1] **DELAR Emmalito** 2] **PROCTOR Jordan** 3] **BRISSON Emmanuel**

groupe 2 licence 1 MIASHS 2017-2018

Le codage numérique

l'encodage et le décodage des nombres en binaire

Sommaire

1 Introduction

Ce rapport nous présente les différentes techniques d'encodage et de décodage des nombres (entier naturel, entier relatif, réel) de base 10 en base 2 (=binaire), vues en cours et travaux dirigées d'Introduction à l'informatique.
Pour commencer, nous verrons le principe de codage des nombres entiers naturels, puis des nombres relatifs et pour finir, des nombres réels.

2 Codage d'entier naturel

FIGURE 1 – Nombres entiers

Le codage d'entier naturel est l'un des rares cas où il existe un code canonique [**?**].
Le principe est le suivant : Convertir la valeur entière de base 10, en base 2.
Le système de numération à base 2 ou encore système binaire est un moyen de représenter les nombres avec 2 symboles : 0 et 1.
Selon sa place, le symbole indique la présence d'une puissance de 2 (1) ou non (0).
Exemple : 1101 en binaire vaut 13 en base 10 :

Position	*3*	*2*	*1*	*0*	
Valeur	2^3	2^2	2^1	2^0	
Chiffres du nombre	**1**	**1**	**0**	**1**	
Calcul				$1 \times 2^0 =$	**1**
				$0 \times 2^1 =$	**0**
				$1 \times 2^2 =$	**4**
				$1 \times 2^3 =$	**8**
Total					**13**

FIGURE 2 – Exemple de tableau d'encodage et de décodage binaire/décimal

La figure **??** est un exemple de conversion de base 2 en base 10 [**?**]
Cependant, le stockage de l'entier naturel varie en fonction de la valeur. En informatique, on stocke 1 symbole dans 1 bit. 1 octet contient 8 bits. De ce fait, plus l'entier est grand, plus la conversion en base 2 comportera de symboles et donc utiliserons plus de mémoire.

Pour information :
— 1 octet (byte) code les entier de 0 à 255

— 2 octets (word ou short) code les entiers de 0 à 65 535
— 4 octets (int) code les entiers de 0 à 4 294 967 295
— 8 octets (long) code les entiers de 0 à 18 446 744 073 709 551 615 (18 milliards de milliards)

3 Codage de nombres relatifs

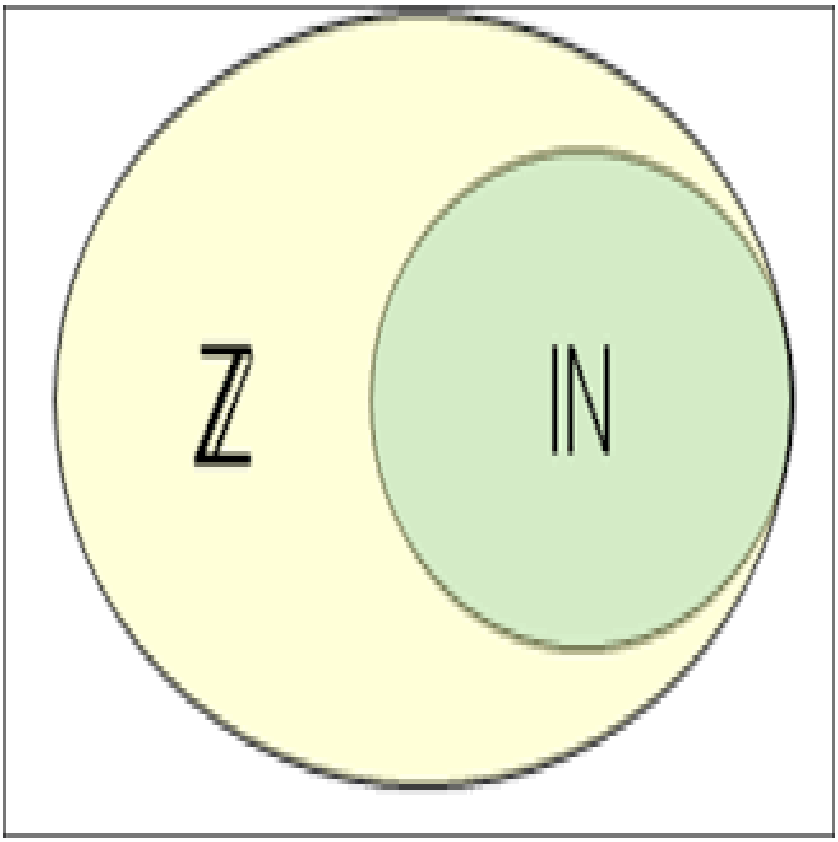

FIGURE 3 – Nombres relatifs

Le principe d'encodage et de décodage des entiers relatifs est le même que celui des entiers naturel sauf que le premier bit est utilisé pour représenter le signe du nombre [**?**] :

— 0 pour le signe plus
— 1 pour le signe moins

Si l'on prend l'exemple d'un octet, on a alors un bit de signe et sept bits pour coder la valeur absolue du nombre. Dans ce cas, un même octet peut représenter un nombre entre -127 et +127.

Exemple de codage sur un octet :

Base 10	Binaire
+82	0 1010010
-82	1 1010010

Les deux représentations de 0 sur 1 octet avec la méthode du bit de signe sont : « 0 0000000 » et « 1 0000000 » (+0 et -0). Ce codage présente deux défauts :

1. L'addition est complexe (il faut traiter quatre cas)
2. Le zéro a deux représentations

Pour ces raisons la méthode du bit de signe seul n'est (presque) plus utilisée.

Pour information :

— 1 octet (byte) code de -128 à 127
— 2 octets (short) code de -32 768 à 32 767
— 4 octets (int) code de -2^{31} à 2^{31}
— 8 octets (long) code de -2^{63} à 2^{63}

4 Codage de réels

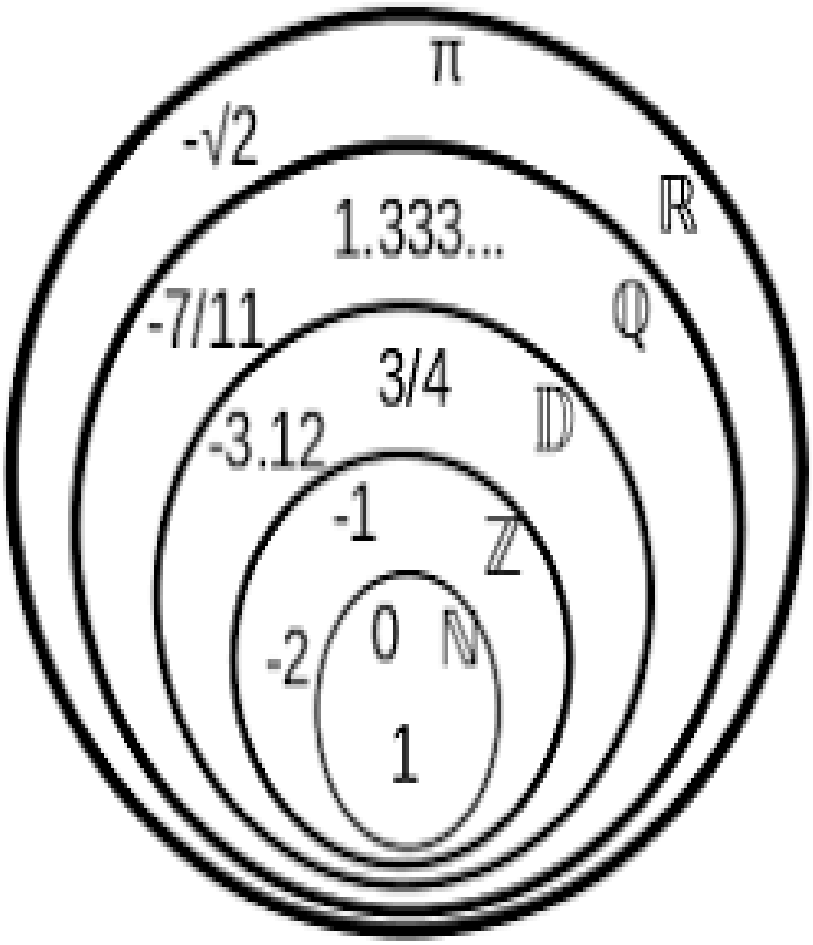

FIGURE 4 – Nombres réels

4.1 Codage en virgule fixe

Principe **L**e principe du codage à virgule fixe est simple : Chaque réel est encodé par 2 entiers : la partie entière et la partie fractionnaire. Pour le codage de la partie fractionnaire, les puissances de 2 seront négatives et on ira de la plus grande à la plus faible puissance. Pour y voir plus claire, rien de mieux qu'un exemple.

10,625 (en base 10) :
$= 8 + 2 + 0.5 + 0.125$
$= 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3}$
$= 1010.101$ (en binaire)

On peut démonter rigoureusement que tout nombre réel positif inférieur à 1 pourrait être écrit de cette manière. [**?**] Resterait à décrire le signe, ce qui peut être fait par un bit particulier (bit de signe) ou par une convention.

Point technique Pour coder la partie fractionnaire d'un réel de base 10 en binaire, on peut multiplier le nombre par deux. Si le produit obtenu est inferieur à 1, on ajoute un 0 à la conversion, sinon on ajoute un 1. Ensuite, on prend la partie décimal du produit et on recommence, ainsi de suite jusqu'à obtenir un produit égale à 1.
Prenons pour exemple le codage de 0,625 :

1. $0.625 * 2 = 1.25 > 1$ On pose : $0,625_{10} = 0,1..._2$
2. $0,25 * 2 = 0,5 < 1$ On pose : $0,625_{10} = 0,10..._2$
3. $0,5 * 2 = 1 = 1$ On pose : $0,625_{10} = 0,101_2$

4.2 Codage en virgule flottante

Base **C**e codage est basé sur la notation scientifique des réels. Notation scientifique en base 10 : Exemple : $123.456 = 1.23456 * 10^2$
Plus généralement : N = $(-1)^S * M * 10^E$, avec S $\in \{0,1\}$ appelé "signe", M $\in$ [1,10[appelé "mantisse" et E $\in$ Z appelé "exposant".
Problème : Comment encoder le zéro ?
Notation scientifique en base 2 : Exemple : $1010.101 = 1.010101 * 2^3$
Standard actuel : Norme IEEE 754 (2008)
N = $(-1)^S * M * 2^{E-B}$, avec S $\in \{0,1\}$, M $\in$ [1,2[, E et B $\in$ N
Rem : B, appelé "biais", sert à obtenir des exposants négatifs (sa valeur est fixée par la norme)
La norme IEEE 754 de 1985, revisitée en 2008, définit 3 format de base [**?**] :

1. Le format simple à 32 bits
2. Le format double à 64 bits
3. Le format quadruple à 128 bits

On a donc :

format	Signe	Mantisse	Exposant	$Exposant_{max}$
32 bits	1 bit	23 bits	8 bits	127
64 bits	1 bit	52 bits	11 bits	1023
128 bits	1 bit	112 bits	15 bits	16383

Principe **U**ne fois après converit en notation scientifique de base 2 le réel, on code le signe (0 ou 1). On code la mentisse, en enlevant le premier bit (qui est forcément 1) par convention. Pour coder l'exposant selon la norme, il suffit de coder l'addition de l'exposant et de l'exposant maximum du format utiliser [**?**] [**?**] .
Pour finir la conversion, les bits codés se rangent dans l'ordre suivant : 1)bit de signe ; 2)bits d'exposant ;
3) bits de mantisse.

Exemple : Coder -18,75 en 32 bits. [**?**]

$18.75_{10} = 10010.110_{2} = 1.0010110 * 2^{4}{}_{2}$

1. Signe = Négatif = 1_{2}
2. Exposant = 4_{10} ; $4 + 127 = 10000011_{2}$
3. Mantisse = $0010110_{2} = 001\,0110\,0000\,0000\,0000\,0000_{2}$ (au format 32 bits)

D'où $-18{,}75_{10} = 1\ 10000011\ 00101100000000000000000_{2}$ selon la norme IEEE 754 au format 32 bits.

Table des figures