CI 1 : Architecture matérielle et logicielle

Chapitre 3 – Principe de la représentation des nombres entiers en mémoire

Savoir

Savoirs

– Capacité Dec - C3 : Initier un sens critique au sujet de la qualité et de la précision des résultats de calculs numériques sur ordinateur
  – Principe de la représentation des nombres entiers en mémoire

1 Généralités [1]

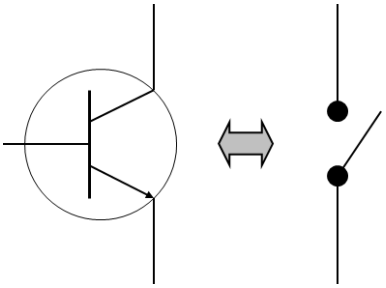
1.1 Logique binaire

Définition

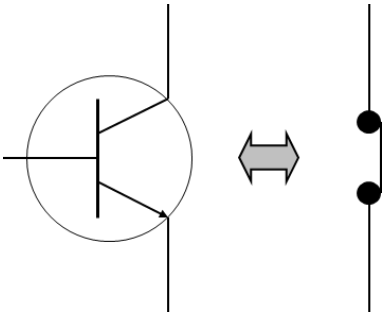
Bit

On appelle bit une information élémentaire de type 0 ou 1 (contraction de l'anglais *binary digit*)

Les systèmes informatiques actuels sont construits à l'aide de circuits intégrés rassemblant pour certains des dizaines voire des centaines de millions de transistors. Ces transistors ne fonctionnent que selon une logique à deux états telle que, de façon schématique « le courant passe » ou « le courant ne passe pas » dans le transistor.



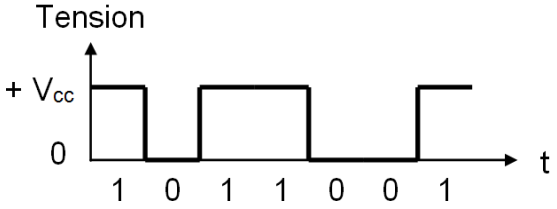
Transistor bloqué
Interrupteur ouvert)
Le courant ne passe pas
État logique : 0



Transistor saturé
(interrupteur fermé)
Le courant passe
État logique : 1

Ces deux états logiques, conventionnellement notés 0 et 1, déterminent cette logique binaire correspondant (de manière un peu « réductrice ») à deux niveaux électriques.

Les informations traitées par les ordinateurs sont de différents types (nombres, instructions, textes, images, sons) mais elles sont toujours représentées en binaire aussi bien en interne, comme on vient de le voir, que sur les « fils » permettant de faire circuler l'information entre les composants de l'ordinateur.



Remarque

On se limitera ici au codage des données numériques (nombres entiers naturels et nombres à virgule).

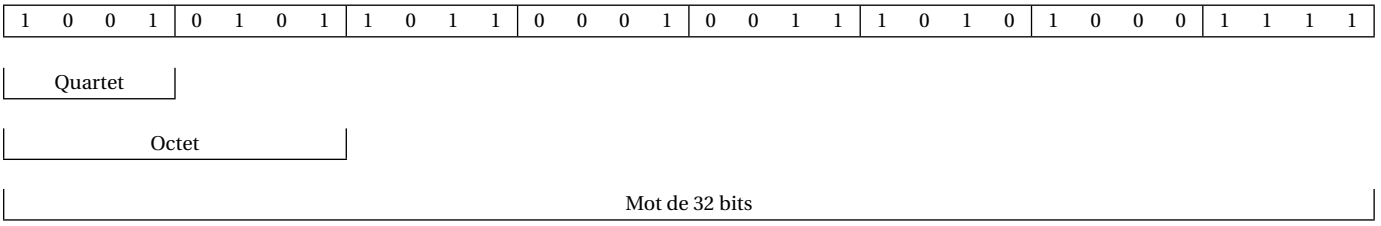
1.2 Notion de mot

Définition

Mot – *Word*

Ensemble de bits de longueur fixe.

Suivant le type de processeur, les mots peuvent avoir 32, 64 ou 128 bits pour les processeurs Intel Itanium.



Remarque

byte est la traduction anglaise du mot octet. En conséquence, un *byte* équivaut à une séquence de 8 bits.

1.3 Notion de pondération

Définition

Écriture d'un nombre dans une base

Dans un système de numération en base B, un nombre noté N_B peut s'écrire sous la forme :

$$N_B = \sum_{k=0}^{n} a_k \cdot B^k$$

s'écrit symboliquement sous la forme :

$$N_B = \underbrace{(a_n a_{n-1} \cdots a_2 a_1 a_0)_B}_{n+1 \text{ chiffres}}$$

On note :

- B : la base ou nombre de chiffres différents qu'utilise le système de numération ;
- a_k : chiffre de rang k ;
- B^k la pondération associée à a_k.

Exemple

$$247_{(10)} = 2 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0$$

On appelle :

- 2 est appelé digit de poids fort (*most significant digit*) ;
- 7 est appelé digit de poids faible (*least significant digit*) ;

$$1001_2 = 1 \cdot 2^{11_2} + 0 \cdot 2^{10_2} + 0 \cdot 2^{1_2} + 1 \cdot 2^{0_2} = 1 \cdot 2^{3_{10}} + 0 \cdot 2^{2_{10}} + 0 \cdot 2^{1_{10}} + 1 \cdot 2^{0_{10}}$$

Remarque

Par la suite, lorsque la base n'est pas précisée, on se considèrera dans le système décimal.

1.4 Les systèmes de numération

1.4.1 Système décimal

Le système décimal est le système universellement utilisé. C'est la base de référence, ce qui signifie qu'un nombre est de manière implicite écrite en décimal, dès ors qu'il est écrit dans précision de base.

1.4.2 Système binaire

C'est la base de numération couramment utilisée en électronique ou informatique. C'est un système en base 2, l'écriture des nombres est donc composée des caractères de 0 et 1.

Exemple

En binaire on compte de la manière suivante :

Base 10	Base 2
0	0000
1	0001
2	0010
3	0011
4	0100

Convertir $(11011001)_2$ *en base 10.*

1.4.3 Système hexadécimal

Ce système à base 16 est le plus utilisé en électronique numérique car il permet une représentation compacte ce qui, dans les systèmes actuels à grande capacité mémoire, est un avantage non négligeable.

Exemple

En hexadécimal on compte de la manière suivante :

Base 10	Base 16	Base 10	Base 16
0	00	10	0A
1	01	11	0B
2	02	12	0C
3	03	13	0D
4	04	14	0E
5	05	15	0F
6	06	16	10
7	07	17	11
8	08	18	12
9	09	19	13

$(2BC5)_{16} = 11205$

2 Représentation des nombres entiers naturels – $\mathbb{N}$

2.1 De la base k à la base 10

Savoir

Savoir-Faire : Représenter en base dix un entier naturel donné en base k.

Pour trouver la représentation en base dix d'un entier naturel donné en base k, on utilise le fait qu'en base k, le chiffre le plus à droite représente les unités, le précédent les paquets de k, le précédent les paquets de $k \cdot k = k^2$, le précédent les paquets de $k \cdot k \cdot k = k^3$, *etc.*

Exemple

Conversion du nombre $(10101)_2$:

$$\begin{aligned}(10101)_2 &= 1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 \\ &= 16+0+4+0+1 \\ &= 21\end{aligned}$$

Exemple

Convertir $(100101)_2$ et $(2C5A)_{16}$ en base 10.

2.2 De la base 10 vers la base k

2.2.1 Méthode des divisions successives

Méthode

Divisions successives

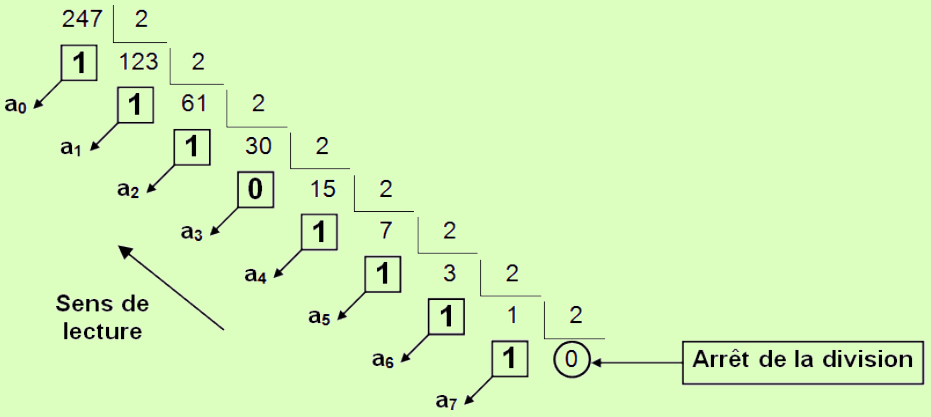
On note $N_{10} = a_n \cdot k^n + a_{n-1} \cdot k^{n-1} + ... + a_1 \cdot k^1 + a_0 \cdot k^0 = a_n a_{n-1} ... a_1 a_0$

1. Calcul de la division de N_{10} par k.
   - Le reste de la division correspond au terme a_0.
2. Le dividende de la division est divisé par k.
   - Le reste de la division correspond au terme a_1.
3. ....

La division s'arrête lorsque le dividende est nul.

Exemple

Conversion de 247_{10} en binaire.



Ainsi $(247)_{(10)} = (11110111)_{(2)}$.

Convertir 247 en base 16.

2.2.2 Méthode de la plus grande puissance

Méthode

Elle consiste à retrancher du nombre initial la plus grande puissance de k possible et ainsi de suite dans l'ordre décroissant des puissances. Si on peut retirer la puissance de k concernée, on note 1 sinon on note 0 et on continue de la sorte jusqu'à la plus petite puissance de k possible soit k^0 pour des entiers naturels.

Exemple

Conversion de 247_{10} *en binaire.*

			⋄		△		
De	247	on peut retirer	0	fois	256.	Il reste	247.
De	247	on peut retirer	1	fois	128.	Il reste	119.
De	119	on peut retirer	1	fois	64.	Il reste	55.
De	55	on peut retirer	1	fois	32.	Il reste	23.
De	23	on peut retirer	1	fois	16.	Il reste	7.
De	7	on peut retirer	0	fois	8.	Il reste	7.
De	7	on peut retirer	1	fois	4.	Il reste	3.
De	3	on peut retirer	1	fois	2.	Il reste	1.
De	1	on peut retirer	1	fois	1.	Il reste	0.

⋄ : lors d'une conversion en base 2, on n'utilise dans cette colonne que des 0 ou des 1.

△ : lors d'une conversion en base 2, on n'utilise dans cette colonne que des puissances de 2 (2^n).

Les valeurs binaires sont lues de haut en bas. On obtient bien le même résultat : $(247)_{(10)} = (11110111)_{(2)}$.

Convertir 247 en base 16.

2.3 Capacités de la représentation

Les limites du codage des nombres entiers naturels sont dues à la longueur du mot binaire nécessaire pour les coder.

Résultat

Si un mot est codé sur n bits, on peut représenter un entier naturel compris entre 0 et $2^n - 1$, soit 2^n valeurs possibles.

Exemple

Si les mots sont codés sur un octet (8 bits), on peut compter de 0 à $2^8 - 1$, c'est-à-dire de 0 à 255.

Bits	Nombre de valeurs	de 0 à ...
4	16	15
8	256	255
16	65 536	65 535
32	4,29... milliards	4,29... milliards
64	$1,84...10^{19}$	$1,84...10^{19}$

Remarque

Dépassement de capacité – *Overflow*

Le résultat de l'addition de deux nombres codés sur le même nombre de bit n'est pas toujours possible car le résultat pourrait demander des bits supplémentaires.

Remarque

En effet, considérons un système où les mots sont codés sur un octet. Calculons $247_{(10)} + 53_{(10)}$

$$247_{(10)} + 53_{(10)} = 300_{(10)} = 1\underbrace{00101100}_{\text{octet retenu}}$$

Ainsi le résultat retenu est $001011100_{(2)} = 44_{(10)}$ au lieu de $300_{(10)}$.

On parle alors de dépassement de capacité (*overflow* en anglais). Sur certains ordinateurs, les calculs continuent. Sur d'autres, une erreur est signalée, d'une façon différente d'un constructeur à l'autre.

Résultat

Addition en binaire

En binaire, l'addition peut être considérée ainsi :

$$\begin{cases} 0_2 + 0_2 = 0_2 \\ 1_2 + 0_2 = 1_2 \\ 0_2 + 1_2 = 1_2 \\ 1_2 + 1_2 = 10_2 \end{cases}$$

On en déduit rapidement que l'addition en binaire s'effectuera sur le même principe que l'addition en écriture décimale, à ceci près que la retenue se produit dès qu'on arrive à 2 (au lieu de 10).

Exemple

Somme de deux nombres binaires

		1	1	1	1	0	1	1	1
$247_{(10)}$	$\rightarrow$		1	1	1	1	0	1	1	1
$53_{(10)}$	$\rightarrow$	+	0	0	1	1	0	1	0	1
			1	0	0	1	0	1	1	0	0

3 Codage pondéré des entiers relatifs – $\mathbb{Z}$

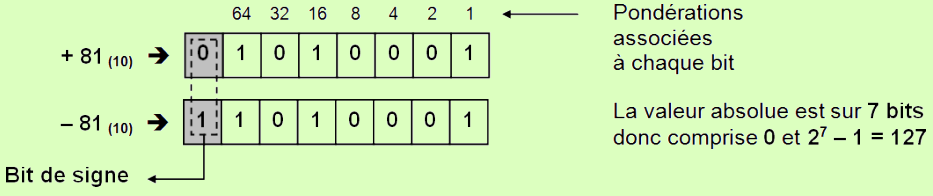
3.1 Première approche

La première solution envisageable pour représenter un entier relatif est de dédier un bit pour le codage du signe puis de représenter sur les autres bits la valeur absolue. La convention retenue impose de mettre le bit de poids fort à 0 pour repérer un nombre positif et à 1 pour un nombre négatif.

On parle de nombres signés quand on utilise cette convention.

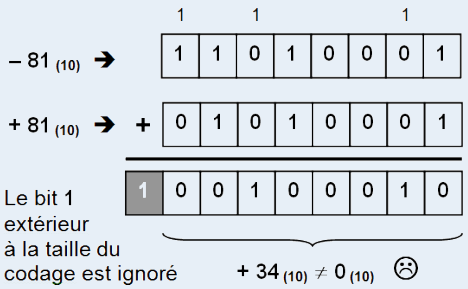
Exemple

Conversion en binaire sur un octet des nombres $+81_{(10)}$ *et* $-81_{(10)}$.



Remarque

Inconvénients : addition binaire de $+81_{(10)}$ *et* $-81_{(10)}$ Cette représentation des nombres signés, si elle est facile à mettre en œuvre, ne permet pas d'utiliser les règles de l'addition binaire pour obtenir un résultat correct. De plus, il y a deux zéros, l'un positif, l'autre négatif.



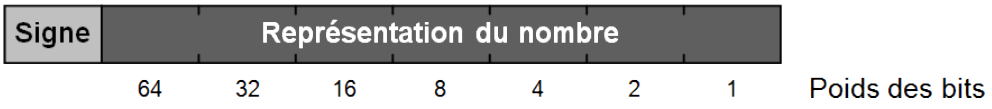
Du fait des problèmes d'arithmétique qu'il pose, ce codage est rarement utilisé.

Cet inconvénient est résolu par l'usage d'une autre forme de représentation des nombres négatifs dit représentation en complément ou représentation sous forme complémentée.

3.2 Codage des entiers relatifs en complément à deux

Les règles suivantes sont adoptées par la majeure partie des constructeurs.

Nombre entier relatif sur un octet (n = 8 bits)



- Signe = 1 : Nombre négatif
- Signe = 0 : Nombre positif
- La valeur zéro est considérée comme un nombre positif.
- Les nombres positifs sont représentés sous leur forme binaire.
- Leur valeur est codée sur n – 1 = 8 – 1 = 7 bits (et non 8 bits).
- Les nombres négatifs sont représentés sous leur forme complément à deux.

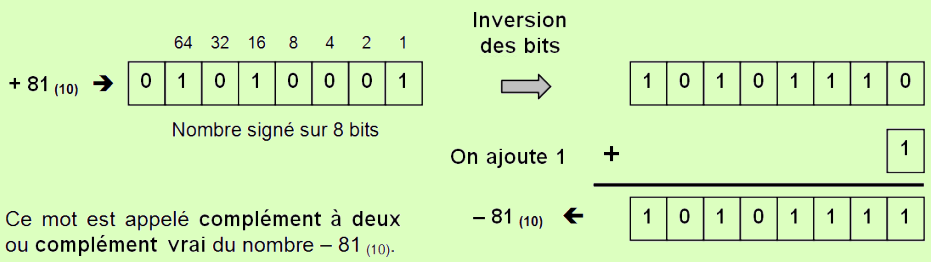
3.2.1 Première méthode pour l'obtention du complément à deux d'un nombre négatif

Méthode

Pour représenter l'opposé d'un nombre positif par son complément à deux, on inverse les bits 0 et 1 et on ajoute 1 au mot binaire obtenu.

Exemple

Complément à deux du nombre $-81_{(10)}$ *sur un octet.*



3.2.2 Seconde méthode pour l'obtention du complément à deux d'un nombre négatif

Méthode

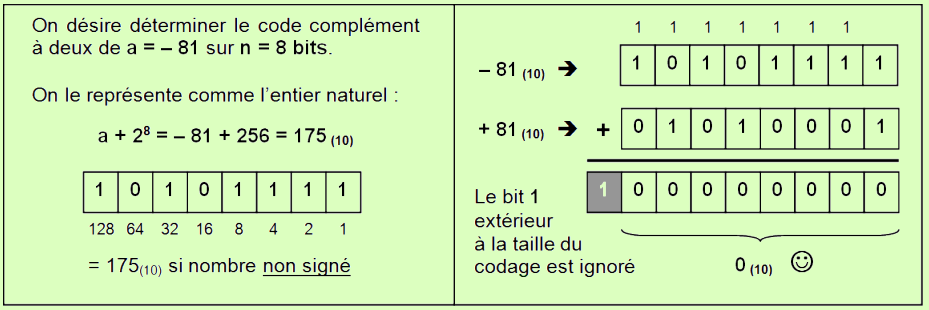
On représente un entier relatif $a \geq 0$ comme l'entier naturel a.

On représente un entier relatif $a < 0$ comme l'entier naturel $a + 2^n$.

Exemple

Complément à deux du nombre $-81_{(10)}$ *sur un octet.*

Vérification de $-81_{(10)} + 81_{(10)} = 0_{(10)}$.



Exercice

Exercice : Trouver les représentations binaires sur huit bits des entiers relatifs 0 et -128.

Exercice

Exercice : Trouver les représentations décimales des entiers relatifs dont les représentations binaires sur huit bits sont 00010111 et 10001100.

3.3 Retour en décimal

Exemple

Trouver la représentation décimale des entiers relatifs dont les représentations binaires sur huit bits sont 01010101 et 10101010.

3.4 Limites de la représentation

Taille du mot	Nombre de bits	Valeurs décimales
n bits	1 bit de signe	0 à $+2^{n-1}-1$
n bits	n-1 bits de valeur	-1 à -2^{n-1}
8 bits	1 bit de signe	0 à + 127
8 bits	7 bits de valeur	-1 à -128
16 bits	1 bit de signe	0 à +32 767
16 bits	15 bits de valeur	-1 à -32 768
32 bits	1 bit de signe	0 à +2 147 483 647
32 bits	31 bits de valeur	-1 à -2 147 483 648

Les limites du codage des entiers relatifs sont principalement dus au dépassement de capacité (overflow) lors d'un calcul. Une addition de deux nombres positifs ou négatifs peut entraîner un dépassement de capacité, celui-ci peut être détecté en regardant le signe du résultat par rapport au signe des deux opérandes (deux nombres positifs donnent un résultat négatif et réciproquement).

Remarque

Dans les versions Python 2.x Lorsque la capacité des entiers machine (32 ou 64 bits) a été dépassée, les nombres sont suivis du marqueur L, qui explicite qu'on passe dans un autre type appelé long. La dernière ligne de l'exemple précédent affiche donc plutôt

```
In [4]: a*a*a*a
Out[4]: 4569760000L
```

En Python 3.x, les types int et long sont fusionnés, et à toutes fins utiles les entiers sont toujours de taille illimitée. Le marqueur L n'est plus utilisé.

Exercice

Représenter les entiers relatifs 99 et 57 en binaire sur huit bits. Ajouter les deux nombres binaires obtenus. Quel est l'entier relatif obtenu ? Pourquoi est-il négatif ?

Trouver la représentation décimale des entiers relatifs dont les représentations binaires sur huit bits sont 01010101 et 10101010.

Quels entiers relatifs peut-on représenter avec des mots de 8 bits ? Combien sont-ils ? Même questions avec des mots de 16 bits, 32 bits et 64 bits.

(À réaliser dans une version Python 2.x) À l'aide de la calculatrice Python, déterminer le plus grand entier représentable par le type int sur votre machine. Vérifiez si votre machine fonctionne en 32 bits ou en 64 bits et calculez la valeur théorique de ce plus grand entier pour vérifier votre réponse.

3.5 Notation hexadécimale

Pour un nombre donné, il faut beaucoup de chiffres en binaire. Dès lors qu'on manipule de grandes séries binaires, on a besoin d'une notation plus concise que le binaire telle que le passage entre elle et le binaire soit très facile. La solution est de faire appel à une base qui soit une puissance de 2. Aujourd'hui, on emploie universellement l'hexadécimal (base $16 = 2^4$).

En notation hexadécimale, on utilise un alphabet comportant 16 symboles (10 chiffres et 6 lettres) :

0 1 2 3 4 5 6 7 8 9 A B C D E F

Méthode

Pour convertir de binaire à hexadécimal, on regroupe les bits par quartet (en ajoutant des 0 à gauche si nécessaire) et on remplace chaque quartet par le symbole hexadécimal correspondant. D'hexadécimal à binaire, on effectue l'opération inverse.

Table de correspondance entre nombres hexadécimaux, décimaux et binaires

$N_{(10)}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$N_{(16)}$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$N_{(2)}$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Exemple

$$3863_{(10)} = \underbrace{1111}_{F} \ \underbrace{0001}_{1} \ \underbrace{0111}_{7}{}_{(2)} = F17_{(16)}$$

Remarque

La représentation des valeurs sous format hexadécimal ne se traduit pas visuellement par la présence de l'indice $_{(16)}$ derrière la valeur. On utilise parfois la lettre H mais on notera le plus souvent la base 16 par la présence d'un $0x$ (comme heXadécimal) devant le nombre. $F17_{(16)}$, $F17H$ ou $0xF17$ sont des représentations valides d'une même valeur hexadécimale.

Références

[1] Christophe François, Représentation de l'information, représentation des nombres.

[2] Manfred GILLI, METHODES NUMERIQUES, Département d'économétrie Université de Genève, 2006.