros inconvénients. Imaginons que sur 8 bits, 4 soient réservés à la partie entière et 4 à la partie fractionnaire. Alors, 111001,01 qui nécessite bien 8 bits ne pourrait pas s'écrire sans voir sa partie entière tronquée ; on ajouterait par ailleurs des zéros inutiles à la partie fractionnaire. Il en va de même avec 0,1101 qui se verrait ajouter des zéros inutiles à ses parties entières et fractionnaires. Pour ces raisons, la notations en virgule fixe n'est pas utilisée.

3.4.2 Virgule flottante

Notation scientifique

Définition 3.2. *On appelle* ***notation scientifique*** *en base b d'un nombre x l'écriture :*

$$x = \pm m b^e,$$

où

- $\pm$ *est le* ***signe*** *;*
- $m \in [1\,;b[$ *est appelée la* ***mantisse*** *;*
- $e \in \mathbb{N}$ *est appelé l'****exposant****.*

Exemples :

- $-0,006234_{10} = -6,234 \times 10^{-3}$ avec pour mantisse $6,234 \in [1\,;10[$ et exposant -3.
- $42789,97 = 4,278997 \times 10^4$ avec pour mantisse $4,278997 \in [1\,;10[$ et exposant 4.
- $10011,1001 = 1,00111001 \times 2^4$ avec pour mantisse $1,00111001 \in [1\,;2[$ et exposant 4.

Remarque : dans la notation scientifique binaire, la mantisse commence nécessairement par un 1. Pour représenter une mantisse en binaire, il suffit donc de représenter sa partie fractionnaire.

Flottant et norme IEEE-754

L'idée est de représenter les nombres en notation scientifique binaire. Il y a donc cette fois-ci trois données à représenter : le signe, la mantisse et l'exposant. Si on représente le signe en codant des nombres négatifs en commençant par un 1 et les nombres positifs en commençant par

un 0 comme on a vu plus, cela nous ramène à deux données à représenter. Ce qui implique donc à nouveau un choix de nombres de bits à attribuer à la mantisse et à l'exposant. On pourrait donc avoir l'impression que le problème reste le même que dans le cas d'une virgule fixe : un manque de précision sur l'une de ces données dû à une limitation du nombre de bits affectées à chacune. Toutefois, cela est nettement moins pénalisant que pour un système à virgule fixe car les exposants sont rarement très grands. Il est dès lors possible d'affecter un petit nombre de bits à l'exposant et un plus grand à la mantisse qui requiert potentiellement plus de précision.

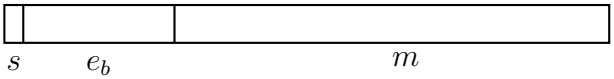
Cette représentation fait l'objet de la norme IEEE-754, proposée par William Kahan (Turing Award 1989), publiée en 1985 et adoptée par la plupart des fabricants d'ordinateurs. Cette norme distingue deux niveaux de précision : simple (sur 4 octets) et double (sur 8 octets).

Prenons l'exemple des flottants en simple précision ; les 32 bits sont répartis en :

— 1 bit de signe s, le bit de poids fort (1 si le nombre est négatif et 0 si le nombre est positif) ;

— 8 bits d'exposant biaisé e_b (11 en double précision) ;

— 23 bits pour la partie fractionnaire de mantisse m (52 en double précision).



Remarque : l'exposant peut-être signé (positif ou négatif), cependant il ne l'est pas par un bit de signe ni l'aide de la méthode du complément à deux. Afin de signer l'exposant, on le biaise en lui ajoutant $b = 2^{n-1} - 1$ où n est le nombre de bits attribués à l'exposant. Ainsi, une fois ce nombre fixé, le biais l'est aussi : en simple précision, le biais vaut $2^{8-1} - 1 = 2^7 - 1 = 127$. Cela donne pour la simple précision

$$e_b = e + b,$$

ou encore

$$\text{exposant biaisé} = \text{exposant} + \text{biais}.$$

Avec un exposant biaisé e_b codé sur 8 bits, on a $e_b \in [\![0\,;255]\!]$ et donc l'exposant $e = e_b - b \in [\![-127\,;128]\!]$. En réalité, les valeurs -127 et 128 étant réservées à des usages spécifiques, on a des exposants biaisés entre -126 et 127.

Exemple : Déterminons la représentation de 1010,11001 en simple précision.

1. Il est positif, son bit de signe sera donc 0.
2. Mettons le maintenant sous notation scientifique : $1,01011001 \times 2^3$. On a donc la partie fractionnaire de la mantisse : 01011001. Toutefois, la partie fractionnaire de la mantisse est représentée par 23 bits, or ici nous n'en avons que 8. On complète donc les bits restants à droite par des zéros :

$$\underbrace{01011001}_{\text{partie fractionnaire}} \ \underbrace{00000000000000}_{\text{complétion à 23 bits}}.$$

3. Il reste à mettre l'exposant sous forme binaire. Il faut d'abord tenir compte du biais :

$$e = e_b - b = 3 \iff e_b = 3 + b \iff e = 130.$$

Il faut donc représenter 130 en binaire : $130_{10} = 10000010_2$.

On a donc

Signe	Exposant	Mantisse
0	10000010	01011001000000000000000

1010,11001 est donc représenté en simple précision par 01000001001011001000000000000000.

3.5 Représentation des caractères

En informatique, chaque caractère est identifié par un code unique qui est un entier naturel. La correspondance entre le caractère et son code est appelé un **Charset**. Toutefois, un code n'est pas utilisable tel quel par un ordinateur qui ne comprend que le binaire. Il faut donc encoder les codes en octets (Encoding).

3.5.1 L'ASCII

Le code ASCII (American Standard Code for Information Interchange) est l'un des plus anciens codes utilisés pour représenter du texte en informatique. Il se base sur un tableau contenant les caractères les plus utilisés en langue anglaise : les lettres de l'alphabet en majuscule (de A à Z) et en minuscule (de a à z), les dix chiffres arabes (de 0 à 9), les signes de ponctuation (point, virgule, point-virgule, guillemet, parenthèses, etc.), quelques symboles et certains caractères spéciaux invisibles (espace, retour-chariot, tabulation, retour-arrière, etc.).

Les créateurs de ce code ont limité le nombre de ses caractères à 128, c'est-à-dire 2^7 , pour qu'ils puissent être codés avec seulement 7 bits : les ordinateurs utilisaient des cases mémoire de un octet, mais ils réservaient toujours le 8 e bit pour le contrôle de parité (c'est une sécurité pour éviter les erreurs, qui étaient très fréquentes dans les premières mémoires électroniques).

Exemple : le caractère « A » est codé en ASCII par le nombre $65_{10} = 41_{16} = 1000001_2$.

Le fait d'utiliser un bit supplémentaire a ouvert des possibilités mais malheureusement tous les caractères ne pouvaient être pris en charge. La norme ISO 8859–1 appelée aussi Latin-1 ou Europe occidentale (ou encore ASCII étendu) est la première partie d'une norme plus complète appelée ISO 8859 (qui comprend 16 parties) et qui permet de coder tous les caractères des langues européennes. Toutefois, ces différents encodages sont source de confusion pour les développeurs de programmes informatiques car un même caractère peut être codé différemment suivant la norme utilisée. Par ailleurs, ils ne règlent pas le problème des symboles des autres alphabets puisqu'il n'est concentré que sur l'alphabet latin.

3.5.2 L'Unicode

La norme Unicode a été créée pour permettre le codage de textes écrits quel que soit le système d'écriture utilisé et donc réglé les problèmes mentionnés ci-dessus (plus d'autres

non mentionnés). On attribue à chaque caractère un nom, une position normative et un bref descriptif qui seront les mêmes quel que soit la plate-forme informatique ou le logiciel utilisés.

Un consortium composé d'informaticiens, de chercheurs, de linguistes, etc, s'occupe donc d'unifier toutes les pratiques en un seul et même système : l'Unicode.

Le répertoire Unicode peut contenir plus d'un million de caractères, ce qui est bien trop grand pour tenir dans un seul octet. La norme Unicode définit donc des méthodes standardisées pour coder et stocker cet index sous forme de séquences d'octets : UTF-8 (8 bits), UTF-16 (16 bits), UTF-24 (24 bits), UTF-32 (32 bits) et leurs différentes variantes.

L'Unicode est une table de correspondance Caractère-Code (Charset) et l'UTF-8 est l'encodage correspondant (Encoding) le plus répandu. Maintenant, par défaut, les navigateurs Internet utilisent le codage UTF-8 et les concepteurs de sites pensent de plus en plus à créer leurs pages web en prenant en compte cette même norme. Voilà pourquoi il y a de moins en moins de problèmes de compatibilité.

3.6 Attendus et savoir-faire

- Évaluer le nombre de bits nécessaire à l'écriture en base 2 d'un entier, de la somme ou du produit de deux nombres entiers.
- Utiliser le complément à 2.
- Calculer sur quelques exemples la représentation de nombres réels : 0.1 ; 0.25 ou $\frac{1}{3}$.
- Identifier l'intérêt des différents systèmes d'encodage.
- Convertir un fichier texte dans différents formats d'encodage.

3.7 Exercices

3.7.1 Démarrage

Exercice 3.1. Dans chaque cas, déterminer si les écritures sur 8 bits suivantes représentent un nombre positif ou négatif puis donner leurs opposés et leurs représentations décimales.

1. 00110100_2.
2. 10100110_2.
3. 01100101_2.

Exercice 3.2. Traduire les nombres suivants représentés sous la norme IEEE-754 simple précision en binaire puis en décimal.

1. 01010001001011011100000000000000.
2. 10101100000101110110000000000000.

Exercice 3.3. Coder en ASCII « I am the Law ! ».

Exercice 3.4. À quel mot correspond le code ASCII

1011001 1101001 1110000 1110000 1100101 1100101 0101101 1101011 1101001 0101101 1111001 1100001 1111001 ?

Exercice 3.5. Peut-on coder « Jafar, j'suis coincé ! » en ASCII ?

Exercice 3.6. Quel est l'encodage utilisé dans cette page Web : auvraymath.net ?

3.7.2 Approfondissement

Exercice 3.7. À l'instar du tableau des entiers relatifs représentés sur 8 bits, dresser celui des entiers relatifs représentés sur 4 bits (représentations machine, binaire et décimale).

Exercice 3.8. On considère le nombre $x = -1039,5_{10}$.

1. Mettre x sous notation scientifique binaire.
2. En déduire la représentation simple précision de x sous la norme IEEE-754.

Exercice 3.9. Mettre les nombres décimaux suivants sous la norme IEEE-754 simple précision.

1. 0,25.
2. 0,1.
3. $\frac{1}{3}$.

Exercice 3.10. [Python] À quoi est égal 0.1+0.2 lorsqu'on demande à la console Python d'effectuer ce calcul ? Pourquoi ? Qu'en déduire sur le test d'égalité de flottants ?

Exercice 3.11. Si le 8^{e} bit en ASCII n'était pas réservé à une sécurité et si on pouvait donc l'utiliser pour l'encodage, combien de caractères pourrait-on encoder ?

Exercice 3.12. Dans un éditeur de texte, enregistrer dans un fichier au format txt la phrase « Jafar, j'suis coincé » en UTF8 puis rouvrir ce même fichier avec des encodages différents. Que constatez-vous ?

3.7.3 Entraînement

Exercice 3.13. Dans chaque cas, déterminer si les écritures sur 8 bits suivantes représentent un nombre positif ou négatif puis donner leurs opposés et leurs représentations décimales.

1. 00110100_2.
2. 10100110_2.
3. 01100101_2.
4. 10001111_2.
5. 10111101_2.
6. 00000111_2.

Exercice 3.14. Traduire les nombres suivants représentés sous la norme IEEE-754 simple précision en binaire puis en décimal.

1. 00001110011010011100000000000000.
2. 11111000011101100010000000000000.

Exercice 3.15. On considère le nombre $x = 105,25_{10}$.

1. Mettre x sous notation scientifique binaire.
2. En déduire la représentation simple précision de x sous la norme IEEE-754.